ONLINE ANOMALY DETECTION IN THE NEYMAN-PEARSON HYPOTHESIS TESTING FRAMEWORK

by
BAŞARBATU CAN

Submitted to
the Graduate School of Engineering and Natural Sciences
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in
Electronics Engineering

Sabancı University
Jul 2022

ABSTRACT

ONLINE ANOMALY DETECTION IN THE NEYMAN-PEARSON HYPOTHESIS TESTING FRAMEWORK

BAŞARBATU CAN

ELECTRONICS ENGINEERING Ph.D. DISSERTATION, JULY 2022

Dissertation Supervisor: Assist. Prof. Dr. Hüseyin Özkan

Keywords: anomaly detection, Neyman-Pearson, online, nonlinear, classification, large scale, kernel, neural network, context tree, partitioning tree, active learning

We consider the statistical anomaly detection problem with particular attention to false alarm rate (or false positive rate, FPR) controllability, nonlinear modeling and computational efficiency for real-time processing. A decision theoretical solution to that problem can be formulated in the Neyman-Pearson (NP) hypothesis testing (anomaly vs nominal) framework in which the detection power is maximized subject to a user defined upper-bound constraint on the FPR. We first introduce a novel NP classifier (NP-NN) that obtains the NP test in a data driven online manner. The introduced classifier (i) infers an asymmetrical cost structure that matches the desired FPR and (ii) minimizes the resulting empirical risk, i.e., the weighted average of type I and type II errors. These two steps are conducted jointly with a single Lagrangian max objective. A compact and computationally efficient nonlinear modeling is achieved based on a single hidden layer feedforward neural network. The first layer initially expands the kernel space of the radial basis function (RBF) with sinusoidal activation, i.e., Fourier features, the second layer follows to classify, and the overall network is sequentially optimized (subject to the NP constraint) via stochastic gradient descent updates. The data processing in our algorithm is truly online, and appropriate for large scale data applications as it only has linear computational and constant space complexity with respect to the data size. Based on many publicly available and widely used benchmark datasets, our algorithm is experimentally shown to be superior to the state-of-the-art techniques with precise FPR controllability, either by outperforming in terms of the NP classification objec-

tive with a comparable computational as well as space complexity or by achieving a comparable performance with significantly lower complexity.

NP-NN has two disadvantages. (i) It requires extensive cross validations for the RBF bandwidth, making it prone to overfitting (underfitting) if the bandwidth is chosen too small (large), and (ii) it has fixed modeling power since the bandwidth is kept constant, causing a model mismatch, i.e., overfitting (underfitting) in the earlier (later) phases of a data stream. In fact, the modeling power (i.e. the degree of nonlinearity) should be gradually increased in online applications as the learnability gradually improves with increasing data. In order to dynamically control the modeling power and mitigate fitting issues, we additionally propose an ensemble NP classifier (Tree OLNP) that is based on a binary partitioning tree. Tree OLNP creates an ensemble of doubly exponential (in the tree depth) number of partitions of the sample space. Each partition corresponds to an online piecewise linear (hence nonlinear) expert classifier and each expert is essentially a union of linear NP classifiers (OLNP, linear version of NP-NN). Partitions with different number of regions define experts with different modeling powers, and we emphasize that any smooth nonlinear class separation can be modeled by a powerful enough (with sufficient number of pieces) piecewise linear expert. For example, the simplest partition/expert is at the root node with a single region/piece whereas the most powerful one is at the leaves. In this setting, and while maintaining a precise control over the FPR, Tree OLNP generates its overall prediction as a performance driven and time varying weighted combination of the experts in the ensemble. Simpler (more powerful) experts receive larger weights early (late) in the data stream, managing the bias-variance trade-off and hence mitigating the fitting issues as desired. We mathematically prove that, for any stream, our Tree OLNP asymptotically performs at least as well as of the best expert in terms of the NP performance with a regret diminishing in the order $O(1/\sqrt{t})$ (here, t is the data size). Tree OLNP is online and its computational complexity scales linearly with respect to both the data size and tree depth, and scales twice logarithmic with respect to the number of experts. We experimentally show that Tree OLNP outperforms the best expert, and largely improves NP-NN by the introduced dynamical control over the modeling power.

Even when an algorithm is designed online, running it in real time can still be challenging if the data stream is voluminous. To address this challenge of real time processing, we also consider selective processing via active learning. Using the binary entropy as a measure for the uncertainty in the aforementioned expert ensemble for a given sample, we empower our Tree OLNP with the property of processing a sample only if the uncertainty is high. As a result, while preserving the false alarm controllability, a desirable NP performance is achieved in our experiments with significantly lower number of data instances. We finally introduce a system pipeline to detect abnormal objects/actions from video streams. The introduced video processing system receives frames from a live video stream, extracts the YoLo objects, reduces dimensionality with a denoising autoencoder and detects abnormal activities with active Tree OLNP. Our experimental results, on 3 different datasets from the literature and another one from Sabanci University campus, demonstrate that Tree OLNP is a viable option for detecting abnormalities from video streams.

ÖZET

NEYMAN-PEARSON HİPOTEZ TESTİ ÇERÇEVESİNDE ÇEVRİMİÇİ ANOMALİ TESPİTİ

BAŞARBATU CAN

ELEKTRONİK MÜHENDİSLİĞİ DOKTORA TEZİ, TEMMUZ 2022

Tez Danışmanı: Dr. Öğr. Üyesi Hüseyin Özkan

Anahtar Kelimeler: anomali tespiti, Neyman-Pearson, çevrimiçi, nonlineer, sınıflandırma, büyük ölçekli, kernel, sinir ağları, bağlam ağacı, bölümlere ayıran ağaç, aktif öğrenme

İstatistiksel anomali algılama problemini, yanlış alarm oranı (veya yanlış pozitif oran, YPO) kontrol edilebilirliğine, doğrusal olmayan modellemeye ve gerçek zamanlı işleme için hesaplama verimliliğine özellikle dikkat ederek ele alıyoruz. Bu soruna yönelik bir karar teorik çözüm, YPO'da kullanıcı tanımlı bir üst sınır kısıtlamasına bağlı olarak, algılama gücünün en üst düzeye çıkarıldığı Neyman-Pearson (NP) hipotez testi (anomaliye karşı nominal) çerçevesinde formüle edilebilir. İlk olarak, NP testini veri odaklı çevrimiçi bir şekilde elde eden, özgün bir NP sınıflandırıcısı (NP-NN) tanıtıyoruz. Tanıtılan sınıflandırıcı (i) istenen YPO ile eşleşen asimetrik bir maliyet yapısı çıkarır ve (ii) ortaya çıkan ampirik riski, yani tip I ve tip II hataların ağırlıklı ortalamasını, en aza indirir. Bu iki adım, tek bir Lagrange maks hedefiyle birlikte yürütülür. Tek bir gizli katman ileri beslemeli sinir ağını baz alan, kompakt ve hesaplama açısından verimli, doğrusal olmayan bir modelleme elde edilir. İlk katman başlangıçta sinüzoidal aktivasyonla radyal tabanlı fonksiyonunun (RTF) çekirdek alanını, yani Fourier özelliklerini, genişletir, onu izleyen ikinci katmanda sınıflandırma yapılır ve genel ağ, stokastik gradyan inişi güncellemeleri ile (NP kısıtlamasına altında) sıralı olarak optimize edilir. Algoritmamızdaki veri işleme, gerçekten çevrimiçidir ve veri boyutuna göre yalnızca doğrusal hesaplama ve sabit alan karmaşıklığına sahip olduğundan, büyük ölçekli veri uygulamaları için uygundur. Kamuya açık ve yaygın olarak kullanılan birçok kıyaslama veri setinde algoritmamızın, en son teknoloji tekniklerinden üstün olduğu deneysel olarak, hassas YPO

kontrol edilebilirliğinde, karşılaştırılabilir bir hesaplama ve uzay karmaşıklığıyla NP sınıflandırma hedefi açısından daha iyi, veya önemli ölçüde daha düşük karmaşıklıkla karşılaştırılabilir bir performans elde edilerek gösterilmiştir.

NP-NN'nin iki dezavantajı vardır. (i) RBF bant genişliği için kapsamlı çapraz geçerlilik sınaması gerekir, bu da bant genişliği çok küçük (büyük) seçilirse modeli aşırı uyumlamaya (yetersiz uyumlamaya) eğilimli hale getirir ve (ii) bant genişliğinin sabit tutulması da, model uyumsuzluğuna, yani bir veri akışının ilk (son) aşamalarında fazla uyumlamaya (yetersiz uyumlamaya) neden olduğu için sabit modelleme gücüne sebep olmaktadır. Aslında, artan verilerle öğrenilebilirlik giderek arttığından, çevrimiçi uygulamalarda modelleme gücü (doğrusal olmama derecesi) kademeli olarak artırılmalıdır. Modelleme gücünü dinamik olarak kontrol etmek ve uyumlama sorunlarını azaltmak için, ikili bölümleme ağacına dayalı bir topluluk NP sınıflandırıcısı (Tree OLNP) öneriyoruz. Tree OLNP, örnek uzayında, ağaç derinliğinin iki kat üslü sayıda bölüntüler topluluğu oluşturur. Her bölüntü bir çevrimiçi parçalı doğrusal (dolayısıyla doğrusal olmayan) uzman sınıflandırıcıya karşılık gelir ve her uzman esasen doğrusal NP sınıflandırıcılarının bir birleşimidir (OLNP, NP-NN'nin doğrusal versiyonudur). Farklı sayıda bölgeye sahip bölüntüler, farklı modelleme güçlerine sahip uzmanları tanımlar ve vurgulamak isteriz ki herhangi bir düzgün doğrusal olmayan sınıf ayrımı, yeterince güçlü parçalı doğrusal bir uzman tarafından (yeterli sayıda parça ile) modellenebilir. Örneğin, en basit bölüntü/uzman tek bir bölge/parça ile kök düğümdeyken, en güçlü uzman yapraklardadır. Bu ayarda ve YPO üzerinde hassas bir kontrol sağlarken, Tree OLNP, topluluktaki uzmanların performans odaklı ve zamana göre değişen ağırlıklı bir kombinasyonu olarak son tahminini oluşturur. Daha basit (daha güçlü) uzmanlar, veri akışında erken (geç) daha büyük ağırlıklar alır, yanlılık-varyans dengesini yönetir ve dolayısıyla uyum sorunlarını istenildiği gibi azaltır. Herhangi bir akış için, Tree OLNP'nin asimptotik olarak, NP performansı açısından, $O(1/\sqrt{t})$ (t veri boyutudur) ile azalan pişmanlık ile, en az en iyi uzman kadar iyi performans gösterdiğini matematiksel olarak kanıtlıyoruz. Tree OLNP çevrimiçidir ve hesaplama karmaşıklığı, veri boyutuna ve ağaç derinliğine göre doğrusal, uzman sayısına göre de iki kez logaritmik olarak ölçeklenir. Deneysel olarak Tree OLNP'nin en iyi uzmandan daha iyi performans gösterdiğini ve modelleme gücü üzerinde tanıtılan dinamik kontrol ile NP-NN'yi büyük ölçüde iyileştirdiğini gösteriyoruz.

Bir algoritma çevrimiçi olarak tasarlansa da, algoritmayı hacimli veri akışı için gerçek zamanlı koşturmak zor olabilir. Gerçek zamanlı işlemenin bu zorluğunu çözmek için, aktif öğrenme yoluyla seçici işlemeyi de değerlendiriyoruz. Bir örnek için yukarıda bahsedilen uzman grubundaki belirsizliği ölçmek için ikili entropiyi kullanarak, Tree OLNP'yi yalnızca belirsizlik yüksekse bir örneği işleme özelliği ile güçlendiriyoruz. Sonuç olarak, YPO kontrol edilebilirliğini korurken, deneylerimizde önemli ölçüde daha az sayıda veri örneği kullanarak, istenen bir NP performansı elde ediyoruz. Son olarak, video akışlarından anormal nesneleri/eylemleri algılamak için bir sistem sunuyoruz. Tanıtılan video işleme sistemi, canlı bir video akışından kareler alır, YoLo nesnelerini çıkarır, gürültü giderici özkodlayıcı ile boyutluluğu azaltır ve aktif Tree OLNP ile anormal faaliyetleri tespit eder. Sabancı Üniversitesi kampüsünden topladığımız set ve kamuya açık 2 veri seti üzerindeki deneysel sonuçlarımız, Tree OLNP'nin video akışlarından anormallikleri tespit etmek için uygun bir seçenek olduğunu göstermektedir.

ACKNOWLEDGEMENTS

I thank to my advisor Assist. Prof. Dr. Hüseyin Özkan for his guidance and effort throughout my doctoral study. His brilliant suggestions and endless support have invaluable impact on the completion of this thesis.

I would like to thank Prof. Dr. Özgür Gürbüz, Prof. Dr. Levent Arslan, Assist. Prof. Dr. Sinan Yıldırım and Assist. Prof. Dr. Erdem Akagündüz for their evaluation of my thesis.

I thank my employer GE and especially to my colleague Tuğba Halıcı for her immense support.

I thank my friends Görkem Argalıoğlu and Türkan Merve Argalioğlu for their support and patience throughout this journey. They were very helpful in reminding me my priorities and keeping me in the correct path.

I am grateful to my family for their love and support throughout my studies. They were always with me when I needed them.

Finally, I thank The Scientific and Technological Research Council of Turkey (TUBITAK) for their support of my study under Contract 118E268.

Dedicated to my family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Anomaly detection refers to targeting the sample points which do not represent the expected behavior of a population [1]. These deviating samples are important as they often provide actionable information. One example can be an intrusion event in a network, where the event may be useful to identify a weak spot [2]. Another example is the event of an accident in a vehicular traffic, where the fast detection of the event (or foreseeing, if possible) is crucial for treating the injuries timely (or even preventing them to happen) [3].

A direct approach for anomaly detection simply suggests the analysis of all normal instances and investigate on the ones that are outliers, or less likely to be from the normal population. In particular, the extreme rarity of the anomalous events provides (from a data driven machine learning perspective) a training problem that is difficult to handle. Hence, it is important to understand the characteristics of the anomalies to better investigate the underlying statistics. In general, there are three types of anomalies: point anomaly, contextual anomaly and collective anomaly [4].

Point anomaly is a single data point that is statistically deviant from the nominal distribution, yielding perhaps the simplest anomaly type. Fraud detection in credit card payments, as explained in [5], is an example, where each instance in the dataset has only one variable, i.e., expense amount. In this example, anomaly is defined as a single data instance, namely, a single payment, that is typically much higher in number than the rest of the population.

Contextual anomalies is a data point that is statistically different compared not to the overall nominal distribution but to the corresponding context which can be defined as the distribution of a certain fraction of the nominal instances. For example, one can consider the measurements of the amount of traffic in a city with time stamps. Although observing high traffic jam in the middle of the day is expected, if the same situation occurs at night, then it can be considered as

anomaly. In this example, time is the contextual variable. These anomalies are also referred to as conditional anomalies [6].

Collective anomalies provide a certain set of data points in which an individual data point is not necessarily an anomaly but the set is [7]. For example, amount of data transfer from one source to another can occasionally deviate from the typical transfer rate, which is certainly worth attention as it might signal a cyber attack.

One can find many more examples for each of the aforementioned anomaly types; for instance, in the field of hyper-spectral imaging, where the detection algorithms exploit the information collected as the intensities of the wide band electromagnetic waves emitted from the environmental materials to identify the objects of interest [8]. RX anomaly detector [9] is a widely used anomaly detection algorithm for hyper-spectral imaging. It assumes the distribution of the background as Gaussian and calculates the Mahalanobis distance [10] between the background clutter and the observed image to generate an anomaly score. However, normal distribution for background is not always accurate. Therefore, the model fails in more complicated scenarios. To overcome this problem, in [11], kernel RX algorithm is explained, which is capable of modelling non-Gaussian backgrounds for more realistic data modeling.

Another important challenge in anomaly detection is the possible drift in the normal behaviour [1]. Even though it is not applicable when working with the statistically stationary datasets, it is a well known problem for most of the daily generated data. In addition to the drifts (time varying or nonstationary source statistics) in the normal population, nonstationarity might well be a subject to the distribution of anomalies [4]. A number of methods focusing on detecting anomalies in the non-stationary settings are available and their applicability heavily depends on data dynamics [1]. In most of the real life applications, it is important to detect anomalies in real time and be able to adapt to the time varying statistics. In [12], an online anomaly detection approach using relative entropy and Pearson correlation is introduced. In [13], an online recurrent neural network (RNN) model is used to train the time series data, where the model is capable of detecting anomalies and change points.

By nature, anomaly detection (from the data driven machine learning perspective without strong statistical assumptions) requires to use datasets that are heavily imbalanced due to the rarity (i.e., low frequency) of an anomaly. For example, in a real-time monitored engine health system, an anomalous event can be the failure of

an engine which is usually rare to encounter and costly to overcome [14]. Hence, the dataset (that is available to infer statistical results from) may or may not contain anomalous observations. *Additionally, as the goal of the model is to implement an alarming system for the anomalous events, it is crucial to control the false alarm rate for credibility.* To address the unavailability of anomalies and the need for controlling the false alarm rate, one solution is to estimate the minimum volume set covering at least α percent of the given normal instances (without any anomaly examples) to guarantee $1-\alpha$ (freely specified by the user) percent false alarm rate [13]. One can estimate and use the false positive rate (FPR) and true positive rate (TPR, based on available anomalous instances or synthetically generated data or by treating one of the classes as anomaly in binary classification) for a performance measure (via the area under curve of the receiver operating characteristics), and tune afterwards to meet the application specific requirements. On top of these, [15] defines another measure by combining FPR and TPR using Neyman-Pearson (NP) detection framework, where the goal is to maximize the detection power under a certain user-specified false alarm rate [15, 16]. Models optimized using NP framework are especially appealing because of the direct control over the false alarm rate allowing (to the desired degree) the suppression of untrue alarms.

We emphasize that depending on the availability of anomalous examples, and in the context of false alarm rate controllability, one can define two possible scenarios: 1) There exist no available anomalous examples, and all the available data is from the nominal distribution. In this case, the problem is a one-class classification problem which can be addressed by the approach of finding the minimum volume set that is essentially binary hypothesis testing, where the anomaly class is assumed to be uniformly distributed in the support of the nominality. Hence, one can sample from the uniform density and the problem can be turned into supervised NP classification. 2) There exist available anomalous examples. In this case, if the label information is available, then the problem is again supervised binary NP classification; and if the label information is unavailable then the problem is unsupervised binary NP clustering which is out of the scope of the presented thesis study.

In this thesis, thanks to its generality (whether anomalous examples are available or not), we consider anomaly detection as an NP classification problem. NP approach has a long history in research [17] and NP classification is still an active area in the context of anomaly detection [18, 19, 20, 21, 22].

1.1 Neyman-Pearson (NP) Classification in Literature

NP detection framework is different from the regular classification problem because of the unequal costs assigned to the errors (which are typically equal in regular classification), and determining those unequal costs for matching the desired false alarm rate is generally non-trivial. This unequal cost assignment makes the NP approach appealing especially for applications generating imbalanced data as in the case of anomaly detection. For example, in medical diagnosis, the type II error (misdiagnosing a person as healthy) has perhaps more severe consequences than the type I error (misdiagnosing a person as unhealthy). The first case may lead to loss of life whereas the the latter is not life threatening but may result in devastating psychological effects. To the goal of controlling the both and optimizing with respect to the requirements of the specific application in hand, the NP framework can be used to guarantee to achieve the maximum detection power (minimum false negative rate) while maintaining a user-defined fixed false alarm rate (high credibility).

An initial theoretical approach to the NP classification problem is provided in [23], which designs a classifier under the NP and min-max criteria. In a study [24], radial basis function neural network (RBFNN) is used to define a model for binary classification. Maximum detection is achieved by grid searching the radial widths as hyperparameters used in each hidden node. Classification threshold is selected empirically to satisfy the desired false alarm rate. Prior to [25], NP research assumes that the underlying class distributions were known or (in special cases) the likelihood ratio is monotonic with respect to a threshold parameter [26, 17, 27] which is tuned to achieve the desired false alarm rate. In [25], a non-parametric version of the problem is introduced by studying empirical risk and structural risk minimization in the NP context.

Imbalanced datasets intrinsically introduce asymmetrical class costs, and as a result of this along with the deviation of the desired rate and the observed in practice, evaluation of different NP classifiers should better not depend only on regular metrics such as the area under the receiver operating characteristics (ROC) curve [28]. In [15], a new evaluation metric for NP classifiers which is also applicable to any anomaly detection, was introduced as

$$\text{NP score} = \frac{1}{\tau}\max(\text{FPR}, \tau) - \text{TPR}, \tag{1.1}$$

where FPR is the observed false positive rate, TPR is the observed true positive

rate and τ is the desired false positive rate.

Remark: Note that an NP classifier is trained to match a user-specified false alarm rate. However, for several reasons such as the limited data or convergence issues, one should obviously observe small mismatches in practice at the end of the processing between the target false alarm and what actually is observed, leading to the definition of "observed false alarm rate". This mismatch or deviation from the target false alarm rate that is to be upper-bounded should be incorporated into the performance measure for evaluating any NP classifier (in addition to, of course, the detection power that is to be maximized). Hence, (1.1) has been introduced in [15]. Intuitively, this evaluation penalizes the model proportional to the percentage deviation from the target false alarm rate. We will further discuss this metric and its variations in Chapter 2.

Many existing studies in NP classification (such as [18, 22]) are based on the kernel support vector machines (SVM) [29]. SVM contains two steps. In the first step, a data instance $\mathbf{x}$ is transformed with the mapping $\Phi : \mathbb{R}^d \to \mathcal{H}$, where $\mathcal{H}$ is the high dimensional Hilbert space implied by the employed kernel [30]. In the new space after the transformation, classifier parameters $\mathbf{w}$ and b are learned which minimizes the linear separation error between the two (in the binary case) classes. This is formulated as [1]

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, \tag{1.2}$$

$$\text{s.t. } y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle_{\mathcal{H}} + b) \geq 1 \quad \text{for } i = 1, ..., n, \tag{1.3}$$

where the two classes are separated by the margin of $\frac{2}{\|\mathbf{w}\|^2}$. This problem can be solved via its Lagrangian dual via the Karush-Kuhn-Tucker (KKT) conditions [31]. Using these conditions, one can solve and obtain $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i)$. Note that here, training is dependent on the data only via the corresponding pair-wise inner products $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle$. This inner product can be considered as the similarity (or affinity) between data instances $\mathbf{x}_i$ and $\mathbf{x}_j$. Hence, the calculations in the high dimensional space are avoided by only using the similarities given by a kernel encoding the required inner products. This is also known as the kernel trick [32]. At the end of the training, the linear classifier obtained in the high dimensional kernel space corresponds to a nonlinear classifier in the original feature space. Theoretically, one can solve any linear or nonlinear classifier with SVM, thanks to the kernel trick

[1] In the scope of this thesis, class labels of the NP classification problem are defined as $y \in \{1, -1\}$. An example can be health status representation of the individuals in a population, where -1 and 1 are healthy (normal) and unhealthy (anomalous) individuals, respectively (from the perspective of a certain condition, e.g., diabetes status). This label representation is also applicable to the anomaly detection problem because it is more likely to observe more healthy individuals. We can also define a relation between the prior distributions of the classes as $P(Y = -1) >> P(Y = 1)$.

(one should use the inner product in the original feature space as the kernel for linear solution, thus, without mapping to high dimension) under Mercer's conditions [33, 30]. However, the end result, i.e., performance, would be heavily dependent on the choice of the kernel.

As mentioned before, many existing NP classification studies use the SVM formulation. Prominent examples are [18, 34, 19, 35, 36, 20, 21, 37, 22] can be divided into 3 groups.

Cost Sensitive Learning In this learning approach, different costs are assigned to the classes and the learning algorithm is updated accordingly, i.e., in a way to find the correct pair of costs yielding the right weighting for matching the desired false alarm rate. One of the important studies along this line of research is in [18]: the primal SVM problem is defined with the asymmetric cost assignment as

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\gamma \sum_{i\in I_+}^{b} \xi_i + C(1-\gamma) \sum_{i\in I_-}^{b} \xi_i, \tag{1.4}$$

$$\text{s.t.} \quad y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle_{\mathcal{H}} + b) \geq 1 - \xi_i \quad \text{for } i = 1, ..., n, \tag{1.5}$$

$$\xi \geq 0, \tag{1.6}$$

where C is the error cost of SVM, $\gamma \in [0,1]$ is used to split the cost between negative and positive classes and ξ is the slack variable (for relaxing the constraint in the optimization problem and to extend to the linearly inseparable datasets) of SVM. The error cost C is split via λ as class specific costs as $C_+ = C\gamma$ and $C_- = C(1-\gamma)$. For example, an NP detector for a radar application is formulated in [34] using (1.4). The parameters C and γ are adjusted to meet the desired user-defined false alarm rate: if the observed false alarm rate (FPR) turns out higher (lower) than the desired rate τ, then C_- is increased by reducing γ (reduced by increasing γ) and the model is re-trained, which defines an iterative training process until the the mismatch vanishes sufficiently (Recall that $C_+ = C\gamma$ and $C_+ = C(1-\gamma)$). Adjustment of the parameters is performed via grid searching in the hyperparameter space. In another study [19], it is argued about the unspecificity of these costs; and learning the cost interval rather than the cost itself is proposed. In this cost interval learning, the parameter C in (1.4) is redefined as $C = 0.5(C_{min} + C_{max}) + \varepsilon$, where ε is a uniformly distributed random variable defined in interval $[-d, d]$ and $\|\varepsilon\| \leq d$, which allows the cost to have multiple values. Hence, compared to the learning of the exact values of the unequal class costs, multiple values assigned for the cost relaxes the problem and allows better convergence to the desired false alarm rate [19]. However, even though the cost is determined within an interval, the indirect relation between the

class costs and desired false alarm rate unnecessarily complicates the problem.

Another approach is the cost sensitive boosting of the classifiers [35]. It is mainly similar to the well known boosting algorithm adaboost [38], where the resulting weighted sum of the weak classifiers converges to the best possible model [39]. In [35], losses used in boosting are redefined as binomial or exponential to make the learning cost sensitive. However, the relation between the cost distributions and the desired false alarm rate is still unclear and remains unknown, where the evaluations are conducted empirically.

Thresholding Methods In this approach, a classifier is trained and an appropriate threshold matching the desired false alarm rate is estimated. A comparison of a typical error minimizing classifier and an NP classifier is shown in Fig. 1.1, where the estimated threshold 1.65 in the NP classifier guarantees the desired false alarm rate 0.05. An example of the explained method is implemented in [24], where a radial basis function neural network is trained and empirically selected (i.e. manually adjusting the parameter in the interval of interest) threshold is used to match the NP constraints. In another study [36], an SVM based spam filter is implemented. They fixed the miss rate (i.e. not detecting the spams) and tried to estimate the threshold to match the desired false alarm rate. Since the threshold necessary to achieve user defined false alarm rate is selected manually, model does not consider any class related costs in the training. Hence, the problem depends on the proper search of the threshold which may be a difficult problem depending on the given data. One of the comprehensive studies [20] introduces an umbrella algorithm for the NP classification paradigm, which is capable of using built-in classifiers such as logistic regression, support vector machine and random forests. Therefore this umbrella algorithm can handle linear and nonlinear data depending on the base classifier. Each previously discussed algorithm do not start optimizing the problem under the NP constraints. The trained classifier does not directly consider the target false alarm rate therefore it is optimized as a typical error minimizing classifier.

There exists another line of research introducing a more sophisticated approach based on an estimated scoring function. One example is explained in [40] as an application to anomaly detection, where a k-nearest neighbour graph (K-NNG) is generated using the nominal data. Generated graph contains a score for each sample, estimating its quantile to meet the false alarm constraints. Although the resulting algorithm is truly (with empirical approximations) an NP classifier, the problem with this method is the complexity of K-NNG (i.e. score generation function), which is $O(N^2)$ (N is the number of processed instances). Therefore,

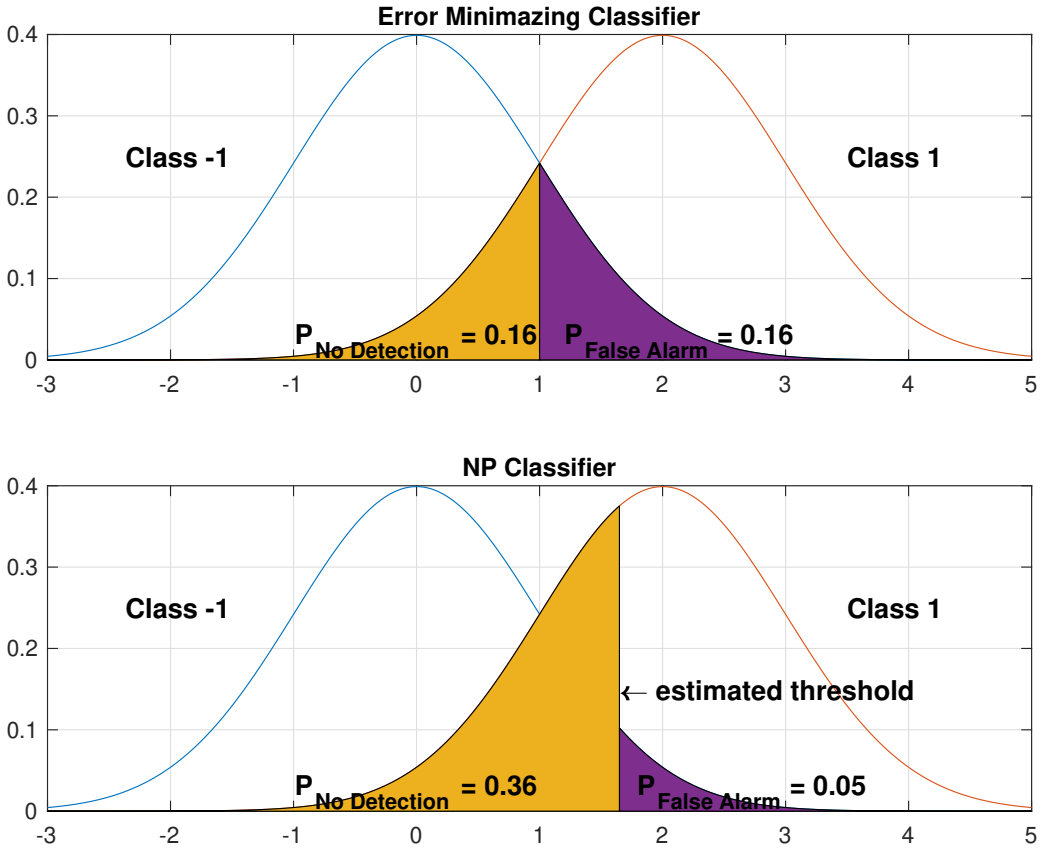


Figure 1.1 A typical error minimizing classifier vs the NP classifier: Two classes of data are represented as two Gaussian distributions with the same variance but different means as $\mathcal{N}(0,1)$ and $\mathcal{N}(2,1)$. Suppose that the user-defined upper bound for the false alarm rate is $\tau = 0.05$. The error minimizing classifier select the threshold as 1. It results in type I error $= 0.16$ which is greater than the desired false alarm rate. On the other hand, NP classifier optimizes the threshold as 1.65 ensuring the maximum detection at the specified false alarm rate.

it does not scale well with increasing data. In another study [21], this scalability problem is addressed by introducing a linear time projection algorithm in the learning of the scoring function. In this study, a linear classifier is used and the dual problem of its parameter estimation is solved via projections in linear time, (which is similar to the online NP classifiers of this thesis complexity-wise) but it is still not an online algorithm as multiple passes over the data is required in the phase of the projection. Also, they only study linear classifiers (hence the result is certainly not an NP classifier when the problem is nonlinear) and leave the nonlinear extension to a kernel use which would require quadratic complexity and certainly hinder again the online/efficient processing.

Constrained optimization The NP formulation can be considered as a constrained optimization problem, where the constraint is upper-bounding the false alarm rate (with the objective of detection power maximization). In [37], a generalization of the online stochastic convex optimization to the case of multiple objectives is explained. In the optimization problem with multiple objectives, tracking the quality of the optimization is not straightforward because it is hard to assign appropriate weight to each objective. Hence, [37] proposes a function defining the optimization constraint based on one objective and tries to bound other objectives by the appropriate thresholds. Even though this solution is not specifically proposed to solve NP classification problem, it is easily adjusted by selecting the corresponding objectives as maximum detection power and upper-bounded false alarm rate. More specific study of the constrained optimization under NP framework is studied in [22] for online and batch classifiers. In this paper, authors introduce no detection and false alarm probabilities as

$$\mathbf{P}_{\text{nd}}(f) = P(f(x) \leq 0 \mid y = 1), \qquad \mathbf{P}_{\text{fa}}(f) = P(f(x) \geq 0 \mid y = -1), \tag{1.7}$$

where $f(x) = w^T x + b$ is the discriminant function with the weight w and the bias b. In the context of NP classification, the $\mathbf{P}_{\text{nd}}$ should be minimized while keeping the $\mathbf{P}_{\text{fa}}$ below a user defined upper-bound. However, it is impossible to know the exact values of the probabilities defined in (1.7) since one can only have empirical estimations based on the given finite datasets. These estimations of no detection and false alarm probabilities can be written as

$$\tilde{\mathbf{P}}_{\text{nd}}(f) = \frac{1}{n_+} \sum_{i \in D_+} \mathbb{I}_{f(x_i \leq 0)}, \qquad \tilde{\mathbf{P}}_{\text{fa}}(f) = \frac{1}{n_-} \sum_{i \in D_-} \mathbb{I}_{f(x_i \geq 0)}, \tag{1.8}$$

where n_+ and n_- are the total number of observed positive and negative samples, $\mathbb{I}$ is the $0-1$ loss function penalizing the misclassified samples, $\tilde{\mathbf{P}}_{\text{nd}}$ and $\tilde{\mathbf{P}}_{\text{fa}}$ are the empirical estimates for true no detection and true false alarm rates, respectively. To obtain a differentiable surrogate for this $0-1$ loss, one can alternatively use the sigmoid:

$$l(z) = \frac{1}{1 + e^{\eta z}}. \tag{1.9}$$

Then, the NP approach can be obtained as

$$\min_f \frac{\lambda_c}{2} \|w\|^2 + \tilde{\mathbf{P}}_{\text{nd}}(f) \;\; \text{subject to} \;\; \tilde{\mathbf{P}}_{\text{nd}}(f) \leq \rho, \tag{1.10}$$

where λ_c is the regularization parameter and ρ is the user-defined upper bound for the false alarm rate. Expanding (1.10) with the empirical definitions of $\mathbf{P}_{\text{fa}}$ and $\mathbf{P}_{\text{nd}}$

leads to the Lagrangian

$$\mathcal{L}(f,\lambda) = \sum_{i=1}^{n} \Big(\frac{\lambda_c}{2} \|w\|^2 + a_i l(y_i f(\mathbf{x}_i) - \lambda \rho \Big), \tag{1.11}$$

where coefficient a_i is defined as $n \,/\, n_+$ for positive samples and $\lambda n \,/\, n_-$ for negative samples. The goal is to minimize the equation (1.11) with respect to f and λ. This is a coordinate-ascent type iterative optimization, where update for one parameter is calculated while the other parameter is fixed, and can be solved with the Uzawa approach [41]. The variable λ is the Lagrangian multiplier and appears in the variable a if the instance is negative. This implies that λ is capable of learning the cost between positive and negative samples. Hence, in (1.11) two sets of parameters are to be optimized: λ for the cost assignment, and f for the class separation under the cost assignment by λ.

In this thesis, we propose a solution for statistical anomaly detection problem focusing on false alarm controllability and scalability. We use Neyman-Pearson framework for false alarm controllability and online learning for scalability. Currently available solutions to Neyman-Pearson classification problem (cost sensitive learning, thresholding methods and constrained optimization) fail to address these features at the same time. Therefore, our goal is to fill this gap in the literature. More detailed description of the thesis contributions are given in the next section.

1.2 Thesis Contributions and Highlights

Main contributions of the thesis can be summarized as follows.

- We propose a novel NP classifier (NP-NN) which is both online and nonlinear as the first time in the literature. NP-NN is capable of processing streaming data in an online manner, maximizing the detection power while upper bounding the false alarm rate by a user-defined threshold. Proposed classifier is based on a single hidden layer feedforward neural network (SLFN). We initialize the network parameters using random Fourier features (RFFs) to construct the kernel space of the radial basis function at its hidden layer with sinusoidal activation. Using RFFs provides great initialization and exponentially decreases the total number of parameters within the network. This compactification mitigates overfitting while improving processing efficiency. We train SLFN with Lagrangian objective with stochastic gradient updates and achieve an online nonlinear NP classifier.

- We propose a novel online context tree based NP classifier (Tree OLNP) containing hierarchically organized NP models. The goal of this approach is to mitigate overfitting caused by the rbf kernel in NP-NN, which has infinite VC dimension (RBF kernel can solve any smooth classification problem if the bandwidth is chosen sufficiently small) [42], using a weighted combination of piecewise linear NP classifiers (i.e. experts). Note that a piecewise linear classifier has finite VC dimension. In addition, in our Tree OLNP approach, (i) nonlinear class separations are approximated piecewise linear and the approximation here can be improved to any desired degree up to the point of no loss by increasing the granularity (by increasing the tree depth), (ii) by using a performance driven nonstationary combination weights in the employed mixture of experts, we achieve a dynamical modeling power that is tuned to the learnability allowed by the available data size. This further mitigates the overfitting issue. Also, (iii) Tree OLNP is online with computational complexity that is linear to the size of data ($O(N)$) as well as the tree depth. Hence, Tree OLNP is scalable to large scale data presenting real time processing capabilities. Lastly, (iv) we provide strong performance guarantees in the deterministic sense without any statistical assumptions. Namely, Tree OLNP is mathematically guaranteed to asymptotically perform (in terms of a designated NP performance measure) at least as well the best expert. To our best knowledge, NP classification acquires these properties first time in the

literature.

- We propose an active learning approach for Tree OLNP in order to speed up the convergence to the best expert performance while maintaining the NP property, detection power maximization with false alarm rate controllability. Proposed active learning method decides on including or excluding an input sample in the learning process by comparing the uncertainity amount (entropy) among its experts with a user defined threshold. With this active extension (selective processing), we further improve the real time processing capability.
- We introduce an anomaly detection pipeline with Tree OLNP, capable of detecting abnormal objects from video streams. Proposed pipeline first detects objects in each from using YoLo object detector [43]. Then detected objects are scaled and fed to the denoising autoencoder proposed in [44] to project higher dimensional images to lower dimensional latent space. Tree OLNP processes projected images in real time and detects anomalies. We evaluate the performance of the pipeline on 3 datasets, where 2 of them are from literature and the last one is created by our group in Sabanci University.
- We present comprehensive experiments with publicly available and widely used real benchmark datasets for performance evaluations.

1.3 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we present our online nonlinear NP classifier (NP-NN) and its comparison with the state of the art algorithms on real and synthetic datasets. Chapter 3 introduces Tree OLNP, as a mitigation method for overfitting behaviour of NP-NN via context tree. In Chapter 4, we introduce an active learning method for Tree OLNP to speed up the performance convergence of the ensemble model while maintaining the user specified false alarm rate. In Chapter 5, we use Tree OLNP in order to solve anomaly detection problem in video data. Finally in Chapter 6, we draw our conclusion and present possible directions for future work.

2. A Neural Network Approach for Online Nonlinear Neyman-Pearson Classification

Designing a binary classifier with asymmetrical costs for the errors of type I (false positive) and type II (false negative) [45, 46, 35], or equivalently designing a Neyman-Pearson classifier [18], is required in various applications ranging from facial age estimation [47], multi-view learning [48] and software defect prediction [49] to video surveillance [50] and data imputation [51]. For example, in medical diagnostics, type II error (misdiagnosing as healthy) has perhaps more severe consequences, whereas type I error (misdiagnosing as unhealthy) may result in devastating psychological effects [52]. In this example, the error costs must be set probably asymmetrically for the cost sensitive learning [45, 46] of the desired classifier, however, it could be difficult to determine the right cost structure to be imposed on the errors. Another example with the same difficulty is anomaly detection for the security and surveillance applications. In such applications of detecting anomalies (e.g. accidents, crimes, frauds, violations), the type I error rate must certainly be controlled since giving a false alarm, i.e., false anomaly, too often is frustrating and costly as it draws unnecessary attention from security agents. It is also important to maintain the reliability of detections and not fail to draw attention in the case of a true anomaly. Hence, the user must set the costs of both error types to match the bearable false alarm rate, however, setting that correctly could be again difficult to guarantee to not give a false alarm, for instance, no more than once a day or week. Therefore, it is often more convenient and practical -but technically equivalent [18]- to describe the user needs by the maximum tolerable type I error, cf. [16] and the references therein, instead of having to determine the error costs to meet the tolerance. This leads to the Neyman-Pearson (NP) characterization of the desired classifier [18] and false positive rate controllability, where the goal is to maximize the detection power, i.e., minimize type II error, while upper-bounding the false positive rate, i.e., type I error, by a user-specified threshold. In this chapter, we target and solve the problem of designing a computationally highly efficient NP classifier while achieving powerful nonlinear data modeling with potential applications in, for instance, security, surveillance and diagnostics.

To this goal, as the first time in the literature, in this chapter, we introduce a novel online and nonlinear NP classifier based on a single hidden layer feedforward neural network (SLFN), which is sequentially learned with a Lagrangian non-convex NP objective (i.e. maximum detection power about a controllable user specified false positive rate). We use stochastic gradient descent (SGD) optimization for scalability to voluminous data and online processing with limited memory requirements. During the SGD iterations, we a) sequentially infer the value of the Lagrangian multiplier in a data driven manner to obtain the correspondence between the asymmetrical error costs and the desired type I error rate, and b) update all the SLFN parameters to maximize the detection power (minimize the resulting cost sensitive classification error) at the desired false positive rate. To achieve powerful nonlinear modeling and improve scalability, we use the SLFN in a kernel inspired manner, cf. [53] for the kernel approach to nonlinearity. For this purpose, the hidden layer is initialized with a sinusoidal activation to approximately construct the high dimensional kernel space (of any symmetric and shift invariant kernel under Mercer's conditions, e.g., radial basis function) through the random Fourier features (RFFs) [53]. The output layer follows with identity activation.

This chapter continues with the Section 2.1, where we discuss state-of-the-art NP classification methods. We provide the problem description in Section 2.2, and then introduce our technique for online and nonlinear NP classification in Section 2.3. After the experimental evaluation is presented in Section 2.4, the results are analyzed in Section 2.5 and we conclude in Section 2.6.

2.1 Related Work

Neyman-Pearson classification has found a wide-spread use across various applications due to the direct control over the false positive rate that it offers, cf. [16] and the references therein. For example, an NP classifier is commonly employed for anomaly detection, where the false positive rate controllability is particularly important. In the one class formulation (due to the extreme rarity of anomalies) of anomaly detection [54, 55, 40, 56], the NP classification turns out (when the anomalies are assumed uniformly distributed) estimating the minimum volume set (MVS) that covers $1-\tau$ fraction of the nominal data (τ is the desired false positive rate). Then, an instance is anomalous if it is not in the MVS. A structural risk minimization approach is presented in [54] for learning the MVS based on a class of

sets generated by a dyadic tree partitioning. Geometric entropy minimization [55] and empirical scoring [40] can also be used to estimate the MVS, both of which are based on nearest neighbor graphs. The scoring of [40] is later extended to the local anomaly detection in [56] and a new one class support vector machines (SVM) in [57]. Although the algorithms in these examples with batch processing, i.e., not online, have decent theoretical performance guarantees, they are not scalable to large scale data due to their prohibitive computational as well as space complexity and hence they cannot be used in our scenario of fast streaming applications. Online extensions to the original batch one class SVM [58], which can be shown to provide an estimator of the MVS [59], have been proposed for distributed processing [60] and wireless sensor networks [61]. However, neither these online extensions nor the original one class SVM address the false positive rate controllability as they require additional manual parameter tuning for that. In contrast, our proposed online NP classifier directly controls (without parameter tuning) the false positive rate and maximizes the detection power with nonlinear modeling capabilities. Furthermore, NP formulation in the one class setting requires the knowledge of the target density (e.g., anomaly), which is often unknown and thus typically assumed to be uniform; but then the problem can be turned into a supervised binary NP classification by simply sampling from the assumed target density. On the other hand, when there is also data from the target class, the one class formulation in aforementioned studies does not directly address how to incorporate the target data. Hence, our two class supervised formulation of binary NP classification also covers the solution of the one class classification, and our proposed algorithm is consequently more general and applicable in both cases of target data availability.

Among the two class binary NP classification studies (cf. [16] for a survey), plug-in approaches (such as [62] and [63]) based on density estimation as an application of the NP lemma [25] are difficult to be applied in high dimension due to overfitting [55]. Particularly, [62] exploits the expectation-maximization algorithm for density estimation using a neural network with -however- batch processing and manual tuning for finding the threshold to satisfy the NP type I error constraint. In [64], a neural network is trained with symmetric error costs for modeling the likelihood ratio, which is thresholded to match the desired false positive rate but determining the threshold requires additional work. Unlike our presented work, approaches in [62, 63, 64] are also not online and do not allow real time false positive rate controllability. Recall that NP classification is equivalent to cost sensitive learning [18] when the desired false positive rate can be accurately translated to error costs, but achieving an accurate translation, i.e., correspondence, is typically nontrivial requiring special attention [18, 65]. This correspondence problem is addressed i) in [18] as

parameter tuning with improved error estimations, and ii) in [65] as an optimization with the assumption of class priors and unlabeled data. Besides the exploitation of SVM [18], other classifiers such as logistic regression [66] have also been considered in [20] and incorporated into a unifying NP framework as an umbrella algorithm. We emphasize that these approaches, the SVM based tuning approach [18] and the risk minimization of [25] as well as the umbrella algorithm [20] in addition to the optimization of [65], do not satisfy our computational online processing requirements, as they are batch techniques and not scalable to large scale data.

In most of the contemporary fast streaming data applications, such as computer vision based surveillance [67] and time series analysis [68], computationally efficient processing along with only limited space needs is a crucial design requirement. This is necessary for scalability in such applications which constantly generate voluminous data at unprecedented rates. However, the literature about the Neyman-Pearson classification (cf. [16] for the current state) appears to be fairly limited from this large scale efficient processing point of view. Out of very few examples, a linear-time algorithm for learning a scoring function and thresholding is presented in [21], which is still not an online algorithm (i.e. it is not designed to process data indefinitely on the fly) since batch processing is assumed with large space complexity and processing latency. Moreover, scoring of [21] is similar to the one of [40] but -unlike [40]- trades off NP optimality for linear-time processing. Also, the technique of [21] is restricted to linearly separable data only, and it requires to adjust thresholding for false positive rate controllability which can be seen impractical. The NP technique of [68] is truly online (and one class) but it is strongly restricted to Markov sources, thus fails in the case of general non-Markov data (whereas our proposed algorithm has no such restriction). Another online NP classifier is presented in [22] without strict assumptions unlike [68], but for only linearly separable data while leaving the online generalization to nonlinear setting as a future research direction.

To our best knowledge, online NP classification has not been studied yet in the nonlinear setting. Thus, as the first time in the literature, we solve the online and nonlinear NP classification problem based on a kernel inspired SLFN within the non-convex Lagrangian optimization framework of [22, 41], and use SGD updates for scalability. Our NP classifier exploits Fourier features [53] and sinusoidal activations in the hidden layer of the SLFN (hence the name kernel inspired) to achieve a powerful nonlinear modeling with high computational efficiency and online real time processing capability.

Random Fourier features (RFFs) and also kernels in general have been successfully used for classification and regression of large scale data (please refer to [53, 69, 70]

and [71] for examples). Our presented work also exploits RFFs (during SLFN initialization) for large scale learning but, in contrast, for the completely different goal of solving the problem of online nonlinear Neyman-Pearson (NP) classification with neural networks in a non-convex Lagrangian optimization framework. Furthermore, the presented work learns the useful Fourier features with SGD updates beyond the initial randomness. On the other hand, kernels and RFFs have been previously studied in conjunction with neural networks. For example, computational relations from certain kernels to large networks are drawn in [72], and a kernel approximating convolutional neural network is proposed in [73] for visual recognition. In particular, RFFs have been used to learn deep Gaussian processes [74], and for hybridization in deep models to connect linear layers nonlinearly [75]. A radial basis function (rbf) network is proposed in [24] with batch processing, i.e., not online, which briefly discusses a heuristic by varying rbf parameters to manually control the false positive rate. Note that our SLFN is not an rbf network since we explicitly construct (during initialization) the kernel space in the hidden layer without a further need for kernel evaluations. We stress that the hidden layer of our SLFN for NP classification is same as the RFF layer of [76] for kernel learning (a simultaneous development of the same layer). The RFF layer in [76] is proposed as a building block to deep architectures for the goal of kernel learning. However, our goal of designing an online nonlinear NP classifier is completely different. Hence, our formulation, network objective and the resulting training process as well as our algorithm and experimental demonstration in this paper are fundamentally different compared to [76]. Moreover, online processing is not a focus in these studies except that [73] and [74] address scalability to voluminous data; and none of those (including [76] for kernel learning, and [73] and [74] for scalability) consider our goal of NP classification. Finally, we note that the presented study comprehensively extends our previous conference paper [77] that only had certain initial findings of the preliminary version of our algorithm NP-NN presented here. In this paper, compared to our conference paper [77], we additionally 1) introduced Fourier feature learning (such features have been randomly drawn and kept untrained in [77]) in the nonconvex optimization framework of neural networks, 2) performed significantly more extensive experiments with a larger number datasets based on additional performance metrics (such as the NP-score), and 3) analyzed from different perspectives such as statistical significance and complexity.

2.2 Problem Description

We provide the problem description in this section. Regarding the notation, all vectors are column vectors and they are denoted by boldface lower case letters. For a vector $\boldsymbol{w}$, its transpose is represented by $\boldsymbol{w}'$ and the time index is given as subscript, i.e., $\boldsymbol{w}_t$. Also, a) $1_{\{.\}}$ is the indicator function returning 1 if its argument condition holds, and returning 0, otherwise; and b) $\text{sgn}(\cdot)$ is the sign function returning 1 if its argument is positive, and returning -1, otherwise.

Neyman-Pearson (NP) classification [16] seeks a classifier δ for a d dimensional observation $\mathbb{R}^d \ni \boldsymbol{x}$ to choose one of the two classes $H_y : \boldsymbol{x} \sim p_y(\boldsymbol{x})$ as $\delta(\boldsymbol{x}) = \hat{y} \in \{-1, +1\}$, where $y \in \{-1, +1\}$ (non-target: -1, target: 1) is the true class label and $p_y(\boldsymbol{x})$ are the corresponding conditional probability density functions. The goal is to minimize the type II error (non-detection) rate P_{nd}

$$\begin{aligned} \text{P}_{\text{nd}}(\delta) &= \int_{\forall \boldsymbol{x} \in \mathbb{R}^d} 1_{\{\hat{y}=-1\}} p_1(\boldsymbol{x}) d\boldsymbol{x} \\ &= E_1[1_{\{\hat{y}=-1\}}] \end{aligned} \tag{2.1}$$

(thus, the detection power $\text{P}_{\text{td}} = 1 - \text{P}_{\text{nd}}$ is maximized) while upper bounding the type I error P_{fa} (false positive) rate by a user specified threshold τ as

$$\begin{aligned} \text{P}_{\text{fa}}(\delta) &= \int_{\forall \boldsymbol{x} \in \mathbb{R}^d} 1_{\{\hat{y}=1\}} p_{-1}(\boldsymbol{x}) d\boldsymbol{x} \\ &= E_{-1}[1_{\{\hat{y}=1\}}] \leq \tau \end{aligned} \tag{2.2}$$

with E_y being the corresponding expectations. Namely, δ^* is an NP classifier, if it satisfies

$$\delta^* = \arg\min_{\delta} \text{P}_{\text{nd}}(\delta) \text{ subject to } \text{P}_{\text{fa}}(\delta) \leq \tau.$$

The Neyman-Pearson (NP) lemma [25, 17] states that the likelihood ratio test provides an optimal solution to the constrained optimization above once the false alarm rate of the test is equated to the user specified threshold τ. Moreover, such a likelihood ratio test always exists, and it is unique up to a subset in the observations space that has a zero probability mass under both hypotheses. We refer to [17] for

a rigorous proof. Thus, the likelihood ratio $\frac{p_1(x)}{p_{-1}(x)}$ provides the NP test, i.e.,

$$\begin{aligned} &\delta^*(\boldsymbol{x}) = -1, \text{ if } u(\boldsymbol{x}) = \frac{p_1(\boldsymbol{x})}{p_{-1}(\boldsymbol{x})} - v(\tau) \leq 0, \text{ and} \\ &\delta^*(\boldsymbol{x}) = 1, \text{ otherwise,} \end{aligned} \tag{2.3}$$

where the offset $v(\tau)$ is chosen to satisfy the false positive rate constraint. Hence, finding the discriminant function u is sufficient for NP testing.

The discriminant function u can be simplified in many cases, and it might be linear or nonlinear as a function of $\boldsymbol{x}$ after full simplification. We provide two corresponding examples in the following. For instance, if the conditional densities $p_y(\boldsymbol{x})$ are both Gaussian with same covariances, then the discriminant is linear. On the other hand, in the example of one class classification [58] with applications to anomaly detection, there is typically no data from the target (anomaly) hypothesis because of the extreme rarity of anomalies, and there is also not much prior information due to the unpredictable nature of anomalies. Hence, the usual approach is to assume that the target density is uniform (with a finite support) [40], i.e., $p_1(\boldsymbol{x}) = c$. Then, the critical region $\text{MVS} = \{\boldsymbol{x} \in \mathbb{R}^d : 1/p_{-1}(\boldsymbol{x}) \leq v(\tau)\}$ for the NP test to decide non-target, i.e., $\delta^*(\boldsymbol{x}) = -1$, is known as the minimum volume set (MVS) [54] covering $1-\tau$ fraction of the non-target instances, i.e., $v(\tau)$ is set with simplification such that $\int_{\boldsymbol{x} \notin \text{MVS} \subset \mathbb{R}^d} p_{-1}(\boldsymbol{x}) d\boldsymbol{x} = \tau$. Consequently, MVS has the minimum volume with respect to the uniform target density and hence maximizes the detection power. Here, the MVS discriminant $u(\boldsymbol{x}) = 1/p_{-1}(\boldsymbol{x}) - v(\tau)$ (after simplification) is generally nonlinear, for instance, even when $p_{-1}(\boldsymbol{x})$ is Gaussian with zero mean unit-diagonal covariance. Therefore, we emphasize that the discriminant u of the NP test[1] might be arbitrarily nonlinear in general. Furthermore, since the discriminant definition requires the knowledge of the conditional densities p_y which are unavailable in most realistic scenarios, the discriminant u is unknown. For this reason, NP classification refers to the data driven statistical learning of an approximation $f^* \in \mathcal{H}$ of the unknown discriminant u based on given two classes of data $\{(\boldsymbol{x}_t, y_t)\}$, where $\mathcal{H}$ is an appropriate set of functions which is sufficiently powerful to model the complexity of u.

As a result, the data driven statistical learning of the NP classifier f^* is obtained as

[1]Note that knowing the continues valued discriminant u is equivalent to knowing the discrete valued test δ^* due to one-to-one correspondence, i.e., $\delta^*(\boldsymbol{x}) = \text{sgn}(u(\boldsymbol{x}))$. Hence, in the rest of the paper, we refer to the discriminant as the NP classifier as well.

the output of the following NP optimization:

$$u \simeq f^* = \arg\min_{f \in \mathcal{H}} \hat{\mathrm{P}}_{\mathrm{nd}}(f) \text{ subject to } \hat{\mathrm{P}}_{\mathrm{fa}}(f) \leq \tau, \tag{2.4}$$

where

$$\hat{\mathrm{P}}_{\mathrm{nd}}(f) = \frac{\sum_{\forall t: y_t = 1} 1_{\{f(\boldsymbol{x}_t) \leq 0\}}}{\sum_{\forall t: y_t = 1} 1} \text{ and}$$
$$\hat{\mathrm{P}}_{\mathrm{fa}}(f) = \frac{\sum_{\forall t: y_t = -1} 1_{\{f(\boldsymbol{x}_t) > 0\}}}{\sum_{\forall t: y_t = -1} 1}$$

empirically estimates the type I (expectation in (2.1)) and type II (expectation in (2.2)) errors, respectively. For example, [22] studies this optimization in (2.4) for the set $\mathcal{H}$ of linear discriminants, in which case -however- the resulting linear NP classifier is largely suboptimal in most realistic scenarios; for example, the MVS estimation for anomaly detection requires to learn nonlinear class separation boundaries with a nonlinear discriminant.

Our goal in this chapter is to develop, as the first time in the literature to our best knowledge, an online nonlinear NP classifier for any given user-specified desired false positive rate τ with real time processing capability. In particular, we use a kernel inspired single hidden layer feed forward neural network (SLFN), cf. Fig. 2.1, to model the set $\mathcal{H}$ of nonlinear candidate discriminant functions in (2.4) as

$$\mathcal{H} = \{f : f(\boldsymbol{x}) = h_o(h_h(\boldsymbol{\alpha}\boldsymbol{x})'\boldsymbol{w} + b), \forall \boldsymbol{\alpha}, \forall \boldsymbol{w}, \forall b\}, \tag{2.5}$$

where $\boldsymbol{\alpha}$ and $(\boldsymbol{w}, b)$ are the hidden and output layer parameters, and h_h and h_o are the nonlinear hidden and identity output layer activations. We sequentially learn the SLFN parameters based on the NP objective (that is maximizing the detection power about a user-specified false positive rate as given in (2.4)) with stochastic gradient descent (SGD) to obtain the nonlinear classification boundary, i.e., to estimate the unknown discriminant u, in an online manner while maintaining scalability to voluminous data.

The data processing in our proposed algorithm is computationally highly efficient and truly online with $O(N)$ computational and $O(1)$ space complexity (N is the total number of processed instances). Namely, we sequentially observe the data $\boldsymbol{x}_t \in \mathbb{R}^d$ indefinitely without knowing a horizon, and decide about its label $\hat{y}_t \in \{1, -1\}$ as $\hat{y}_t = 1$ if the SLFN $f_t \in \mathcal{H}$ at time t provides $f_t(\boldsymbol{x}_t) > 0$, and as $\hat{y}_t = -1$, otherwise. Then, we update our model f_t, i.e., update the SLFN at time t, to obtain $f_{t+1} \in \mathcal{H}$ based on the error $y_t - \hat{y}_t$ via SGD and discard the observed data, i.e., $\boldsymbol{x}_t$ and y_t, without storing. Hence, each instance is processed only once. In this processing

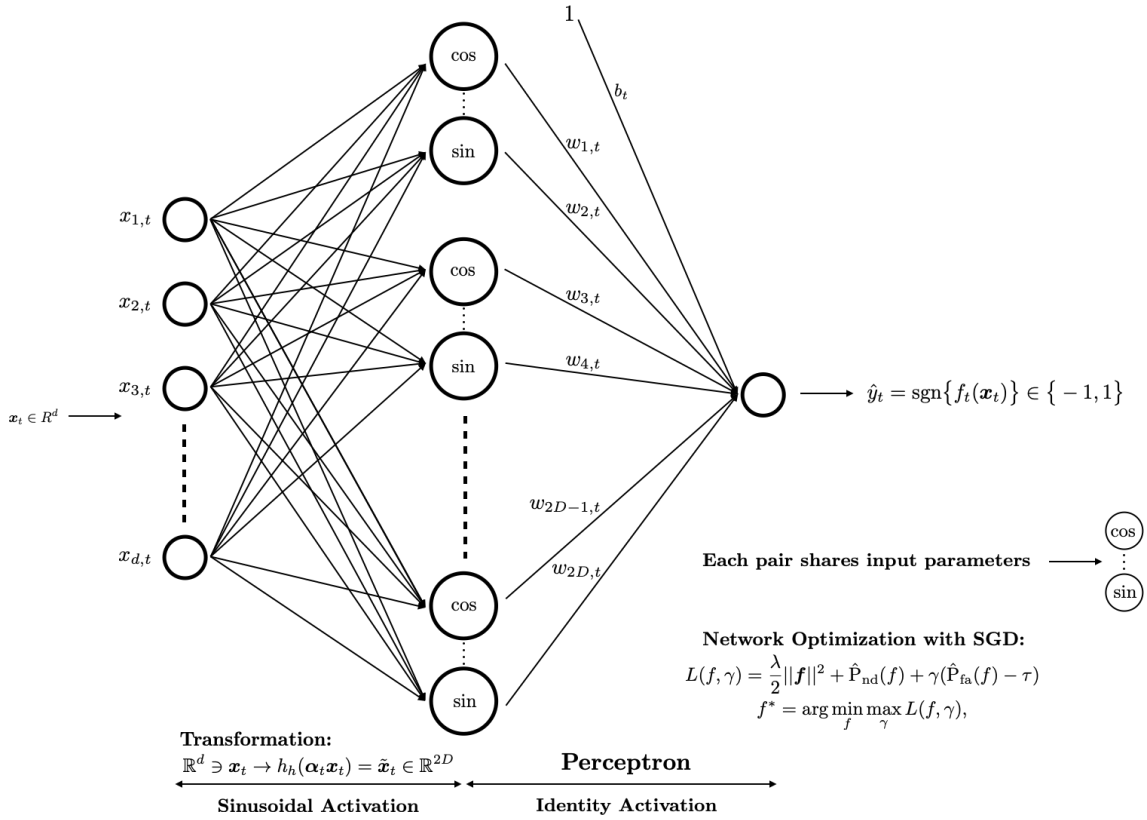


Figure 2.1 The single hidden layer feed forward neural network (SLFN) that we use for online nonlinear Neyman-Pearson (NP) classification is illustrated. The hidden layer is initialized to approximately construct the high dimensional kernel space (e.g., radial basis function) via random Fourier features (RFFs) with sinusoidal activation. The output layer follows with identity activation. This network is compact and strongly nonlinear with expedited learning ability thanks to i) the exponential convergence of the inner products (w.r.t. the number of hidden nodes) in the space of RFFs to the true kernel with an excellent random network initialization, and ii) the learning of Fourier features (instead of relying on randomization) by the data driven network updates. We learn the network parameters sequentially via SGD based on a nonconvex Lagrangian NP objective. The result is an expedited powerful nonlinear NP modeling with high computational efficiency and scalability.

framework, $f_t \to f^* \in \mathcal{H}$ models the NP discriminant u in (2.3). As a result of this processing efficiency, our algorithm is appropriate for large scale data applications.

2.3 SLFN for Online Nonlinear NP Classification

In order to learn nonlinear Neyman-Pearson classification boundaries, we use a single hidden layer feed forward neural network (SLFN), illustrated in 2.1, that is

designed based on the kernel approach to nonlinear modeling (cf. [53] and the references therein for the mentioned kernel approach). Namely, the hidden layer is randomly initialized to explicitly transform the observation space (via $\phi_{\boldsymbol{\alpha_1}}$) into a high dimensional kernel space with sinusoidal hidden layer activations by using the random Fourier features [53]. We use a certain variant of the perceptron algorithm [78] as the output layer with identity activation followed by a sigmoid loss. Based on this SLFN, we sequentially (in a truly online manner) learn the network parameters, i.e., the classifier parameters $\boldsymbol{w}_t, b_t$ as well as the kernel mapping parameters $\boldsymbol{\alpha_t}$, through SGD in accordance with the NP optimization objective (2.4).

In the hidden layer of the SLFN, the randomized initial transformation $\phi_{\boldsymbol{\alpha_1}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ at time $t = 1$,

$$\mathbb{R}^d \ni \boldsymbol{x} \to \tilde{\boldsymbol{x}} = \phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}) \in \mathbb{R}^{2D}, \tag{2.6}$$

is constructed based on the fact (as provided in [53]) that any continuous, symmetric and shift invariant kernel can be approximated as $k(\boldsymbol{x}^i, \boldsymbol{x}^j) \triangleq k(\boldsymbol{x}^i - \boldsymbol{x}^j) \approx \phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}^i)' \phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}^j)$ with an appropriately randomized kernel feature mapping. Note that the kernel $k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is an implicit access to the targeted high dimensional kernel space as it encodes the targeted inner products. This kernel space is explicitly and approximately constructed by the sinusoidal hidden layer activations of the SLFN in which the new inner products across activations approximate originally targeted inner products. Hence, linear techniques applied to the sinusoidal hidden layer activations can learn nonlinear models. In our method, we use the radial basis function (rbf) kernel[2] $k(\boldsymbol{x}^i, \boldsymbol{x}^j) = \exp(-g||\boldsymbol{x}^i - \boldsymbol{x}^j||^2)$ with the bandwidth parameter g (that is inversely related to the actual bandwidth).

In order to obtain a randomized mapping that explicitly constructs the kernel space, one can apply here the Bochner's theorem by using the derivation in [53]. This theorem states that (quoting from [53]) "a continuous kernel $k(\boldsymbol{x}^i, \boldsymbol{x}^j) = k(\boldsymbol{x}^i - \boldsymbol{x}^j)$ on $\mathbb{R}^d$ is positive definite if and only if $k(\boldsymbol{x}^i - \boldsymbol{x}^j)$ is the Fourier transform of a

[2] We use the rbf kernel in this chapter as an example but it is not required. Thus, the presented technique can be straightforwardly extended to any symmetric and shift invariant kernel satisfying the Bochner's theorem, cf. [53].

non-negative measure". Then,

$$
\begin{aligned}
k(\boldsymbol{x}^i - \boldsymbol{x}^j) &= \int_{\mathbb{R}^d} p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}) \exp(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'(\boldsymbol{x}^i - \boldsymbol{x}^j)) d\bar{\boldsymbol{\alpha}}_{\mathbf{1}} \\
&= \int_{\mathbb{R}^d} p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}) \cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'(\boldsymbol{x}^i - \boldsymbol{x}^j)) d\bar{\boldsymbol{\alpha}}_{\mathbf{1}} \\
&= \int_{\mathbb{R}^d} p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}) (\cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'\boldsymbol{x}^i) \cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'\boldsymbol{x}^j) + \\
&\sin(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'\boldsymbol{x}^i) \sin(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'\boldsymbol{x}^j)) d\bar{\boldsymbol{\alpha}}_{\mathbf{1}} \\
&= E_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}[r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}(\boldsymbol{x}^i) r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}(\boldsymbol{x}^j)'], \qquad (2.7)
\end{aligned}
$$

where the Fourier feature is

$$
r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}(\boldsymbol{x}) = [\cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'\boldsymbol{x}), \sin(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}{}'\boldsymbol{x})] \qquad (2.8)
$$

and $\bar{\boldsymbol{\alpha}}_{\mathbf{1}}$ is sampled from the d dimensional multivariate Gaussian distribution $p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}) = N(\mathbf{0}, 2g\boldsymbol{I})$ (which is the Fourier transform of the kernel in hand) with $E_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}$ being the corresponding expectation. From the first equation above to the second, we use that $p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}})$ is real since the kernel is real and even. Hence, by replacing the expectation in (2.7) with the independent and identically distributed (i.i.d) sample mean of the ensemble $\{r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{\boldsymbol{q}}}(\boldsymbol{x}^i) r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{\boldsymbol{q}}}(\boldsymbol{x}^j)'\}_{q=1}^{D}$ of size D, we define our kernel mapping as

$$
\begin{aligned}
\tilde{\boldsymbol{x}} &= \phi_{\boldsymbol{\alpha}_{\mathbf{1}}}(\boldsymbol{x}) \\
&= \sqrt{\frac{1}{D}} [r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{\mathbf{1}}}(\boldsymbol{x}), r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{\mathbf{2}}}(\boldsymbol{x}), \cdots, r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{\boldsymbol{D}}}(\boldsymbol{x})]', \qquad (2.9)
\end{aligned}
$$

which can be directly implemented in the hidden layer of the SLFN, cf. Fig. 2.1, along with the sinusoidal activation due to the definition of $r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}$.

Note that $\boldsymbol{\alpha}_{\boldsymbol{t}}$ keeps all the hidden layer parameters at time t as a matrix of size $2D \times d$ consisting of $\bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{i}}$'s corresponding to the hidden units, i.e., $\boldsymbol{\alpha}_{\boldsymbol{t}} = [\bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{1}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{1}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{2}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{2}}, \cdots, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{D}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{D}}]'$. And the hidden layer activation is sinusoidal: $h_h(m) = \cos(m)$ and $h_h(m) = \sin(m)$ for the odd and even indexed hidden nodes, respectively, due to the definition in (2.8). At time $t = 1$, $\boldsymbol{\alpha}_{\mathbf{1}}$ is randomly initialized with an appropriate g of the rbf kernel so that the SLFN starts with approximately constructing the high dimensional kernel space $\bar{\mathcal{H}} = \{f : f(\boldsymbol{x}) = h_o(h_h(\boldsymbol{\alpha}_1 \boldsymbol{x})' \boldsymbol{w} + b), \forall \boldsymbol{w}, \forall b\}$ in its hidden layer, and in relation to (2.6), $\tilde{\boldsymbol{x}} = \phi_{\boldsymbol{\alpha}_{\mathbf{1}}}(\boldsymbol{x}) = h_h(\boldsymbol{\alpha}_1 \boldsymbol{x})$. Note that $\bar{\mathcal{H}}$ of the rbf kernel readily provides a powerful nonlinear modeling to the SLFN even if the hidden layer is kept untrained. Thanks to this excellent network initialization, we achieve an expedited process of learning from data. Moreover, in the course of our sequential processing, the SLFN continuously updates and improves

the hidden layer, i.e., kernel mapping, parameters as $\boldsymbol{\alpha_t}$. Therefore, we optimize a nonlinear NP classifier in actually the larger space $\mathcal{H} \supset \bar{\mathcal{H}}$ (as our optimization is not restricted to $\boldsymbol{\alpha_1}$ of the random initialization, cf. the definition of $\mathcal{H}$ in (2.5)) for greater nonlinear modeling capability compared to the rbf kernel.

The SLFN in Fig. 2.1 that we use for online and nonlinear NP classification is compact in principle since the required number of hidden nodes is relatively small. The reason is that the convergence of the sample mean of the i.i.d. ensemble $\{r_{\bar{\boldsymbol{\alpha}}_1^q}(\boldsymbol{x}^i) r_{\bar{\boldsymbol{\alpha}}_1^q}(\boldsymbol{x}^j)'\}_{q=1}^D$ of size D to the true mean $k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is exponentially fast with the order of $O(e^{-D})$ by Hoeffding's inequality [53]. On the other hand, since random Fourier features are independent of data, further compactification is possible by eliminating irrelevant, i.e., not useful, Fourier features in a data driven manner, cf. the examples of feature selection in [70] and Nyström method in [79] for this purpose. In contrast, and alternatively, we distill useful Fourier features in the hidden layer activations as a result of the sequential learning of the kernel mapping parameters, i.e., $\bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{i}}$, via SGD. Hence, nodes of the SLFN are dedicated to only useful Fourier features, and thus we achieve a further network compactification by reducing the necessary number of hidden nodes as well as reducing the parameter complexity. Then, one can expect to better fight overfitting with great nonlinear modeling power and NP classification performance. This compactification does also significantly reduce the computational as well as space complexity of our SLFN based classifier, which -together with the SGD optimization- yields scalability to voluminous data. Consequently, the proposed online NP classifier is computationally highly efficient and appropriate for real time processing in large scale data applications.

Remark 1: We obtain a sequence of kernel mapping parameters $\boldsymbol{\alpha}_t$ in the course of data processing. This means that at the end of processing N instances, one can potentially construct a new non-isotropic rbf kernel by estimating the multivariate density of the collection $\{\bar{\boldsymbol{\alpha}}_N^j\}_{j=1}^D$ (here, we assume that D is large and the density is multivariate Gaussian. If it is not Gaussian, then one can straightforwardly incorporate a Gaussianity measure into the overall network objective) and then finding out the corresponding non-isotropic rbf kernel by taking back the inverse Fourier transform of the estimated density. Therefore, our algorithm is also kernel-adaptive since it essentially learns a new kernel (and also improves the previous one) at each SGD learning step. This kernel adaptation ability can be improved. For instance, one can start with a random mapping as described and estimate the density of the mapping parameters after convergence, and then re-start with new samples from the converged density. Multiple iterations of this process may yield better kernel adaptation (but re-running would hinder online processing and define batch processing, hence it is out of scope of the present work), which we consider as

future work.

In the output layer of the SLFN, we use a certain variant of perceptron [78] with the identity activation, i.e., $h_o(m) = m$. Then, the classification model is defined linearly after the hidden layer kernel inspired transformation as $f(\boldsymbol{x}) = h_o(\langle \boldsymbol{w}, \tilde{\boldsymbol{x}} \rangle) + b = \langle \boldsymbol{w}, \tilde{\boldsymbol{x}} \rangle + b = h_h(\boldsymbol{\alpha x})' \boldsymbol{w} + b$, where $\boldsymbol{w} \in \mathbb{R}^{2D}$ is the normal vector to the linear separator and $b \in \mathbb{R}$ is the bias. Thus, the decision of the SLFN is $\hat{y} = \text{sgn}(f(\boldsymbol{x}))$.

Regarding the overall network objective for sequential learning of the network parameters $\boldsymbol{\alpha}_t, \boldsymbol{w}_t, b_t$ and solving the NP optimization in (2.4) to obtain our SLFN based online nonlinear NP classifier, we next formulate the NP objective similar to [22] as

$$\begin{aligned} f^* = &\arg\min_{f \in \mathcal{H}} \frac{\lambda}{2} ||f||^2 + \hat{\text{P}}_{\text{nd}}(f) \\ &\text{subject to } \hat{\text{P}}_{\text{fa}}(f) \leq \tau, \end{aligned} \tag{2.10}$$

where the first term $\lambda/2||f||^2$ is the regularizer for which we use the magnitude of the classifier parameters in the output layer, i.e., $\lambda/2||\boldsymbol{w}||^2$, and λ is the regularization weight. For differentiability, the non-detection $\hat{\text{P}}_{\text{nd}}$ and false positive $\hat{\text{P}}_{\text{fa}}$ error rates are estimated based on data until time t as

$$\begin{aligned} \hat{\text{P}}_{\text{nd}}(f) &= \frac{1}{n_{t_+}} \sum_{t' \in S_1^t} l(f(\boldsymbol{x}_{t'})) \text{ and} \\ \hat{\text{P}}_{\text{fa}}(f) &= \frac{1}{n_{t_-}} \sum_{S_{-1}^t} l(-f(\boldsymbol{x}_{t'})) \end{aligned} \tag{2.11}$$

with $S_c^t = \{t' : 1 \leq t' \leq t, y_{t'} = c\}$, $n_{t_+} = |S_1^t|$ (set cardinality) and $n_{t_-} = |S_{-1}^t|$. Note that another appropriate function can be used here to obtain a differentiable surrogate for the $0-1$ errors in (2.4) for estimating the error rates. However, our results in the rest of this chapter are based on the sigmoid loss $l(m) = 1/(1+\exp(m))$.

For sequential optimization of the NP objective in (2.10), we next define the following Lagrangian

$$L(f, \gamma) = \frac{\lambda}{2} ||\boldsymbol{f}||^2 + \hat{\text{P}}_{\text{nd}}(f) + \gamma(\hat{\text{P}}_{\text{fa}}(f) - \tau), \tag{2.12}$$

where τ is the user-specified desired false positive rate and $\gamma \in \mathbb{R}^+$ is the corresponding Lagrange multiplier.

Since the saddle points of (2.12) correspond to the local minimum of (2.10), cf. [22] and [41] for the details, we apply the Uzawa approach [41] to search for the

saddle points of (2.12) and learn our parameters in the online setting with SGD updates. To be more precise, we follow the optimization framework of [22] and solve the min max optimization $f^* = \arg\min_f \max_\gamma L(f,\gamma)$ via an iterative approach with gradient steps, where one iteration minimizes $L(f,\gamma)$ for a fixed γ and the other maximizes $L(f,\gamma)$ for a fixed f. Note that the fixed-γ minimization

$$\begin{aligned}\arg\min_{f\in\mathcal{H}} L(f,\gamma) &= \arg\min_{f\in\mathcal{H}} \frac{\lambda}{2}||\boldsymbol{f}||^2 + \hat{\mathrm{P}}_{\mathrm{nd}}(f) + \gamma(\hat{\mathrm{P}}_{\mathrm{fa}}(f) - \tau) \\ &= \arg\min_{f\in\mathcal{H}} \frac{\lambda}{2}||\boldsymbol{f}||^2 + \hat{\mathrm{P}}_{\mathrm{nd}}(f) + \gamma\hat{\mathrm{P}}_{\mathrm{fa}}(f)\end{aligned}$$

is a regularized weighted error minimization, where the ratio of the type I error rate cost to the one of type II error rate is γ. Hence, the unknown Lagrange multiplier γ defines (up to a scaling with the prior probabilities) the asymmetrical error costs that correspond to the false positive rate constraint in (2.4). On the other hand, the gradient ascent updates $\gamma \leftarrow \gamma + \beta\nabla_\gamma L(f,\gamma) = \gamma + \beta(\hat{\mathrm{P}}_{\mathrm{fa}}(f) - \tau)$ in the fixed-f maximization determines the unknown multiplier γ so that the type I error cost is decreased (increased) if the error estimate is below (above) the tolerable rate τ in favor of detection power (true negative detection). This provides an iterative learning of the correspondence between the asymmetrical error costs and the NP constraint.

To this end, inserting the definitions in (2.11) and (2.11) into (2.12) with the regularization $\lambda/2||f||^2 = \lambda/2||\boldsymbol{w}||^2$ yields the overall SLFN objective as follows

$$\begin{aligned}L(f,\gamma) =& \frac{\lambda}{2}||\boldsymbol{w}||^2 + \frac{1}{n_{t_+}} \sum_{1\le t'\le t: y_{t'}=1} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) \\ &+ \frac{\gamma}{n_{t_-}} \sum_{1\le t'\le t: y_{t'}=-1} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) - \gamma\tau \\ =& \frac{1}{t}\sum_{t'=1}^{t} \left(\frac{\lambda}{2}||\boldsymbol{w}||^2 + \mu_{t'} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) - \gamma\tau\right) \\ =& \frac{1}{t}\sum_{t'=1}^{t} s(f,\gamma,t'),\end{aligned} \tag{2.13}$$

where $s(f,\gamma,t') = \left(\lambda/2||\boldsymbol{w}||^2 + \mu_{t'} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) - \gamma\tau\right)$ and $\mu_{t'} = t/n_{t_+}$ if $y_{t'} = +1$, and $\gamma t/n_{t_-}$, otherwise.

In order to learn the SLFN parameters for obtaining the proposed online nonlinear NP classifier via the NP optimization explained above, we use stochastic gradient descent (SGD) to sequentially optimize the overall network objective defined in (2.13). These network parameters are 1) $\boldsymbol{\alpha}$, to project input $\boldsymbol{x}$ to the higher dimensional

kernel space, 2) $\boldsymbol{w}$ and b, which are the perceptron parameters of the output layer to classify the projected input $\tilde{\boldsymbol{x}}$, and 3) γ, to learn the correspondence between the error costs and the NP constraint.

Suppose at the beginning of time t, we have an existing model f_t learned with the past data as well as the error costs corresponding to γ_t; and a little later, we observe the instance $\boldsymbol{x}_t$. SGD based optimization takes steps to update f_t and γ_t to obtain f_{t+1} and γ_{t+1} with respect to the partial derivatives of the instantaneous objective $s(f_t, \gamma_t, t)$. Namely, $f_{t+1} = f_t - \eta_t \nabla_f s(f_t, \gamma_t, t)$ and $\gamma_{t+1} = \gamma_t + \beta_t \nabla_\gamma s(f_t, \gamma_t, t)$. Based on the partial derivatives of the instantaneous objective $s(f_t, \gamma_t, t)$ defined in (2.13), the SGD updates for the SLFN parameters can be computed $\forall i \in \{1, \cdots, D\}$ as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \Big(\lambda \boldsymbol{w}_t + \mu_t \nabla_{\boldsymbol{w}} l\Big(y_t f_t(\boldsymbol{x}_t)\Big)\Big)$, $b_{t+1} = b_t - \eta_t \Big(\mu_t \nabla_b l\Big(y_t f_t(\boldsymbol{x}_t)\Big)\Big)$, $\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}_{\boldsymbol{t+1}} = \bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}_{\boldsymbol{t}} - \eta_t \Big(\mu_t \nabla_{\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}} l\Big(y_t f_t(\boldsymbol{x}_t)\Big)\Big)$, and $\gamma_{t+1} = \gamma_t + \beta_t \Big((1_{\{y_t=-1\}} t/n_{t_-}) l\Big(y_t f_t(\boldsymbol{x}_t)\Big) - \tau \Big)$, where η_t is the learning rate and β_t is named as the Uzawa gain [41] controlling the learning rate of the Lagrange multiplier. Using the sigmoid $l(m) = 1/(1+\exp(m))$ yields the partial derivatives with $\tilde{\boldsymbol{x}}_t = h_h(\boldsymbol{\alpha}_t \boldsymbol{x}_t)$ as

$$\nabla_{\boldsymbol{w}} l\Big(y_t f_t(\boldsymbol{x}_t))\Big) = -\tilde{\boldsymbol{x}}_t l^2(y_t f_t(\boldsymbol{x}_t)) \exp(y_t f_t(\boldsymbol{x}_t)) y_t, \tag{2.14}$$

$$\nabla_b l\Big(y_t f_t(\boldsymbol{x}_t))\Big) = -l^2(y_t f_t(\boldsymbol{x}_t)) \exp(y_t f_t(\boldsymbol{x}_t)) y_t, \text{ and} \tag{2.15}$$

$$\begin{aligned} \nabla_{\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}} l\Big(y_t f_t(\boldsymbol{x}_t))\Big) &= -\boldsymbol{x}_t l^2(y_t f_t(\boldsymbol{x}_t)) \exp(y_t f_t(\boldsymbol{x}_t)) y_t \\ &\times \Big(-\boldsymbol{w}_t^{2i-1} \sin(\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}'}_{\boldsymbol{t}} \boldsymbol{x}_t) + \boldsymbol{w}_t^{2i} \cos(\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}'}_{\boldsymbol{t}} \boldsymbol{x}_t)\Big), \end{aligned} \tag{2.16}$$

which can be straightforwardly incorporated into the backpropagation.

In our experiments, we obtain an empirical false positive rate estimate $\hat{\mathrm{P}}_{\mathrm{fa}}$ based on a sliding window keeping the $0-1$ errors for a couple hundreds of the past negative data instances, and use the following γ update instead of the aforementioned stochastic one:

$$\gamma_{t+1} = \gamma_t \Big(1 + \beta_t \Big(\hat{\mathrm{P}}_{\mathrm{fa}} - \tau \Big)\Big), \text{ when } y_t = -1, \tag{2.17}$$

which has been observed to yield a more stable and robust performance. Note that this update is directly resulted from (2.12), and does certainly not disturb real-time online processing since a past window of positive decisions requires almost no additional space complexity (only $\frac{2}{\tau}$, where τ is the TFPR, bits in the case of, for instance, storing binary decisions for 200 (for $\tau = 0.01$) past negative instances).

Based on the derivations above, we sequentially update the SLFN at each time in a truly online manner with $O(N)$ (here, N: total number of processed data instances) computational and $O(1)$ space complexity in accordance with the NP objective.

Algorithm Proposed Online Nonlinear Neyman-Pearson Classifier (NP-NN)

Set the desired (or target) false positive rate (TFPR) τ, regularization λ, number $2D$ of hidden nodes and bandwidth g for the rbf kernel
Initialize the SLFN parameters $\boldsymbol{\alpha}_1$, $\boldsymbol{w}_1$, b_1, and learning rates η_1, β_1, γ_1
Set $n_{t_+} = n_{t_-} = 0$, and sliding window size $W_s = \frac{2}{\tau}$
for $t = 1, 2, \ldots$ **do**
    Receive $\boldsymbol{x}_t$ and calculate $\tilde{\boldsymbol{x}}_t = h_h(\boldsymbol{\alpha}_t \boldsymbol{x}_t)$ and $f_t(\boldsymbol{x}_t) = \boldsymbol{w}_t' \tilde{\boldsymbol{x}}_t + b_t$
    Calculate the current decision as $\hat{y}_t = \mathrm{sgn}(f_t(\boldsymbol{x}_t))$ and observe y_t
    Calculate $n_{t_+} = n_{t_+} + 1_{\{y_t=1\}}$ and $n_{t_-} = n_{t_-} + 1_{\{y_t=-1\}}$
    Calculate $\mu_t = t/n_{t_+} 1_{\{y_t=1\}} + \gamma t / n_{t_-} 1_{\{y_t=-1\}}$
    Update $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \Big(\lambda \boldsymbol{w}_t + \mu_t \nabla_{\boldsymbol{w}} l\big(y_t f_t(\boldsymbol{x}_t)\big) \Big)$, cf. (2.14)
    Update $b_{t+1} = b_t - \eta_t \Big(\mu_t \nabla_b l\big(y_t f_t(\boldsymbol{x}_t)\big) \Big)$, cf. (2.15)
    Update $\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}_{\boldsymbol{t+1}} = \bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}_{\boldsymbol{t}} - \eta_t \Big(\mu_t \nabla_{\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}} l\big(y_t f_t(\boldsymbol{x}_t)\big) \Big)$, cf. (2.16)
    Update $\gamma_{t+1} = \gamma_t \Big(1 + \beta_t \big(\hat{\mathrm{P}}_{\mathrm{fa}} - \tau \big) \Big)$, if $y_t = -1$, cf. (2.17) and the explanation about the estimate $\hat{\mathrm{P}}_{\mathrm{fa}}$ of the false positive rate
    Update $\eta_{t+1} = \eta_1 (1 + \lambda t)^{-1}$ and $\beta_{t+1} = \beta_1 (1 + \lambda t)^{-1}$
end for

Hence, we construct our method called "NP-NN" in Algorithm that can be used in real time for online nonlinear Neyman-Pearson classification. We refer to the Section 2.4 of our experimental study for all the details about the input parameters and initializations.

Remark 2: Recall that the goal in NP classification is to achieve the minimum miss rate (maximum detection power) while upper bounding the false positive rate (FPR) by a user-specified threshold τ. Therefore, both aspects (minimum miss rate and its FPR constraint) of this goal should be considered in evaluating the performance of NP classifiers. The NP-score of [15, 18] is defined as

$$\text{NP-score} = \kappa \max(\hat{P}_{\mathrm{fa}}(f) - \tau, 0) + \hat{P}_{\mathrm{nd}}(f), \tag{2.18}$$

where f is the NP model to be evaluated and κ controls the relative weights of the miss rate (with weight 1) and its FPR constraint (with weight κ if the desired rate is exceeded, and with weight 0 otherwise). Namely, κ controls the hardness of the NP FPR constraint, and **a smaller NP-score indicates a better NP classifier**. By enforcing a strict hard constraint on FPR with a very large $\kappa \simeq \infty$, one can immediately reject models (while evaluating various models) that violate FPR constraint with even a slight positive deviation from the desired FPR τ (a negative deviation does not violate). However, even though the original NP formulation requires a hard constraint, we consider that it is not appropriate to use a hard

constraint in practice, as also extensively explained in [15], based on the following two reasons: (1) An NP classifier is typically learned using a set of observations, and that set is itself a random sample from the underlying density of the data. Hence, the estimated FPR $\hat{P}_{\text{fa}}(f)$ of the model is also a random quantity, which is merely an estimator of the unknown true FPR $P_{\text{fa}}(f)$. Note that the true FPR $P_{\text{fa}}(f)$ is actually the one to be strictly constrained, but unavailable. Thus, it is unreliable to enforce a strict hard constraint (with a very large $\kappa \simeq \infty$) on the random estimator $\hat{P}_{\text{fa}}(f)$, and a relatively soft constraint has surely more practical value by allowing a small positive deviation from the desired FPR τ. (2) Also, one might be willing to exchange true negatives in favor of detections with a small positive deviation from the desired FPR τ, when the gain is larger than the loss as the NP-score improves. Consequently, for parameter selections with cross validation in our algorithm design as well as for performance evaluations in our experiments, we opt for a relatively soft constraint and use $\kappa = 1/\tau$ in accordance with the recommendation by the authors [15]. This choice allows a relatively small positive deviation from the desired FPR, and normalizes the deviation by measuring it in a relative percentage manner. For example, the positive deviations 0.1 and 0.001 both degrade the score equally by 50% when the desired rates are 0.2 and 0.002, respectively. Various other NP studies in the literature do also practically allow small positive deviations from the desired FPR τ. For instance, we observe such a deviation in [22] with theirs and compared algorithms [18] in the case of spambase dataset, in [21] with theirs in the case of heart and breast cancer datasets, and finally in [65] with one of the compared algorithms [80] in all datasets.

A comprehensive experimental evaluation of our proposed technique (NP-NN) is next provided based on real as well as synthetic datasets in comparison to state-of-the-art competing methods.

2.4 Experiments

We present extensive comparisons of the proposed kernel inspired SLFN for online nonlinear Neyman-Pearson classification (NP-NN), described in Algorithm , with 3 different state-of-the-art NP classifiers. These compared techniques are online linear NP (OLNP) [22], as well as logistic regression (NPROC-LOG) [66] and support vector machines with rbf kernel (NPROC-SVM) [81] in the NP framework of the umbrella algorithm described in [20]. Among these, OLNP (linear NP classification)

is an online technique with $O(N)$ computational complexity, whereas NPROC-LOG (linear NP classification) and NPROC-SVM (nonlinear NP classification) are batch techniques with at least $O(N^2)$ computational complexity, where N is the number of processed instances. In contrast, we emphasize that to our best knowledge, the proposed NP classifier NP-NN is both nonlinear and online as the first time in literature, with $O(N)$ computational and negligible space complexity resulting real time nonlinear NP modeling and false positive rate controllability. Consequently, the proposed NP-NN is appropriate for challenging fast streaming data applications.

Since our proposed algorithm NP-NN is the first online and nonlinear NP classifier, there is technically no fully comparable algorithm in the literature. Nevertheless, we set our experiments in the fairest manner by considering comparisons among all possibilities: batch nonlinear (NPROC-SVM), batch linear (NPROC-LOG), online nonlinear (our proposed NP-NN) and online linear (OLNP). Although we compare with NPROC-LOG, it is not particularly strong in terms of efficiency with small computational and space complexity, and it is also not particularly strong in terms of nonlinear modeling capability. For this reason, we mainly concentrate on comparisons to NPROC-SVM in terms of the classification performance, and on comparisons to OLNP in terms of the complexity. NPROC-SVM is extremely powerful and well-known with regards to nonlinear modeling, whereas OLNP is extremely powerful with regards to efficiency both computationally and space-wise. Otherwise, there is no algorithm (except our proposed online and nonlinear algorithm NP-NN) in the literature which is powerful in terms both nonlinear modeling and efficiency. In order to further ensure fairness in our experiments, we pay special attention to the followings. 1) Through extensive cross validations, we optimize the parameters for each algorithm and for each dataset separately in their respective contexts. 2) We run the algorithms on the same exact sequence in each case of our datasets, which is particularly important while comparing online algorithms (our algorithm NP-NN and OLNP). 3) Note that a specific data sequence can favor one algorithm by luck. In order to remove this dependency on sequence order, we run the algorithms on 15 different random permutations (all algorithms are run on the same exact set of randomly permuted data sequences) and report the mean performance along with the standard deviations. 4) We do not rely on a specific performance measure. Instead, we report the performance in terms of the area under the ROC (receiver operating characteristics) curve (AUC) as well as the NP-scores for each target false alarm rate separately. Furthermore, we also report the achieved false alarm rate and the true positive rate. 5) The proposed algorithm NP-NN and the OLNP are online algorithms and thus they do not have separate training and test phases. However, NPROC-SVM is a batch algorithm with separate training and test

phases. Hence, to ensure fairness, we use separate training and test phases while comparing with NPROC-SVM (although it is certainly NOT needed for OLNP and the proposed NP-NN). Otherwise, while comparing the proposed online algorithm NP-NN with OLNP on large scale datasets, we test them (NP-NN and OLNP) in the online data processing framework (without separate training and test) that they (NP-NN and OLNP) are originally designed for. 6) We present detailed statistical significance (for fairly evaluating the performance differences) and complexity analyses (for fairly evaluating the complexity and running time differences) in the end as two separate sections. After providing this summary and important remarks about our experimental paradigm, next, we continue with the details and present our results.

We conduct experiments based on various real and synthetic datasets [82, 83] from several fields such as bioinformatics and computer vision, each of which is normalized by either unit-norm (each instance is divided by its magnitude) or z-score (each feature is brought down to zero mean unit variance) normalization before processing. For each dataset, smaller class is designated as the positive (target) class. The details of the datasets are provided in Table 2.1, where the starred ones and unstarred ones are normalized with unit norm and z-score, respectively. For performance evaluations, we generate 15 random permutations of each dataset, and each random permutation is split into two as training (%75) and test (%25) sequences. We strongly emphasize that the processing in the proposed algorithm NP-NN is truly online, meaning that, there are no separate training and test phases. However, since NPROC-LOG and NPROC-SVM are batch algorithms requiring a separate training, we opt to use training/test splits in this first set of experiments for a fair and statistically unbiased robust performance comparison. Such a split is in fact not needed in practice in the case of the proposed NP-NN that -by design- processes data on the fly. Additional experiments based on two larger scale datasets to demonstrate the ideal use-case (i.e. online processing without separate training/tests phases) of the proposed algorithm NP-NN are presented in Fig. 2.4.

The rbf kernel bandwidth parameter g (for the proposed NP-NN as well as NPROC-SVM), the error cost parameter C (for NPROC-SVM) and the number $2D$ of hidden nodes (for the SLFN in the proposed NP-NN) are all 3-fold cross-validated (based on NP-score) for each random permutation using the corresponding training sequence by a grid search with $g \in \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$, $C \in \{0.1, 1, 2, 4\}$ and $D \in \{2, 5, 10, 20, 40, 80, 100\} \times d$, where d is the data dimension. As the regularization has been observed to help little, we opt to use $\lambda \sim 0$ along with SGD learning updates $\eta_t = 0.01$ and $0.1 \geq \beta_t/\eta_t \geq 0.01$, randomly initialized $\boldsymbol{w}_1$ and b_1 (around 0) and $\gamma_1 = 1$ for both the proposed NP-NN and OLNP uniformly in all of our

Table 2.1 We present performance results of the proposed algorithm NP-NN and the competing algorithms OLNP and NPROC-SVM for each targeted false positive rate (TFPR $\in \{0.05, 0.1, 0.2, 0.3, 0.4\}$) on the datasets in the leftmost column. In each case, we run the algorithms 15 times over 15 different permutations of the dataset, where 75% (25%) is used for training (testing). Therefore, average of the test results are presented with the corresponding standard deviation. For each dataset, the first row is the achieved true positive rate (TPR), the second row is the achieved false positive rate (FPR), the third row is the NP-score, and the fourth row is the area under curve (AUC) of the receiver operating characteristics (ROC) curve of TPR vs TFPR. The best and the second best performing algorithms are signified with fonts in bold style. Overall, we observe that A) the proposed NP-NN and NPROC-SVM outperform (due to their nonlinear modeling) OLNP (based on its results in Fig. 2.2), B) the proposed NP-NN and NPROC-SVM perform comparably in terms of AUC, and NPROC-SVM performs better in terms of NP-score, where the advantage of NPROC-SVM in terms of NP-score seems to disappear as the data size and/or TFPR increase, and C) the proposed NP-NN with online and real-time processing capabilities at $O(N)$ computational and negligible $O(1)$ space complexity has huge advantages over the competing NPROC-SVM in the case of contemporary large scale fast streaming data applications. Namely, when one has a fast streaming dataset of size in the order of millions or more, then the only choice of high performance in real time is the proposed NP-NN, cf. Fig. 2.4.

Dataset	OLNP					NPROC-SVM					NP-NN				
(n_-, n_+, d)	**TFPR**(τ)→ 0.05	0.1	0.2	0.3	0.4	0.05	0.1	0.2	0.3	0.4	0.05	0.1	0.2	0.3	0.4
Banana	**TPR**→ 0.177±0.017	0.213±0.014	0.299±0.038	0.444±0.025	0.546±0.032	0.773±0.095	0.884±0.017	0.948±0.012	0.978±0.006	0.994±0.004	0.749±0.188	0.894±0.007	0.946±0.01	0.979±0.005	0.991±0.003
	FPR→ 0.052±0.012	0.097±0.015	0.197±0.032	0.309±0.032	0.407±0.023	0.038±0.008	0.078±0.012	0.176±0.017	0.276±0.016	0.383±0.02	0.053±0.006	0.098±0.012	0.2±0.017	0.298±0.016	0.392±0.015
(2924, 2376, 2)	**NP-score**→ 0.935±0.114	0.815±0.064	0.752±0.085	0.613±0.056	0.487±0.026	**0.231**±0.092	**0.116**±0.017	**0.053**±0.012	**0.022**±0.004	**0.011**±0.012	**0.325**±0.179	**0.148**±0.068	**0.089**±0.51	**0.039**±0.031	**0.016**±0.014
	AUC→0.128±0.012					**0.347**±0.006					**0.347**±0.009				
Breast*	0.908±0.039	0.949±0.025	0.987±0.018	0.985±0.017	0.996±0.008	0.936±0.032	0.974±0.024	0.987±0.013	0.994±0.009	0.994±0.009	0.958±0.026	0.972±0.024	0.985±0.019	0.991±0.01	0.998±0.006
	0.06±0.023	0.108±0.025	0.196±0.043	0.294±0.057	0.401±0.054	0.021±0.014	0.057±0.028	0.13±0.036	0.226±0.054	0.36±0.041	0.053±0.027	0.099±0.027	0.19±0.034	0.273±0.026	0.397±0.04
(357, 212, 30)	0.391±0.324	0.193±0.175	0.085±0.15	0.073±0.119	0.056±0.09	**0.077**±0.054	**0.039**±0.043	**0.013**±0.013	**0.011**±0.016	**0.016**±0.03	**0.261**±0.376	**0.113**±0.205	**0.051**±0.106	**0.015**±0.016	**0.037**±0.057
	0.363±0.005					**0.368**±0.005					**0.368**±0.005				
Pageblocks	0.833±0.015	0.905±0.015	0.955±0.013	0.979±0.011	0.981±0.015	0.925±0.02	0.954±0.025	0.981±0.013	0.992±0.011	0.996±0.005	0.926±0.015	0.967±0.014	0.982±0.012	0.993±0.007	0.994±0.005
	0.06±0.008	0.095±0.012	0.195±0.11	0.295±0.016	0.397±0.012	0.042±0.009	0.096±0.011	0.188±0.014	0.288±0.016	0.387±0.009	0.052±0.008	0.088±0.017	0.176±0.019	0.283±0.018	0.402±0.03
(4913, 560, 10)	0.375±0.16	0.122±0.058	0.055±0.023	0.034±0.026	**0.026**±0.022	**0.09**±0.028	**0.075**±0.059	**0.027**±0.037	**0.013**±0.018	**0.004**±0.005	**0.159**±0.081	**0.061**±0.074	**0.021**±0.015	**0.013**±0.015	0.04±0.05
	0.352±0.004					**0.366**±0.003					**0.366**±0.003				
Bupaliver*	0.311±0.063	0.428±0.1	0.497±0.079	0.558±0.061	0.658±0.051	0±0	0.242±0.082	0.444±0.083	0.603±0.119	0.706±0.096	0.369±0.091	0.456±0.073	0.544±0.087	0.661±0.029	0.739±0.094
	0.064±0.035	0.13±0.063	0.216±0.091	0.3±0.066	0.428±0.074	0±0	0.042±0.048	0.114±0.069	0.2±0.059	0.3±0.086	0.094±0.048	0.12±0.059	0.21±0.061	0.366±0.035	0.442±0.076
(200, 145, 6)	**1.129**±0.481	1.012±0.413	0.723±0.332	**0.535**±0.132	0.462±0.108	**1**±0	**0.778**±0.073	**0.576**±0.092	**0.397**±0.119	**0.299**±0.089	1.531±0.909	**0.904**±0.365	**0.606**±0.215	0.559±0.125	**0.396**±0.134
	0.187±0.022					0.156±0.026					**0.21**±0.024				
Cod-rna	0.915±0.002	0.977±0.002	0.995±0.001	0.998±0	0.999±0	0.959±0.003	0.983±0.02	0.997±0.003	0.999±0	1±0	0.938±0.002	0.979±0.002	0.995±0.001	0.998±0.001	0.999±0
	0.05±0.002	0.097±0.004	0.196±0.004	0.294±0.005	0.395±0.007	0.047±0.003	0.08±0.024	0.145±0.053	0.222±0.051	0.294±0.005	0.054±0.002	0.09±0.003	0.187±0.004	0.292±0.008	0.393±0.006
(36690, 19845, 8)	**0.102**±0.026	0.029±0.008	0.006±0.004	**0.003**±0.004	**0.002**±0.003	**0.046**±0.015	**0.017**±0.02	**0.005**±0.005	**0.001**±0	**0**±0	0.14±0.042	**0.021**±0.002	**0.005**±0.001	0.004±0.005	**0.002**±0.004
	0.368±0					**0.371**±0.001					**0.37**±0				
Pen Digits*	0.975±0.015	0.987±0.01	0.996±0.005	0.998±0.002	0.999±0.002	1±0	1±0	1±0	1±0	1±0	0.993±0.005	0.996±0.003	0.999±0.002	0.997±0.003	0.998±0.002
	0.049±0.009	0.095±0.014	0.202±0.014	0.294±0.014	0.401±0.018	0.04±0.007	0.088±0.006	0.186±0.009	0.279±0.018	0.382±0.011	0.032±0.008	0.088±0.012	0.188±0.015	0.292±0.013	0.393±0.011
(6714, 780, 16)	0.084±0.1	0.043±0.05	0.038±0.036	**0.009**±0.016	0.02±0.023	**0.001**±0.004	**0**±0	**0**±0	**0.001**±0.002	**0**±0	**0.007**±0.005	**0.009**±0.014	**0.01**±0.026	**0.009**±0.011	**0.005**±0.009
	0.372±0.002					**0.375**±0					**0.374**±0.001				
Spiral	0.34±0.066	0.336±0.097	0.328±0.088	0.332±0.063	0.516±0.289	0±0	1±0	1±0	1±0	1±0	1±0	0.98±0.063	0.996±0.013	0.996±0.013	1±0
	0.056±0.034	0.125±0.051	0.229±0.075	0.31±0.07	0.463±0.232	0±0	0.052±0.048	0.085±0.051	0.213±0.069	0.335±0.058	0.019±0.024	0.11±0.043	0.229±0.058	0.348±0.071	0.45±0.09
(211, 101, 2)	**0.983**±0.475	1.002±0.437	0.905±0.266	0.786±0.132	0.793±0.185	1±0	**0.069**±0.172	**0**±0	**0.015**±0.049	**0.015**±0.049	**0.054**±0.17	**0.235**±0.309	**0.222**±0.188	**0.204**±0.174	**0.159**±0.191
	0.141±0.038					**0.325**±0					**0.373**±0.006				
Iris	0.355±0.1	0.455±0.214	0.727±0.086	0.664±0.332	0.955±0.064	0±0	0±0	1±0	1±0	1±0	0.964±0.047	0.982±0.038	1±0	1±0	1±0
	0.116±0.076	0.148±0.082	0.248±0.082	0.316±0.15	0.428±0.15	0±0	0±0	0.096±0.097	0.168±0.106	0.24±0.075	0.1±0.074	0.148±0.087	0.244±0.104	0.308±0.136	0.404±0.106
(100, 50, 4)	2.045±1.386	1.165±0.524	0.593±0.277	0.55±0.397	0.245±0.212	**1**±0	**1**±0	**0.06**±0.19	**0.02**±0.063	**0**±0	**1.136**±1.362	**0.618**±0.735	**0.34**±0.366	**0.2**±0.276	**0.11**±0.173
	0.234±0.048					**0.249**±0.003					**0.37**±0.008				
Svmguide1*	0.606±0.011	0.68±0.012	0.776±0.008	0.831±0.009	0.875±0.009	0.697±0.07	0.852±0.011	0.924±0.007	0.951±0.005	0.965±0.004	0.762±0.022	0.82±0.028	0.884±0.017	0.92±0.017	0.947±0.014
	0.051±0.007	0.102±0.01	0.204±0.015	0.299±0.013	0.4±0.015	0.039±0.009	0.082±0.01	0.176±0.021	0.277±0.019	0.371±0.019	0.053±0.006	0.104±0.012	0.204±0.014	0.301±0.02	0.395±0.024
(4000, 3089, 4)	0.446±0.068	0.369±0.064	0.267±0.045	0.186±0.022	0.141±0.023	**0.319**±0.06	**0.148**±0.011	**0.077**±0.008	**0.052**±0.013	**0.035**±0.005	**0.312**±0.084	**0.245**±0.08	**0.154**±0.049	**0.107**±0.043	**0.072**±0.021
	0.285±0.003					**0.334**±0.004					**0.326**±0.006				
Fourclass1	0.463±0.063	0.516±0.044	0.609±0.054	0.829±0.036	0.912±0.023	1±0	1±0	1±0	1±0	1±0	1±0	1±0	0.986±0.046	1±0	1±0
	0.056±0.023	0.13±0.028	0.233±0.027	0.312±0.024	0.423±0.045	0.033±0.03	0.066±0.027	0.144±0.035	0.254±0.041	0.366±0.034	0.021±0.017	0.065±0.012	0.163±0.029	0.285±0.049	0.353±0.029
(555, 307, 2)	0.75±0.394	0.807±0.238	0.557±0.119	0.225±0.065	0.169±0.087	**0.133**±0.329	**0.009**±0.027	**0.005**±0.016	**0.013**±0.029	**0.003**±0.01	**0.03**±0.096	**0**±0	**0.017**±0.045	**0.043**±0.09	**0.001**±0.005
	0.252±0.015					**0.375**±0					**0.372**±0.006				

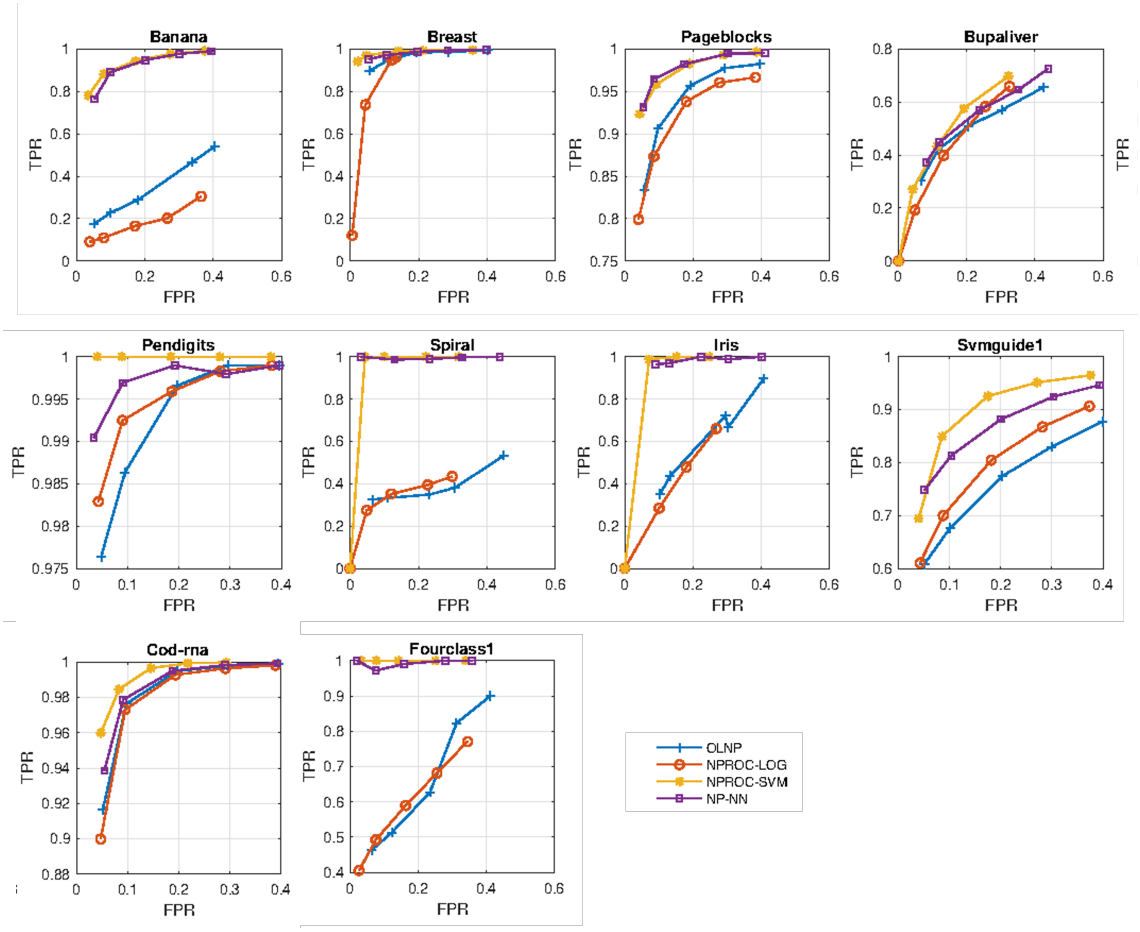


Figure 2.2 Visual presentations of the results in Table 2.1 are provided via the receiver operating characteristics (ROC) curves for all compared algorithms of OLNP, NPROC-LOG, NPROC-SVM and the proposed NP-NN, based on the achieved true positive rates and achieved false positive rates, i.e., TPR vs FPR, corresponding to the targeted false positive rates TFPR $\in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. Note that the presented ROC curves (TPR vs FPR) are mean curves over 15 trials of random data permutations for which the standard deviations can be followed from Table 2.1. Overall, in terms of the area under ROC (AUC), we observe that the proposed NP-NN and NPROC-SVM (due to their nonlinear modeling) outperform the other two. On the other hand, the proposed NP-NN performs similarly with NPROC-SVM while providing significant computational advantages. For the quantification of the false positive rate tractability alone, AUC alone and both as a combined measure, we refer to the results in Fig. 2.3, the AUC scores in Table 2.1 and the NP-scores in Table 2.1, respectively.

experiments. We directly use the code provided by the authors [20] for NPROC-LOG and NPROC-SVM and also optimize it by the aforementioned cross validation in terms of parameter selection. We observe that for the datasets of relatively short length, algorithms using SGD optimization, i.e., OLNP and NP-NN, improve with multiple passes over the training sequence. Hence, the length of the training sequence of each random permutation is increased by concatenation with additional randomizations for only OLNP and NP-NN (not for NPROC-LOG and NPROC-

SVM) during training of both the cross-validation and actual training, resulting in an epoch-by-epoch training procedure. This concatenation is only for training purposes, and hence it is not used in testing and validation, i.e., the actual data size is used in all types of testing to avoid statistical bias and multiple counting. Our proposed algorithm NP-NN does certainly not need such a concatenation approach for data augmentation in the targeted fast streaming data applications (cf. Fig. 2.4), where data is already abundant and scarcity is not an issue.

We run all the algorithms on the test sequence of each of the 15 random permutations (after training on the corresponding training sequences), and record in each case the achieved false positive rate, i.e., FPR, and true detection rate, i.e., TPR, for the target false positive rates (TFPR) $\tau \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. For performance evaluation, we compare the mean area under curve (AUC) of the resulting 15 receiver operating characteristics (ROC) curves of TFPR vs TPR, as well as the mean of the resulting 15 NP-scores [15], cf. (2.18) with $\kappa = 1/\tau$. Note that the mean AUC (higher is better) accounts only for the resulting detection power without regard to false positive rate tractability, whereas the mean NP-score (lower is better) provides an overall combined measure. We evaluate with the both (Table 2.1) in addition to visual presentation of the mean ROC curves (Fig. 2.2) of FPR and TPR. Table 2.1 additionally reports the mean TPRs and mean FPRs. We also provide the decision boundaries and the mean convergence of the achieved false positive rate during training for the visually presentable 2-dimensional Banana dataset (Fig. 2.3). All of our results are provided with the corresponding standard deviations.

We exclude the results of NPROC-LOG in Table 2.1 (instead we keep NPROC-SVM since it generally performs better than NPROC-LOG) due to the page limitation, as the table gets too wide otherwise. One can access the results of NPROC-LOG from our Fig. 2.2. Based on our detailed analysis presented in Table 2.1 along with the visualization with ROC curves in Fig. 2.2, we first conclude that in general the algorithms NPROC-SVM and the proposed NP-NN with powerful nonlinear classification capabilities significantly outperform the linear algorithms OLNP and NPROC-LOG in terms of both AUC and NP-score, hence the proposed NP-NN and NPROC-SVM better address the need for modeling complex decision boundaries in the contemporary applications. This significant performance difference in favor of nonlinear algorithms NPROC-SVM and the proposed NP-NN is much more clear (especially in terms of the AUC) in highly nonlinear datasets such as Banana, Spiral, Iris, SVMguide1 and Fourclass, as shown in Fig. 2.2. In the case of a small size dataset that seems linear or less nonlinear (e.g., Bupaliver), although OLNP and NPROC-LOG are both linear by design and targeting this dataset with the right complexity and hence expected to be less affected by overfitting, the proposed NP-

NN competes with the both well and even slightly outperforms them in terms of AUC (while staying comparable in terms of NP-score). We consider that this is most probably due to the successful compactification of the SLFN in the proposed NP-NN which reduces the parameter complexity by learning the Fourier features in the hidden layer.

As for the comparison between the nonlinear algorithms NPROC-SVM and the proposed NP-NN, we first strongly emphasize that NPROC-SVM has computational complexity (in the worst case of full number of support vectors) between $O(N^2)$ and $O(N^3)$ in training and $O(N)$ in test, where the space complexity is $O(N)$. On the other hand, the proposed NP-NN is truly online without separate training or test phases, which only requires $O(N)$ computational and $O(1)$ negligible space complexity. Hence, NPROC-SVM cannot be applied in our targeted large scale data processing applications due to its prohibitive complexity; nevertheless, we opt to include it in our experiments to set a baseline that is achievable by batch processing. According to the numeric results in Table 2.1 and the ROC curves in Fig. 2.2, we first observe that NPROC-SVM and the proposed NP-NN perform comparably in terms of the AUC, hence our technique (thanks to its computationally highly efficient implementation) can be used in large scale applications (where NPROC-SVM computationally fails) without a loss in classification performance. In addition, our algorithm NP-NN outperforms NPROC-SVM in terms of AUC in 3 datasets; and for small target false positive rate ($\tau = 0.05$), the proposed NP-NN has higher TPR compared to NPROC-SVM in 6 datasets. On the other hand, comparing in terms of the NP-score, NPROC-SVM performs better as a result of enhanced false positive rate controllability due to batch processing. However, this advantage of NPROC-SVM over the proposed NP-NN seems to disappear or decrease as the data size (relative to the dimension) and/or the desired false positive rate increases as observed in the cases of, for instance, Banana and Cod-rna datasets. Therefore, we expect no loss (compared to NPROC-SVM) with the proposed NP-NN in terms of false positive rate controllability as well, when data size increases as in the targeted scenario of the big data applications where NPROC-SVM cannot be used. Indeed, we observe a decent nonlinear classification performance and false positive rate controllability with the proposed NP-NN on, for example, the Banana dataset (5300 instances in only 2 dimensions), as clearly visualized in Fig. 2.3 which shows the false positive rate convergence as well as the nonlinear decision boundaries for various desired false positive rates. Lastly, NPROC-SVM seems to be failing when TFPR requires only a few mistakes in the non-target class. In this case, NPROC-SVM picks zero mistake resulting in zero TPR and a poor NP-score in return. In contrast, the proposed NP-NN successfully handles such situations as demonstrated by, for instance,

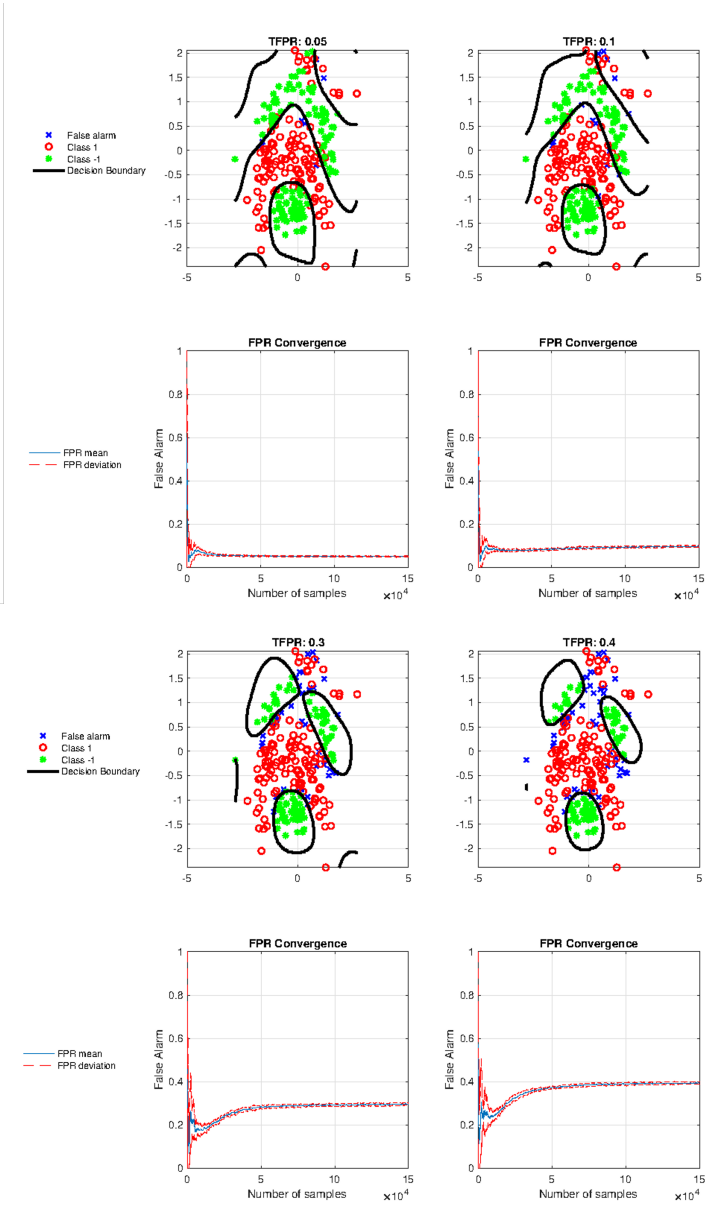


Figure 2.3 Using the visually presentable 2-dimensional Banana dataset, upper graphs show the variation in the decision boundary of the proposed NP-NN as the target false positive rate (TFPR) changes as TFPR $\in \{0.05, 0.1, 0.3, 0.4\}$, and the lower graphs show the mean convergence as well as the standard deviation of the achieved false positive rate (TPR) of the proposed NP-NN over 15 trials of random data permutations with respect to the number of processed instances during training. To better show the convergence, in each trial, length of the training sequence is increased with concatenation resulting in an epoch-by-epoch training of multiple passes. Overall, as indicated by these results, we observe a decent nonlinear modeling as well as a decent false positive rate controllability with the proposed NP-NN.

TFPR=0.05 at Iris dataset in Table 2.1.

Our experiments in Table 2.1, Fig. 2.2 and Fig. 2.3 include comparisons of the pro-

posed NP-NN with certain batch processing techniques (NPROC-SVM and NPROC-LOG). Hence, we utilize separate training and test phases, along with multiple passes over training sequences (due to small sized datasets in certain cases such as Iris), in those experiments for statistical fairness. However, we emphasize that in the targeted scenario of large scale data applications: 1) one can only use computationally scalable online (such as the proposed NP-NN and OLNP) algorithms, 2) multiple passes are not necessary as the data is abundant, and also 3) one can target for even smaller false positive rates such as 0.01 and 0.005. Therefore, to better address this scenario of large scale data streaming conditions, we conduct additional experiments to compare the online methods (OLNP and the proposed NP-NN) when processing 2 large datasets (after z-score normalization and 15 random permutations) on the fly without separate training and test phases based on just a single pass: Covertype (581012 instances in 54 dimensions) and Cod-rna (488565 instances in 8 dimensions, this is the original full scale, for which we previously use in Table 2.1 a relatively small subset for testing the batch algorithms). We run for τ (TFPR) $\in \{0.005, 0.01\}$ and present the resulting TPR and FPR at each time (in a time-accumulated manner after averaging over 15 random permutations) in Fig. 2.4. Parameters are set with manual inspection based on a small fraction of the data.

Although the false positive rate constraint is set harder (i.e. smaller as τ (TFPR) $\in \{0.005, 0.01\}$) in this experiment (compared to the smallest TFPR value 0.05 in Table 2.1), both techniques (OLNP and the proposed NP-NN) successfully converge (the proposed NP-NN appears to converge slightly better) to the target rate (FPR $\rightarrow$ TFPR) uniformly in all cases. Therefore, both techniques promise decent false positive rate controllability (almost perfect) when the data is sufficient. On the other hand, the proposed NP-NN strongly outperforms OLNP in terms of the TPR (again uniformly in all cases), which proves the gain due to nonlinear modeling in the proposed NP-NN. In terms of the NP-score, the proposed NP-NN again strongly outperforms OLNP (except one case, where we observe comparable results). We finally emphasize that the proposed NP-NN achieves this high performance while processing data on the fly in a computation- as well as space-wise extremely efficient manner, in contrast to failing batch techniques in large scale streaming applications due to complexity and failing linear techniques due to insufficient modeling power.

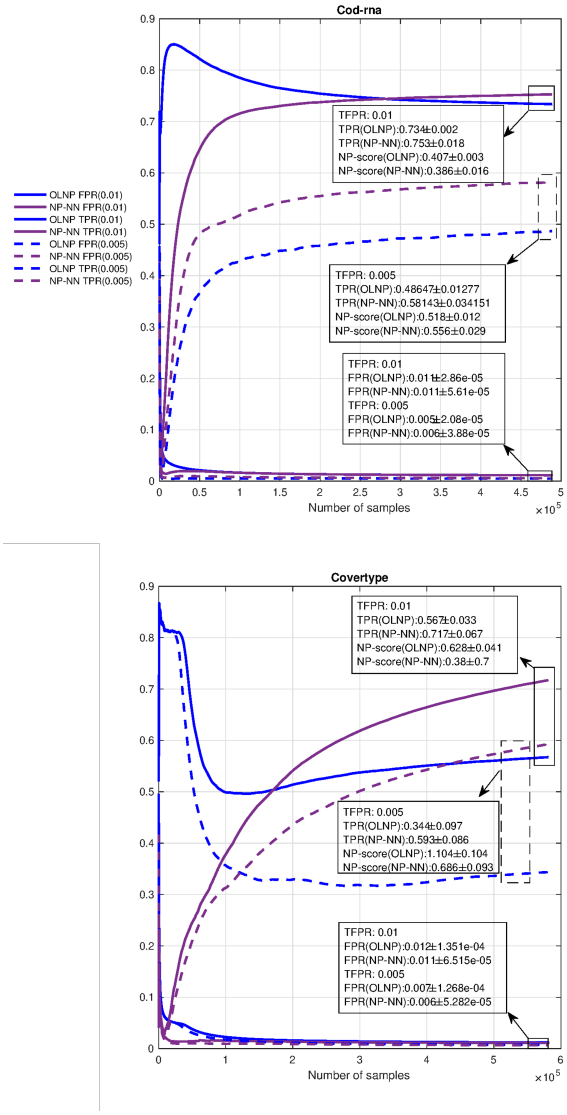


Figure 2.4 We demonstrate the proposed online algorithm NP-NN on two large scale datasets (Cod-rna and Covertype) with relatively small target false positive rates, i.e., TFPR $\in \{0.01, 0.005\}$. The data processing in this case is truly online, and based on only a single pass over the stream without separate training and test phases. Hence, this experiment better demonstrates the typical use-case of our algorithm in large scale scenarios. Time accumulated false positive error rate (FPR) and detection rate (TPR) are obtained after averaging over 15 trials of random data permutations. Overall, we observe that the proposed NP-NN and OLNP are both decent and comparable in terms of the false positive rate controllability, whereas the proposed NP-NN strongly outperforms OLNP in terms of both the detection power and NP-score.

2.5 Analysis of Results and Discussion

In the previous Section 2.4, we have compared our proposed online algorithm NP-NN with the competing algorithms OLNP and NPROC-SVM based on 10 datasets in terms of NP-scores and AUC. In this section, we next provide a statistical significance analysis of the observed performance differences as well as a detailed complexity analysis with running times, and then discuss our findings.

2.5.1 Statistical Significance Analysis

A manual inspection of the error bars (Table 2.1) is already indicative about the significance of the performance differences among the compared algorithms, which reinforces our previous discussions/conclusions. In this section, we further quantify the statistical significance for all pairwise performance differences by following the recommendations of the highly comprehensive study in [84], and opt to use the Wilcoxon signed-rank test (WSRT) for this purpose. Note that a paired t-test can also be considered to measure the significance for each dataset separately across the 15 different runs. However, since such runs are based on random permutations of the dataset with k-fold cross validation, the train/test sets inevitably overlap and the independence assumption of the t-test fails to hold, compromising (under estimating) the standard error estimation. Thus, a paired t-test is not recommended after k-fold cross validation. Then, one can use 5×2 cross validation as a remedy, again followed by the paired t-test. However, the paired t-test has its own weaknesses such as the failing Gaussian assumption when the number of data samples is relatively small (≤ 30), the issue of commensurability and the sensitivity to the outliers. Thus, we use the Wilcoxon signed-rank test (WSRT) across all datasets as recommended (as a better alternative to the paired t-test) in [84]. The advantages of WSRT are: testing across datasets (not separately for each) satisfies the independence, ranking is less sensitive to outliers, Gaussian distribution is not assumed, and the issue of commensurability is relieved as it is considered only qualitatively. We refer to [84] for an elegant treatment of this topic of comparing classifiers statistically.

We conduct the WSRT for all pairwise performance differences (in Table 2.1) of the compared algorithms across all datasets, and present the corresponding T-statistics (of WSRT) in Table 2.2. We consider the AUC differences as well as the NP-

score differences for the target false positive rates $\tau \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. Note that when the T-statistic is less than 3 (or less than 8 but greater than 3), then the difference is highly statistically significant at the level $p < 0.01$ (or still significant but not highly at the level $0.01 < p < 0.05$). We consider that a difference is insignificant if the corresponding T-statistic is greater than 8 since then the significance is $p > 0.05$. The critical values here (3 and 8) are from the two sided t table [84]. We observe that all pairwise performance differences are statistically highly significant, except the three single-starred (Table 2.2) ones that are only significant (not highly) and except the three double-starred ones that are insignificant. Note that the sign "+" (or "-") in Table 2.2 is used to indicate that the performance difference is in favor of the algorithm x (or y) in the comparison x vs y.

This statistical significance analysis firmly supports our performance conclusions in Section 2.4. First, our algorithm strongly (cf. the high performance differences in favor of our online algorithm NP-NN *within* each dataset of Table 2.1) outperforms OLNP (uniformly in all cases of NP-scores or AUC *across* datasets as seen in Table 2.2) and the performance difference is statistically either highly significant (in four cases of Table 2.2) or significant (in the remaining two cases of Table 2.2). Second, our algorithm performs comparably with NPROC-SVM in terms of AUC as well as in terms of NP-score (two cases of $\tau = 0.01$ and $\tau = 0.05$) since the corresponding performance differences are not significant. In the remaining three cases, NPROC-SVM performs better than NP-NN. On the other hand, we emphasize that our online algorithm NP-NN provides huge computational advantages compared to NPROC-SVM despite their, for instance, comparable AUC performance as well as comparable NP-score performances in two cases. We next provide our complexity analysis.

2.5.2 Complexity Analysis

Recall that our algorithm NP-NN is an online algorithm with the computational complexity $O(NDd)$, where N is the number of data instances, $2D$ is the number of hidden units and d is the data instance dimension. The compared algorithm OLNP is also online with the computational complexity $O(Nd)$. Note that the online algorithms (NP-NN and OLNP) do not have separate training and test phases. Whereas the compared NPROC-SVM is a batch algorithm (not online) with the computational complexity $O(N_{\text{tr}}^2 N_{\text{SV}} d)$ in training and $O(N_{\text{test}} N_{\text{SV}} d)$ in test (cf. [85]), where the number of support vectors $N_{\text{SV}} \leq N_{\text{tr}}$ can be as large as N_{tr} (N_{tr} and N_{test} are the number of instances in training and test sets). Hence, the online

	NP-NN vs OLNP	NP-NN vs NPROC-SVM	NPROC-SVM vs OLNP
$\tau = 0.05$	$7^{*}(+)$	$18^{**}(-)$	$1(+)$
0.1	$0(+)$	$16.5^{**}(-)$	$0(+)$
0.2	$0(+)$	$2.5(-)$	$0(+)$
0.3	$6.5^{*}(+)$	$0.5(-)$	$0(+)$
0.4	$2.5(+)$	$1.5(-)$	$0(+)$
AUC	$0(+)$	$25^{**}(+)$	$6^{*}(+)$

Table 2.2 We present the T-statistics of Wilcoxon signed-rank significance tests for the pairwise comparisons of the algorithms based on their performance results (across all 10 datasets) in Table 2.1. The last row is based on the AUC scores whereas the others are based on the NP scores. All performance differences are highly significant with the level $p < 0.01$, except the three single-starred cases where the performance differences are significant (but not highly) since the level is $0.01 < p < 0.05$ as well as the three double-starred cases where the performance differences are considered insignificant since the level is $p > 0.05$. Also, for a comparison x vs y, we use the sign "+" (or "-") if the performance difference is in favor of the algorithm x (y).

algorithms (NP-NN and OLNP) are computationally highly efficient and scalable, however, the batch algorithm NPROC-SVM is prohibitively complex both computationally and space-wise. Note that, also, the online algorithms (NP-NN and OLNP) require negligible space complexity that is only $O(1)$.

We point out that the main complexity difference between the algorithm NPROC-SVM and the online algorithms NP-NN and OLNP stems from the number of data instances (N) whereas the same difference between our proposed algorithm NP-NN and OLNP stems from the number of hidden units ($2D$), i.e., the kernel space expansion cost, in NP-NN. We next observe these differences in terms of the actual running times, and devise an experiment for this purpose that is controlled with respect to the variables N and D. This experiment is based on three datasets with the properties $(n_-, n_+, d) \in \{(5000, 5000, 5), (50000, 50000, 5), (150000, 150000, 5)\}$. Here, D is set as $D = 5 \times d = 25$, $N = n_- + n_+$ and n_+ (n_-) is the number of class 1 (class -1) labeled instances, and the datasets are randomly generated as bimodal Gaussian distributed with random means and covariances around 0. We run the algorithms 10 times on random permutations of such datasets and report in Table 2.3 the mean running times in seconds (s) with the corresponding standard deviations. Note that we report separately for training and test phases in the case of NPROC-SVM. TFPR is set as $\tau = 0.1$. Our online algorithm NP-NN and the compared algorithm OLNP are implemented in MATLAB, and the compared algorithm NPROC-SVM is implemented in Python with the original code provided in [20]. Computations are conducted on a computer containing 2.2 GHz Quad-Core Intel

i7. Our mean running time observations (Table 2.3) are in line with the complexity analysis above. The online algorithms NP-NN and OLNP both run fast where NP-NN takes slightly more time due to the cost of the hidden layer of our SLFN with $2D$ hidden units. Note that, here, we use a straightforward unoptimized MATLAB implementation (for both NP-NN and OLNP), and thus the kernel space expansion cost in the hidden layer of our algorithm NP-NN can be significantly reduced with a more CPU-friendly and optimized implementation in a low level language. Therefore, our algorithm NP-NN can significantly speed up and the running time difference between NP-NN and OLNP can further decrease. On the other hand, the algorithm NPROC-SVM quickly becomes impractical and prohibitively complex, as the required running time polynomially increases as $\sim 2000\times$ when the number of data instances increases only $30\times$. Hence, NPROC-SVM is not scalable.

2.5.3 Discussion

As extensively demonstrated in Section 2.4, Section 2.5.1 and Section 2.5.2, in summary, our performance results (in terms of both the AUC and NP-score) and significance as well as complexity analyses statistically significantly and firmly conclude that our online algorithm NP-NN either outperforms the state-of-the-art (e.g. the compared algorithm OLNP in our experiments) at a comparable complexity, or performs comparably (to the compared algorithm NPROC-SVM in our experiments) while providing huge computational and space advantages. In the following, we explain the underlying reason for the superiority of our proposed algorithm NP-NN over the state-of-the-art (i.e. the competing algorithms OLNP and NPROC-SVM).

The kernel inspired SLFN of the proposed NP-NN has two benefits: expedited powerful nonlinear modeling and scalability. Namely, first, it enables an excellent network initialization as random Fourier features (RFFs) are already sufficiently powerful to learn complex nonlinear decision boundaries even when kept untrained. This speeds up and enhances the learning of complex nonlinearities by relieving the burden of network initialization. Second, the hidden layer is compactified thanks to the exponential rate of improvement in approximating the high dimensional kernel space due to Hoeffding's inequality [53]. As a result, the number of hidden nodes, parameter complexity and the computational complexity of forward-backward network evaluations reduce, and therefore the scalability substantially improves while also mitigating overfitting. Moreover, thanks to the learning of the hidden layer, the randomly initialized Fourier features are continuously improved during SGD steps

Dataset (n_-, n_+, d)	OLNP	NPROC-SVM (train)	NPROC-SVM (test)	NP-NN
Dataset 1 (5000, 5000, 5)	0.197 ± 0.033 s	0.931 ± 0.238 s	0.014 ± 0.001 s	0.547 ± 0.048 s
Dataset 2 (50000, 50000, 5)	1.444 ± 0.028 s	240.153 ± 6.325 s	0.170 ± 0.008 s	3.709 ± 0.091 s
Dataset 3 (150000, 150000, 5)	4.265 ± 0.087 s	1899.047 ± 27.088 s	0.454 ± 0.009 s	10.877 ± 0.134 s

Table 2.3 Mean running times are reported along with standard deviations for all algorithms across 10 different trials. Datasets are generated randomly as bimodal Gaussian distributed with random mean and covariances around 0. Training and test observations are provided separately for the algorithm NPROC-SVM since it is not online.

for even further compactification and better nonlinear modeling. We point out that the competing algorithm NPROC-SVM is also powerfully nonlinear but it does not exploit Fourier features and explicit kernel space expansion for scalability and computational as well as space-wise efficiency. Whereas the competing algorithm OLNP is also online and efficient but it is not designed to model nonlinearities. Hence, our online NP classifier is powerfully nonlinear (which clearly explains the superiority over the competing algorithm OLNP) and computationally highly efficient with $O(N)$ processing and negligible $O(1)$ space complexity (which clearly explains the superiority over the competing algorithm NPROC-SVM), where N is the number of data instances.

While the learning (i.e. optimizing) of Fourier features significantly strengthen the capability of nonlinear modeling, our method can find features that are only locally optimal due to the nonconvexity of the optimization we studied. This appears as a limitation of our technique. On the other hand, those learned locally optimal Fourier features do not necessarily define a proper kernel which satisfies the Bochner's theorem that we use to initialize the network with the radial basis function kernel. This is perhaps not a limitation but certainly an indicator of that our method is not appropriate for learning a nonisotropic rbf kernel. Then, if desired, one can attempt to enforce positive definiteness of the weights in the hidden layer of our SLFN as a constraint to the network optimization, which would be a point of improvement. Similarly, it is true that Fourier features is a powerful means for nonlinear classification but it also makes it prone to overfitting. For this reason, one has to carefully tune the parameters kernel bandwidth and the hidden layer size for which we successfully use extensive cross validations. However, in the case of a very large dimensionality (e.g. images), overfitting may still appear as a limitation. For this issue, we suggest using a strong regularization while incorporating the introduced network as the fully connected layers of a deep architecture. Then, as a remedy, the earlier layers can be designed to extract features and reduce dimensionality for our

network following in the deeper layers.

2.6 Final Remarks

We considered binary classification with particular regard to i) a user defined constraint on the type I error (false positive) rate that requires false positive rate (FPR) controllability, ii) nonlinear modeling of complex decision boundaries, and iii) computational scalability to voluminous data with online processing. To this end, we propose a computationally highly efficient online algorithm to determine the pair of asymmetrical type I and type II error costs to satisfy the FPR constraint and solve the resulting cost sensitive nonlinear classification problem in the non-convex sequential optimization framework of neural networks. The proposed algorithm is essentially a Neyman-Pearson classifier, which is based on a single hidden layer feed forward neural network (SLFN) with decent nonlinear classification capability thanks to its kernel inspired hidden layer. The SLFN that we use for Neyman-Pearson classification is compact in principle for two reasons. First, the hidden layer exploits -during initialization- the exponential convergence of the inner products of random Fourier features to the true kernel value with sinusoidal activation. Second, learning of the hidden layer parameters, i.e., Fourier features, help to improve the randomly initialized Fourier features. Consequently, the required number of hidden nodes, i.e., the required number of network parameters and Fourier features, can be chosen relatively small. This reduces the parameter complexity and thus mitigates overfitting while significantly reducing the computational as well as space complexity. Then the output layer follows as a perceptron with identity activation. We sequentially learn the SLFN parameters through stochastic gradient descent based on a Lagrangian non-convex optimization to goal of Neyman-Pearson classification. This procedure minimizes the type II error rate about the user specified type I error rate, while producing classification decisions in the run time. Overall, the proposed algorithm is truly online and appropriate for contemporary fast streaming data applications with real time processing and FPR controllability requirements. Our online algorithm was experimentally observed to either outperform (in terms of the detection power and false positive rate controllability) the state-of-the-art competing techniques with a comparable processing and space complexity, or perform comparably with the batch processing techniques, i.e., not online, that are -however- computationally prohibitively complex and not scalable.

3. Online Neyman-Pearson Classification with Hierarchically Represented Models

Neyman-Pearson (NP) classification framework [16] is widely used in areas such as video anomaly detection [86, 87] and medical diagnosis [88, 89], where false alarm rate controllability is essential and requires asymmetrical cost assignment to the type I (false alarm) and type II (miss) errors. In cyber fraud detection [90, 91] for instance, missing a suspicious activity of a client (miss) might have severe security consequences whereas predicting a regular activity as suspicious (false alarm) leads to a tiring unnecessary measure. The CFAR (constant false alarm rate) detection in radar applications [92] is another example among numerous others developed in the past decades [16]. The NP framework appears as a viable approach in such situations as it minimizes one of the error rates (typically the miss rate) under a user-specified upperbound constraint on the other (typically the false alarm rate). The classification solution in this framework is then the minimization of the Bayes risk (cf. the NP lemma [17]) with unknown asymmetrical costs satisfying the error constraint.

In this chapter, we introduce a novel online NP classifier for false alarm controllability matching to the user-specified level, while addressing the two important practical needs of fast stream data applications: Dynamical modeling power that is optimally tuned to the learnability that the available data allows at a time and computational scalability for real time processing. In particular, our method constructs a rich hierarchical ensemble of expert algorithms via a binary tree partitioning of the observation space, and obtains a randomized mixture of experts [93, 94] as the proposed NP classification algorithm. We mathematically show that the proposed algorithm asymptotically achieves the NP performance of the best expert for any input stream.

3.1 Related Work

Neyman-Pearson (NP) classification is a statistical framework that studies problems with asymmetrical type I (predicting non-target as target, false positive / alarm) and type II (predicting target as non-target, false negative or miss) error costs [16], with the main purpose of controlling the false alarm rate by guaranteeing that it does not exceed a certain level provided by the user. In the data driven cost sensitive learning, the asymmetry is directly incorporated to the empirical error minimization [18, 95], where the drawback is that the correspondence between the desired false alarm rate and the right cost structure is typically unknown. Another approach is to use plug-in methods to estimate the class specific densities and apply the log likelihood ratio test with the right threshold that matches the desired false alarm rate [24, 20]. The same drawback is still in effect since the issue of unknown cost structure in the cost sensitive learning translates to the issue of unknown threshold in the plug-in methods. Likewise, the neural network in [24] estimates the densities and leaves the threshold as a parameter to manual tuning. In [20], authors introduce an umbrella algorithm that can incorporate any given batch classifier into the NP framework. Their method succeeds to remove the additional work required for selecting the suitable threshold manually. However, this umbrella algorithm [20] as well as the plug-in studies [24, 20] are computationally prohibitive and thus experience scalability issues when the data is voluminous or streaming fast due to the multiple passes in their batch processing.

Similar to these prominent examples, the vast majority of the existing NP methods in the literature either fail at addressing the false alarm rate controllability without manual tuning or do not scale to large amount fast streaming data. Nevertheless, the online classifier in [22] addresses the both through two stochastic gradient descent (SGD) updates for a Lagrangian minimax resulting from the constrained optimization of the NP classification. The first update is for the classifier parameters, and the second is for estimating the asymmetrical costs. The outcome is an online linear NP classifier (OLNP) that is scalable with constant processing complexity per instance, but it is incapable of modeling nonlinear class separations. The authors [22] has left the handling of nonlinear separations to a future study. As a follow up to that within the formulation of OLNP, we introduced an NP classifier (NP-NN) in [96] based on a single hidden layer feed forward neural network (detailed explanation is available in Chapter 2) for extending to the nonlinear settings as an initial solution which utilizes the random Fourier features [53] expansion of the radial basis function (RBF) kernel. Our NP-NN is online (data are received sequentially, and each

instance is discarded after it is looked only once), scalable allowing real time processing, capable of nonlinear modeling and successfully controls the false alarm rate without manual tuning. Updates of NP-NN are for the Fourier feature projection (first layer) and the classifier (second layer) parameters as well as the asymmetrical costs. However, as a disadvantage, NP-NN requires cross-validation for the kernel bandwidth as well as the number of Fourier features, necessitating a separate phase before the actual processing starts.

The focus of this chapter is the nonlinear data modeling in the context of online (i.e. scalable to large fast stream data) NP classification. To that end, we linearly combine an ensemble of online piecewise linear (so nonlinear) NP classifiers (i.e. experts as piecewise OLNPs) of varying modeling powers (i.e. varying number of pieces). The combination weights are time varying and proportional to the accumulated NP performances of the experts. Therefore, simpler models take larger weights early in the stream and more complex models start dominating as time progresses. This implements a desired gradual increase in the modeling power that is dynamically adaptive to the learnability implied by the size of the available data at a time. We mathematically show that the NP performance difference between our mixture of experts algorithm and that of the best expert vanishes asymptotically in the order $O(\frac{1}{\sqrt{T}})$ (T is the total number of processed instances), where the vanishing difference characterizes the difficulty of the NP classification (the NP problem becomes more difficult as $(\tau, \pi^-) \to (0,0)$). Our proposed algorithm successfully controls the false alarm rate at any specified level, it is dynamically optimal in its modeling power and computationally highly efficient with online processing of complexity $O(T \log\log N_q)$, (N_q is the number of experts). In order to design the expert ensemble (as piecewise OLNPs; piecewise low complexity NP-NNs are also a possible design) and cutting down the processing complexity from $O(TN_q)$ to $O(T \log\log N_q)$, we use the well-known context tree weighting (CTW) technique [97] as a binary tree partitioning of the observation space. We call our algorithm (proposed in this manner) as Tree OLNP, which falls into the category of cost sensitive learning with the stochastic updates described above. It is important to mention that one can also consider and implement Tree NP-NN, similar to the Tree OLNP, but it is out of the scope of this chapter. To our best knowledge, the proposed NP classification approach here acquires these properties and performance guarantees as the first time in the literature which draws the main contribution of the presented study.

The celebrated CTW technique was originally introduced in [97] for sequential universal data compression and coding, which has been later adapted to various applications such as Bayesian inference in Markov chain for modeling discrete time sequences [98], recommendation systems [99], kernel design for string classification

[100] and reinforcement learning [101]. Examples that are most relevant to our study are in [102, 103, 104, 105]. The CTW partitions the observations space in [102] to obtain a piecewise linear regression that is twice universal in the space of all linear regressors as well as the ensemble of piecewise linear regressors over the tree. The binary context tree is generalized to nonbinary trees in [104] for multiarmed bandit problems. On the other hand, the CTW is used in [103] for regular binary classification while also optimizing the tree split parameters (hence, self organizing; we use iterative PCA in this study for a sound splitting) whereas [105] studies CTW based kernel density estimation for anomaly detection. We strongly emphasize that the problem considered in these studies and the problem we consider in the presented study are completely different. None of them considers the estimation of asymmetrical costs for false alarm rate controllability, and thus, we are the first in the literature to generalize the classification with the CTW into the Neyman-Pearson framework.

3.2 Problem Description, Highlights and Novel Contributions

We introduce a randomized binary classification algorithm g to classify an observation $\boldsymbol{x} \in \mathbb{R}^d$ as $\hat{y} \triangleq g(\boldsymbol{x}) \in \{1, -1\}$ (predicted label), where $p_{\boldsymbol{X},Y}$ is the joint distribution that an i.i.d. labeled data stream $(\boldsymbol{x} \in \mathbb{R}^d, y \in \{-1, 1\}) \sim p_{\boldsymbol{X},Y}$ follows. The proposed algorithm is desired to have the following properties.

P1 *False alarm (or false positive) controllability:* The detection power is maximized at a user-specified false alarm rate (type I error). We address this by using the Neyman-Pearson (NP) formulation from the detection estimation theory [17]. Our algorithm solves the optimization problem

$$\max_{g} \mathrm{P}_d(g) \text{ subject to } \mathrm{P}_{fa}(g) \leq \tau, \tag{3.1}$$

where $\mathrm{P}_d(g) = \mathrm{E}[\mathrm{sgn}(g(\boldsymbol{X}))|Y = 1]$[1] is the detection power, $\mathrm{P}_{fa}(g) = \mathrm{E}[\mathrm{sgn}(g(\boldsymbol{X}))|Y = -1]$ is the false alarm rate, τ is the desired user-specified false alarm rate (TFPR: target false positive rate), and $\mathrm{E}[\cdot]$ is the expectation operator with respect to the data distribution as well as the randomization in the algorithm. The solution can be obtained through the NP-lemma [17] that also draws an *NP*

[1] Let $\mathrm{sgn}(\cdot)$ be the sign function returning 1 if its argument is nonnegative and 0 otherwise.

loss (explained in the next section) measuring the level of optimality in (3.1). We call a classifier having this property **P1** as an *NP classifier* [16].

P2 *Optimally dynamic data modeling power:* The proposed algorithm operates at the right data modeling power (degree of nonlinearity in separation) by dynamically adapting to the amount of available data in an NP-optimal manner (based on the NP loss). We address this with a randomized mixture of experts [93]:

$$g(\boldsymbol{x}) = 2 \times \text{sgn}(f_i(\boldsymbol{x})) - 1 \text{ with probability } q_i \text{ for } 1 \leq i \leq N_q, \tag{3.2}$$

where f_i's are the nonrandom and continuous valued (akin discriminants), i.e., $f_i(\boldsymbol{x}) \in \mathbb{R}^d$, experts solving the same classification problem at varying modeling powers (at varying nonlinearities) as $\hat{y}_i = 2 \times \text{sgn}(f(\boldsymbol{x})) - 1 \in \{-1, 1\}$, and q_i's are the expert randomization weights such that $\sum_{i=1}^{N_q} q_i = 1$ and $q_i \geq 0 \; \forall i$. The proposed algorithm adjusts the weights in a way that the powerful experts achieve larger weights when enough data is available. Otherwise, simpler (less powerful) experts are favored. This property (**P2**) helps us mitigate the issue of overfitting.

P3 *Computational scalability:* The proposed algorithm is designed computationally highly efficient for ensuring scalability to large-scale data. In particular, we use online processing together with a tree-based hierarchical representation of the experts. Given an i.i.d. fast stream data $\{(\boldsymbol{x}_t, y_t)\}$, the instance $\boldsymbol{x}_t$ is received at time t, the prediction $\hat{y}_t = g_{t-1}(\boldsymbol{x}_t)$ is made, and g_{t-1} is updated based on the error $y_t - \hat{y}_t$, yielding g_t in an online manner with linear $O(T)$ computational complexity as desired (T is the total number processed instances). On the other hand, the number of experts N_q is typically chosen large to guarantee sufficient data modeling power, but this might be computationally prohibitive since the proposed algorithm requires (when naively implemented) complexity $O(N_q)$. To greatly cut down the computational complexity to $O(\log \log N_q)$ in a twice-logarithmic manner, we use the binary partitioning tree based data processing framework of [103, 102] which designs an ensemble of experts (piecewise linear classifiers) of varying modeling powers (varying pieces).

Our main contribution in this chapter is to propose, as the first time in the literature, an algorithm that satisfies all these properties (P1, P2 and P3) at once. We use (1) the templates of online linear NP classifier (OLNP) of [22] for satisfying the property P1, (2) mixture of piecewise OLNPs across which our algorithm provides a dynamically optimal weighting in the competitive sense in terms of the NP performance for satisfying the property P2, and (3) the binary partitioning tree of [103] to design a hierarchically represented ensemble for satisfying the property P3. We emphasize that the tree based classification in [103] does not have the NP property

(P1) whereas we specifically consider here the false alarm rate controllability which is a fundamentally important difference (between [103] and the presented study). We also emphasize that the studies [22] and [96] have the NP property (P1) but they are not optimally dynamic in terms of the nonlinear modeling power (lacking the property P2), which is another fundamentally important difference (between [22], [96] and the presented study). While satisfying these properties in our methodology, we mathematically prove that our algorithm is competitive against the expert ensemble in terms of the NP loss. Namely, our algorithm approaches the NP performance of the best expert as more data become available while matching to the desired false alarm rate which has been achieved as the first time in the literature, to our best knowledge.

3.3 Method

In this section, we present our method and construct the proposed algorithm by satisfying the desired properties.

3.3.1 False Alarm Controllability: Neyman-Pearson (NP) Classification

We start with explaining the false alarm controllability with the notation "f" below to denote an NP classifier, which distinguishes our final randomized binary valued classifier g from any generic nonrandom continuous valued NP classifier f. It is known by the NP lemma that the solution f^* to the optimization problem in (3.1) is given by the likelihood ratio test (LRT) when the threshold is chosen appropriately, whereas LRT is known to minimize the Bayes risk [17]. Hence,

$$f^* = \arg\max_f \mathrm{P}_d(f) \text{ subject to } \mathrm{P}_{fa}(f) \leq \tau$$

yields LRT as

$$\begin{aligned} 2 \times \mathrm{sgn}(f^*(\boldsymbol{x})) - 1 &= \hat{y} \\ \hat{y} = 1 \Leftrightarrow \frac{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|1)}{p_{\boldsymbol{X}|Y}(\boldsymbol{x}|-1)} &\geq \hat{\mu}\frac{\pi^-}{\pi^+} \text{ subject to } \mathrm{P}_{fa}(f^*) = \tau, \end{aligned} \tag{3.3}$$

where π^-, π^+ and $\hat{\mu}$ are negative (non-target), positive (target) class prior probabilities and unknown class cost for satisfing the target false alarm rate (τ), respectively. After LRT, we define Bayes risk minimization

$$f^* = \arg\min_f \hat{\mu}_\tau \pi^- \mathrm{P}_{fa}(f) + \pi^+ \mathrm{P}_{nd}(f) \tag{3.4}$$

with $\hat{\mu}_\tau \geq 0$ such that $\mathrm{P}_{fa}(f^*) = \tau$. In 3.4, $\hat{\mu}$ is the unknown cost of deciding 1 when in fact $y = -1$ (false alarm or false positive or type-I error cost) and it is to be estimated such that $\mathrm{P}_{fa}(f^*) = \tau$. Thus, $\hat{\mu}$ is replaced with $\hat{\mu}_\tau$. In addition, within the same equation (equation 3.4), we define $\mathrm{P}_{nd}(f) = 1 - \mathrm{P}_d(f)$ as the no detection rate (the type-II error or miss rate). Using equation 3.4, we can write the empirical risk as

$$\begin{aligned} f^* = \arg\min_f \hat{\mu}_\tau \frac{n_T^-}{T} \frac{1}{n_T^-} \sum_{t=1}^{T} \mathrm{sgn}(f(\boldsymbol{x}_t)) \mathrm{sgn}(-y_t) + \\ \frac{n_T^+}{T} \frac{1}{n_T^+} \sum_{t=1}^{T} \mathrm{sgn}(-f(\boldsymbol{x}_t)) \mathrm{sgn}(y_t). \end{aligned} \tag{3.5}$$

In order to achieve 3.5, we replace false alarm $\mathrm{P}_{fa}(f)$ and miss $\mathrm{P}_{nd}(f)$ rates as well as the prior probabilities (π^-, π^+) in equation 3.4 with their corresponding empirical estimates

$$\begin{aligned} \hat{\mathrm{P}}_{fa}(f) &= \frac{1}{n_T^-} \sum_{t=1}^{T} \mathrm{sgn}(f(\boldsymbol{x}_t)) \mathrm{sgn}(-y_t) \\ \hat{\mathrm{P}}_{nd}(f) &= \frac{1}{n_T^+} \sum_{t=1}^{T} \mathrm{sgn}(-f(\boldsymbol{x}_t)) \mathrm{sgn}(y_t) \\ \hat{\pi}^- &= \frac{n_T^-}{T} \\ \hat{\pi}^+ &= \frac{n_T^+}{T}, \end{aligned}$$

where n_T^- (and n_T^+) is the number of data instances with the label -1 (and 1) out of T observations. To further simplify 3.5, we can substitute $\mu_\tau = \hat{\mu}_\tau$ if $y_t = -1$ and $\mu_\tau = 1$ if $y_t = 1$ and achieve

$$f^* = \arg\min_f \frac{1}{T} \sum_{t=1}^{T} \mu_\tau \mathrm{sgn}(-y_t f(\boldsymbol{x}_t)). \tag{3.6}$$

In order to obtain a differential loss function for the optimization in (3.6), we replace the binary valued $\mathrm{sgn}(-y_t f(x_t)) \in \{0, 1\}$ with the continuous valued sigmoid

$$[0, 1] \ni \text{ł}(-y_t f(x_t)) = \frac{1}{1 + e^{y_t f(\boldsymbol{x}_t)}},$$

and arrive at the optimization

$$
\begin{aligned}
f^* &= \arg\min_f \frac{1}{T}\sum_{t=1}^{T} \mu_\tau l(-y_t f(\boldsymbol{x}_t)) \\
&= \arg\min_f \frac{1}{T}\sum_{t=1}^{T} s_{t+1} \\
&= \arg\min_f L(f).
\end{aligned} \tag{3.7}
$$

This optimization is solved by the stochastic gradient descent based steps:

$$
f^* \leftarrow f_t = f_{t-1} - \beta_f \nabla_{f_{t-1}} \mu_{\tau,t-1} l(-y_t f_{t-1}(\boldsymbol{x}_t)) \tag{3.8}
$$

with $\beta_f > 0$ being the step size. As for the update for μ_τ, since $\hat{\mathrm{P}}_{fa} \simeq l(f(\boldsymbol{x}_t))$ can be seen as an estimate for $\mathrm{P}_{fa}(f)$ based on a single instance and since the false alarm rate cost $\hat{\mu}_\tau$ should be increased if $l(f(\boldsymbol{x}_t))$ exceeds the user-specified desired false alarm rate τ (and decreased otherwise), we can estimate $\hat{\mu}_\tau$ through stochastic updates as in [22] by

$$
\hat{\mu}^* \leftarrow \hat{\mu}_{\tau,t} = \hat{\mu}_{\tau,t-1} + \beta_\mu (l(f_{t-1}(\boldsymbol{x}_t)) - \tau) \text{ only when } y_t = -1 \tag{3.9}
$$

with $\beta_\mu > 0$ being the step size, which also sets $\mu_{\tau,t}$. Here we designate

$$
s_t = \mu_{\tau,t-1} l(-y_t f_{t-1}(\boldsymbol{x}_t))
$$

as the instantaneous or stochastic NP loss measuring the level of optimality when accumulated in time.

To obtain the *online linear NP classifier (OLNP)* of [22], one can simply use $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$ with $\boldsymbol{w} = [\tilde{\boldsymbol{w}}, b]^T$, where $\tilde{\boldsymbol{w}}$ denotes the linear projection parameters and b is the bias. Then, based on a data stream $\{(\boldsymbol{x}_t, y_t)\}$ and the updates in (3.8) and (3.9) for estimating the parameters $\boldsymbol{w}$, we sequentially train for $f_t(\cdot)$ in an online manner. To obtain an online nonlinear NP classification, we opt to use the single hidden layer feedforward neural network (SLFN) of our work [96] which is an expansion of the radial basis function (RBF) kernel $k(\boldsymbol{x}_t, \boldsymbol{x}_{t'}) = e^{-\Omega||\boldsymbol{x}_t - \boldsymbol{x}_{t'}||^2}$ via the random Fourier features (RFF) [53]. We call this classifier *online nonlinear neural network based NP classifier (NP-NN)*. Together with an appropriately chosen kernel bandwidth parameter Ω, NP-NN can model any nonlinearity as the size M of the random Fourier features increases. More information can be found in Chapter 2 but complete description is available in [96]. Both of the algorithms OLNP and NP-NN have the property of false alarm controllability, and they both employ a typical regularization by also minimizing $\lambda\frac{||f||^2}{2}$, i.e., penalizing the squared magnitude of

the classifier parameters where λ is the regularization constant. In the scope of this chapter, we only use OLNP as a template in our design of the expert ensemble ($\{f_i\}_{i=1}^{N_q}$ in (3.2)) but, it is also possible to use NP-NN. Goal of this chapter is to introduce an ensemble model capable of increasing its complexity in a data driven manner. We design our classifier in a way that only the context tree framework is used to introduce nonlinearity to the ensemble model. Therefore, we are opt to use only OLNP (not NP-NN) as the template classifier. Ensemble model with NP-NN template can also be used in further studies where Tree OLNP fails to achieve desired performance. We left that topic open of future research.

In this section, we obtained the derivations of the NP classifiers based on the NP lemma, in the detection/estimation theoretical framework, which allowed us to use the original inequality constrained in (3.1) as an equality constraint and we were able to parameterize $\hat{\mu}$ as $\hat{\mu}_\tau$ as a function τ. Similar end results are obtained in [22] through the Lagrange optimization by directly considering the original inequality constraint. We used their $\hat{\mu}$ updates here. We stress that our particular purpose in these derivations was to obtain the stochastic NP-loss which we use in the following section in our performance analysis.

3.3.2 Optimally Dynamic Data Modeling Power: Ensemble of Experts

Linear NP classifier (OLNP) is designed to operate at a fixed modeling power, which gives rise to problems if we consider the well-known central concept of bias-variance trade-off, or simply the issue of overfitting. OLNP (NP-NN) is eventually too simple (too powerful) for data with nonlinear (linear) separation. Moreover, when given some data at time T from a data stream $\{(\boldsymbol{x}_t, y_t)\}$, a linear classifier (e.g. OLNP) suffers from high bias (or high approximation error or simply underfitting) when the true separation is nonlinear. One can think of using a nonlinear classifier (e.g. NP-NN) in this case, but then at an earlier time when given less data, the nonlinear model potentially suffers from high variance (or high estimation or simply overfitting), as illustrated in Fig. 3.1.

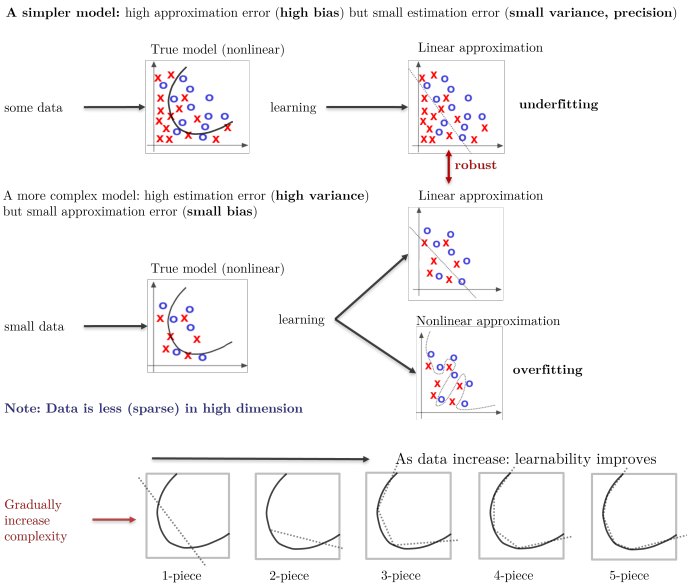


Figure 3.1 Bias-variance trade-off, and optimally dynamic data modeling power: Ensemble of experts

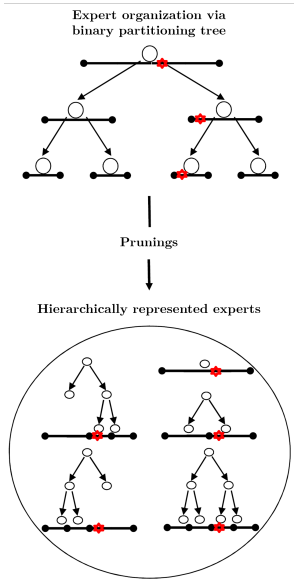


Figure 3.2 Hierarchical expert ensemble

To that end, we introduce an optimally dynamic (optimally varying) data modeling power through a finite ensemble of piecewise linear classifiers, i.e., ensemble of experts. An η-piece piecewise linear classifier $f_{\mathcal{P}}$ utilizes a partition $\mathcal{P} = \{R_1, R_2, \cdots, R_\eta\}$ of the observation space $\mathbb{R}^d$, where we have $R_k \cap R_{k'} = \emptyset$ for $k \neq k'$ and $\bigcup_{k=1}^{\eta} R_k = \mathbb{R}^d$ for the partition regions, and keeps a separate OLNP f_R for each partition region R. Figure 3.3 explains separate regions and possible partitions for a context tree of depth 2 using visually representative banana dataset. The update of the false alarm cost $\hat{\mu}$ in (3.9) is conducted based on the complete set of observations in order for all partition region OLNPs to have access to the same cost structure, whereas each partition region OLNP (say f_R) conducts the parameter update in (3.8) with only those instances falling in the respective region (with only the instances $\boldsymbol{x}_t \in R$). Overall, the entire set of parameters $\bigcup_{k=1}^{\eta} \boldsymbol{w}_{R_k}$ are learned. The resulting $f_{\mathcal{P}}$ is an online η-piece piecewise linear (so nonlinear) OLNP, whose degree of nonlinearity is η (the number of pieces or partition regions it employs). Thus, $f_{\mathcal{P}}(x) = f_R(x)$ if $x \in R$. Note that if $\eta = 1$, then $f_{\mathcal{P}}$ is just a regular OLNP, whereas as η increases it acquires the power of modeling any smooth nonlinearity (cf. Fig. 3.1) as an η-piece piecewise linear (overall nonlinear).

Given an ensemble of partitions $\{\mathcal{P}_i\}_{i=1}^{N_q}$ with various corresponding pieces $\{\eta_i\}_{i=1}^{N_q}$, an ensemble of piecewise OLNPs $\{f_{\mathcal{P}_i}\}_{i=1}^{N_q}$ with various degrees of nonlinearities (with varying modeling powers) follows. We call $f_{\mathcal{P}_i}$ as an expert and denote it by f_i (reducing the subscript for simplicity). Weights of each piecewise OLNP (expert) is dynamically optimized in a data driven way. Ensemble model favors complex models (if necessary) as the total number of processed sample increases.

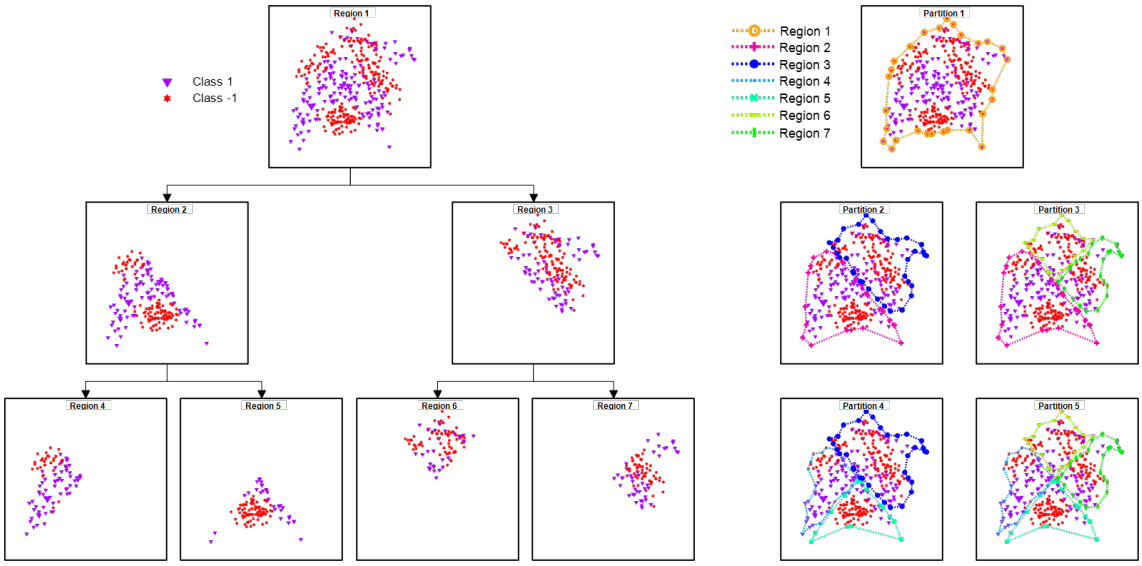


Figure 3.3 We use 2D banana dataset to visualize separation of feature space into different regions via iterative PCA method as explained in Section 3.3.3. For a tree of depth 2, the feature space is separated into 7 disjoint regions and it is possible to represent the whole feature space with 5 different combinations of the created regions. Since we train OLNP models at each region, every different region combination (partition, expert) represents piecewise OLNP (or any other template classifier, e.g NP-NN) model.

In order to better explain dynamic control of the model complexity, we propose 2 datasets at different complexities containing unit variance Gaussian distributions with different mean vectors in Figure 3.4 and 3.5. In both of the figures, upper graphs represent the synthetic datasets. First dataset is simpler (it only contains 2 Gaussian distributions) and can be solved with a linear classifier such as perceptron. In other words, ensemble model can use prediction of the simplest expert (OLNP trained at root node) to have high detection power. With a slight variation, in Figure 3.5, dataset contains 4 different data clusters, each sampled from different Gaussian distributions. Latter problem can not be solved with a simple perceptron so ensemble model needs to adjust its complexity to achieve maximum detection power. We investigate dynamic complexity control of Tree OLNP by analyzing transient response of the model during learning for $\tau = 0.1$.

In our experiments, we use Tree OLNP of depth 5. Figures on the bottom left (both for Figure 3.4 and 3.5) visualize how ensemble model adjusts weight of each expert. Simplest expert of the ensemble is the root node and complexity increases as more data is processed by Tree OLNP. In Figure 3.4, model converges to root node (simplest model), where classification is conducted by a single OLNP. Complexity convergence is also visible from the Figure 3.4, bottom left, where weight of the simplest model quickly dominates over the other experts. It is important to note

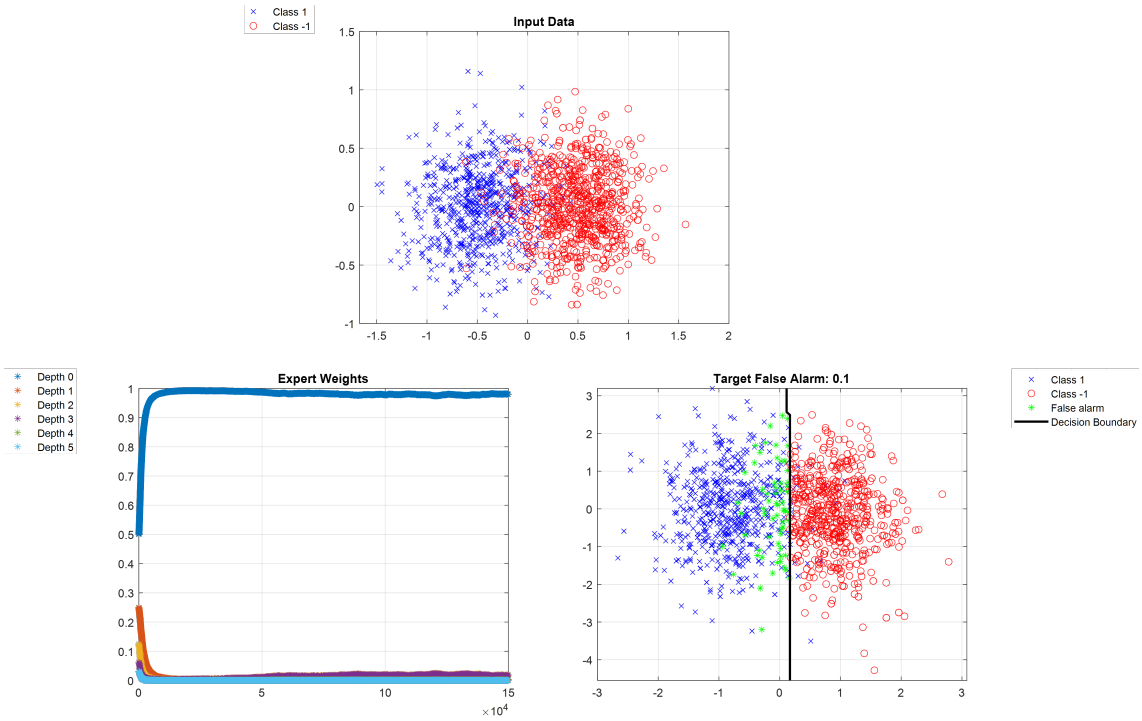


Figure 3.4 Visualization of dynamic data modelling power. Ensemble model converges to the simplest expert (root node), since the classification problem is solvable with a single OLNP (more complex experts are not required). Even the ensemble model has access to more complex experts, the end predictions are calculated from the root node.

that even there are more complex models that Tree OLNP can use, the overall complexity is dynamically adjusted according to the problem.

In the second phase of the experiment, we use slightly harder problem where the optimum solution is not the simplest expert (partition), and we expect Tree OLNP to optimize its complexity to maximize detection power while upper bounding the target false alarm rate by a threshold specified by the user ($\tau = 0.1$). As seen in the Figure 3.5, weight of the simplest model quickly vanishes as Tree OLNP processes more data. We observe that optimum complexity is achieved by using partitions from depth 3. Similar to the previous results, even though we have more complex experts available, Tree OLNP dynamically controls its complexity and does not use overly powerful models in calculating its prediction.

We leave the question of how to design the ensemble of partitions to the next section, and continue with our proposed randomized NP classifier (previously defined in (3.2)) as a mixture of experts in the online setting here with the subscript t:

$$g_t(\boldsymbol{x}) = 2 \times \text{sgn}(f_{i,t}(\boldsymbol{x})) - 1 \text{ with probability } q_{i,t} \text{ for } 1 \leq i \leq N_q. \tag{3.10}$$

At any time $t \geq 1$, the accumulated NP loss $S_{i,t}$ as well as the NP performance $U_{i,t}$

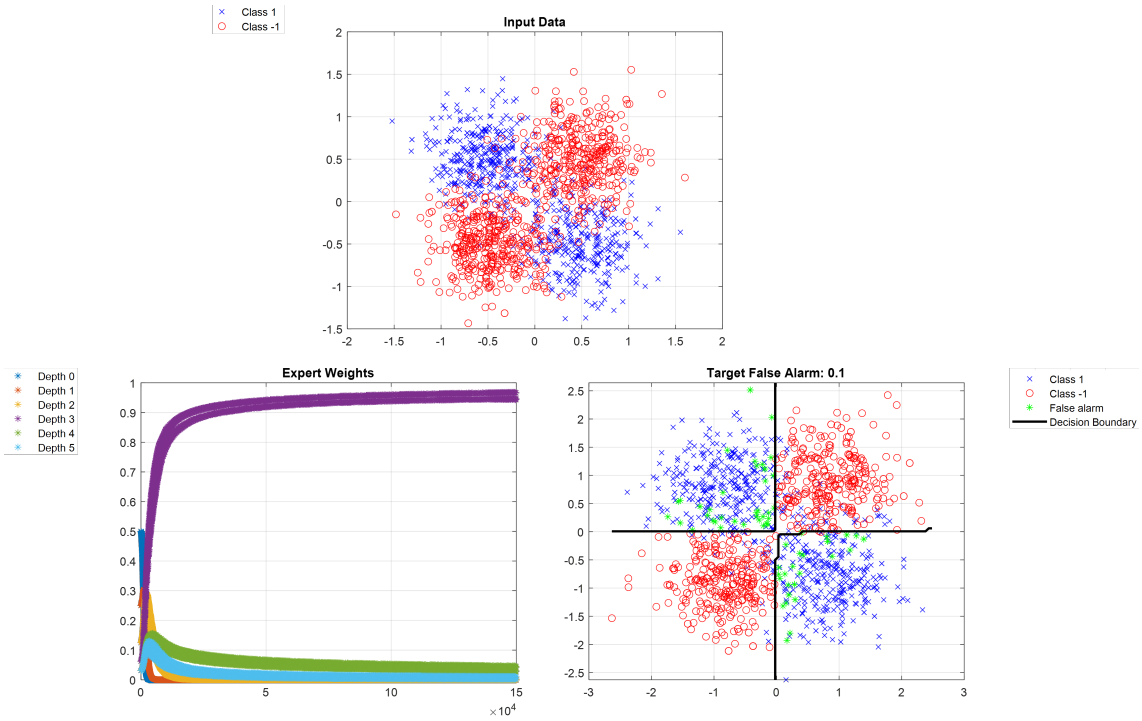


Figure 3.5 Visualization of dynamic data modelling power, similar to the Figure 3.4 with an important difference of having the original problem more complex. In other words, it is not possible to solve this problem using single OLNP (simplest expert, root node), therefore it is expected from Tree OLNP to converge more complex models. As seen from the expert weight convergence (bottom left), weight of the root node diminishes very quickly and predictions generated from depth 3 are being used. Similar to the previous case, even though there are more complex models involved, Tree OLNP dynamically controls its modelling power by selecting optimally complex expert.

(i.e. quality) of the expert f_i can be measured by

$$\begin{aligned} S_{i,t} &= \sum_{t'=1}^{t} s_{i,t'} \\ &= S_{i,t-1} + s_{i,t} \end{aligned} \tag{3.11}$$

with the initial $S_{i,0} = 0$ (accumulated NP loss at $t = 0$ should be 0) and

$$\begin{aligned} U_{i,t} &= q_{i,0} \exp(-hS_{i,t}) \\ &= U_{i,t-1} \times \exp(-hs_{i,t}), \end{aligned} \tag{3.12}$$

respectively, where $s_{i,t}$ is the instantaneous or stochastic NP loss, $h > 0$ is a learning parameter for tree framework (we would like to emphasize that learning rate h is different from the learning rate of the template classifier) and $q_{i,0}$'s are initial probability assignments ($\sum_{i=1}^{N_q} q_{i,0} = 1$ and $q_{i,t} \geq 0$, $\forall t \geq 1$). These probability values are initiated according to the tree structure [102] and then they scale inversely

proportional to the expert powers with the streaming data. The critical observation regarding this probability assignment is that, intuitively, simpler experts outweigh in the beginning of the stream and more powerful experts gradually dominate as time progresses, implementing the idea in Fig. 3.1 just as desired. We can define these probabilities using the accumulated NP performance 3.11 as

$$q_{i,t} = \frac{U_{i,t-1}}{\sum_{j=1}^{N_q} U_{j,t-1}}. \tag{3.13}$$

The instantaneous loss of the ensemble model can be calculated as

$$\sum_{i=1}^{N_q} q_{i,t} \exp(-h s_{i,t}) = E[\exp(-h s_{i,t})], \tag{3.14}$$

for which we can define an upper bound using Hoeffding's inequality [106] as

$$E[\exp(-h s_{i,t})] \leq \exp(-h E[s_{i,t}] + h^2 C^2/8), \tag{3.15}$$

where C is a constant upper bounding $C \geq |s_{i,t}|$ for all i and t. First we redefine right hand side (RHS) of the equation 3.14 in terms of accumulated performance $U_{i,t}$ as

$$\begin{aligned} \sum_{i=1}^{N_q} q_{i,t} \exp(-h s_{i,t}) &= \sum_{i=1}^{N_q} \frac{U_{i,t-1}}{\sum_{j=1}^{N_q} U_{j,t-1}} \exp(-h s_{i,t}) \\ &= \frac{\sum_{i=1}^{N_q} U_{i,t-1}}{\sum_{j=1}^{N_q} U_{j,t-1}} \exp(-h s_{i,t}) \\ &= \frac{\sum_{i=1}^{N_q} U_{i,t}}{\sum_{j=1}^{N_q} U_{j,t-1}}. \end{aligned} \tag{3.16}$$

Now we update left hand side (LHS) of 3.15 as

$$\frac{\sum_{i=1}^{N_q} U_{i,t}}{\sum_{j=1}^{N_q} U_{j,t-1}} \leq \exp(-h E[s_{i,t}] + h^2 C^2/8). \tag{3.17}$$

Introduce telescoping series as

$$\prod_{t'=1}^{t} \frac{\sum_{i=1}^{N_q} U_{i,t'}}{\sum_{j=1}^{N_q} U_{j,t'-1}} \leq \prod_{t'=1}^{t} \exp(-h E[s_{i,t'}] + h^2 C^2/8) \tag{3.18}$$

Simplification of the following series

$$\prod_{t'=1}^{t} \frac{\sum_{i=1}^{N_q} U_{i,t'}}{\sum_{j=1}^{N_q} U_{j,t'-1}} = \frac{\sum_{i=1}^{N_q} U_{i,t'}}{\sum_{j=1}^{N_q} U_{j,t'-1}} \times \frac{\sum_{i=1}^{N_q} U_{i,t'-1}}{\sum_{j=1}^{N_q} U_{j,t'-2}} \cdots \frac{\sum_{i=1}^{N_q} U_{i,1}}{\sum_{j=1}^{N_q} U_{j,0}} \tag{3.19}$$

yields

$$\begin{aligned} \sum_{i=1}^{N_q} U_{i,t} &\leq \exp(-hE[\sum_{t'=1}^{t} s_{i,t'}] + th^2C^2/8) \\ &\leq \exp(-hE[S_{i,t'}] + h^2C^2/8) \end{aligned} \tag{3.20}$$

So we can write the below inequality for all experts

$$\begin{aligned} &U_{i,t} = q_{i,0} \exp(-hS_{i,t}) \leq \exp(-hE[S_{i,t'}] + th^2C^2/8) \\ &hE[S_{i,t'}] \leq hS_{i,t} - \log(q_{i,0}) + th^2C^2/8 \\ &E[S_{i,t'}] \leq S_{i,t} - \frac{\log(q_{i,0})}{h} + thC^2/8. \end{aligned} \tag{3.21}$$

To conclude, we need to show that the expected accumulated NP loss difference between the ensemble model and its best partition (i.e best expert) as

$$\begin{aligned} &\frac{1}{t}\Big(E[S_{i,t'}] - S_{i,t}\Big) \leq \frac{1}{8}hC^2 - \frac{1}{ht}\log q_{i,0} \text{ yielding} \\ &\frac{1}{8}hC^2 - \frac{1}{ht}\log q_{i,0} = \frac{1}{\sqrt{t}}\left(\frac{C^2}{8} - \log q_{i,0}\right) \to 0 \text{ as } t \to \infty \text{ with } h = \frac{1}{\sqrt{t}}. \end{aligned} \tag{3.22}$$

These derivations follow similar lines of the derivations in [103], but also deviate in that we adapted here to the Neyman-Pearson formulation by specifically considering the NP performance. To be more precise, the loss structure in [103] and our NP loss here are completely different. Above derivation provides direct implementation of our proposed online randomized NP classification g_t given in (3.10), which is asymptotically guaranteed to outperform (or perform at least as well as) the best expert on average in terms of the NP-loss, and is tuned to the best observation space partition.

In this sense, the introduced algorithm weights over the ensemble dynamically optimally to operate with the right modeling power and mitigate overfitting. Here, we set for the constant $C \geq \max(\hat{\mu}_{\tau,t-1}, 1)\ \forall i,t \Rightarrow C \geq |\mu_{\tau,t-1}|\ \forall i,t \Rightarrow C \geq |\mu_{\tau,t-1} l(-y_t f_{i,t-1}(x_t))|\ \forall i,t \Leftrightarrow C \geq |s_{i,t}|\ \forall i,t$, where $\hat{\mu}_{\tau,t-1} \to \hat{\mu}_\tau \to \infty$ as $(\tau,t) \to (0,\infty)$ and $\hat{\mu}_{\tau,t-1} \to \hat{\mu}_\tau = 0$ as $(\tau,t) \to (1,\infty)$ with an appropriately chosen β_μ. Inspecting (3.22), outperforming (or achieving the performance of) the best expert takes longer time as τ gets smaller. On the other hand, $\hat{\mu}_{\tau,t}$ takes updates only when $y_t = -1$ (as shown in (3.9)) which essentially multiplies the required number

of i.i.d. samples from the stream with $\frac{1}{\pi^-}$ for convergence to the desired false alarm rate τ, for both our algorithm as well as the ensemble experts. Thus, as another important contribution of the presented study, here we also characterize that the NP problem becomes harder as $(\tau, \pi^-) \to (0,0)$. As a final remark, although piecewise OLNPs are considered only so far, our results can be straightforwardly extended to piecewise NP-NNs (whose power can be controlled with the dimension multiplier M) as well, if one wishes to limit the ensemble size without reducing the modeling power.

3.3.3 Computational Scalability: Online Processing and Binary Partitioning Tree

The data modeling power (nonlinear learning capability) of our algorithm can be improved arbitrarily to any desired degree by increasing the size as well as the variety of the employed expert ensemble. As this size increases, the best expert improves, so does the proposed algorithm as guaranteed in a competitive manner, cf. (3.22). On the contrary, a naive implementation of the proposed algorithm requires to run N_q experts in parallel to pick a decision randomly at each time. The resulting computational complexity $O(N_q)$ that is linear with respect to the ensemble size is impractical if one uses a necessarily large ensemble in an attempt to improve the modeling, which potentially hinders the real-time processing. However, the complexity can be immensely reduced to $O(\log\log N_q)$ with a versatile implementation, if the experts are designed carefully and represented hierarchically. For this, we use the general information processing framework of the binary partitioning context tree that has been previously successfully applied for various purposes such as density estimation in [105] and classification (this is not Neyman-Pearson, unlike the presented study) in [103], dating back to coding in [97]. A depth-D binary tree is constructed on which a region from the observation space is kept and split into two, and the resulting two sub-regions are assigned to the corresponding child nodes. This process is applied to all the nodes recursively starting from the root node until the leaves are reached at the depth D, with the root node keeping the complete observation space. As an example, in the case of a one dimensional observation space $[0,4]$ being partitioned by a depth-2 tree, the starred observation $x_t = 2.1$ in Fig. 3.2 visits the root node $[0,4]$ and its right $(2,4]$ and the right-left $(2,3)$ child nodes, respectively. The set of the prunings of this tree here provides an ensemble of partitions, $\mathcal{P}_1 = \{[0,4]\}$ (single piece), $\mathcal{P}_2 = \{[0,2],(2,4]\}$ (two pieces), $\mathcal{P}_3 = \{[0,2],(2,3],(3,4]\}$ (three pieces), $\mathcal{P}_4 = \{[0,1],(1,2],(2,4]\}$ (three pieces), $\mathcal{P}_5 =$

$\{[0,1],(1,2],(2,3],(3,4]\}$ (four pieces), where an expert $f_{\mathcal{P}_i,t}$ is obtained when a separate OLNP is trained for each region of the partition $\mathcal{P}_i$, e.g., $f_{\mathcal{P}_3,t-1}(2.1) = f_{(2,3],t-1}(2.1)$, and the ensemble size is $N_q = 5$. In general, $f_{\mathcal{P}_i,t-1}(x_t) = f_{R_{i,j},t-1}(x_t)$ if $x_t \in R_{i,j}$, $\mathcal{P}_i = \{R_{i,1}, R_{i,2}, \cdots, R_{i,\eta_i}\}$ and $f_{R_{i,j},t-1}$ is the OLNP trained with only those observations falling in $R_{i,j} \in \mathcal{P}_i$. As for the tree in this example, the node regions are given by $O \equiv [0,4], O_l \equiv [0,2], O_r \equiv (2,4], O_{l,l} \equiv [0,1], O_{l,r} \equiv (1,2], O_{r,l} \equiv (2,3]$ and $O_{r,r} \equiv (3,4]$, where O is the root node and O_l (O_r) are the left (right) child of the root node, and similarly for the others. Hence, at each node ν, there is a region and an OLNP $f_{\nu,t-1}$ producing the corresponding node decision. Let us use for simplicity $f_{i,t-1} \triangleq f_{\mathcal{P}_i,t-1}$.

Observing that $\{f_{i,t-1}(2.1)\}_{i=1}^{N_q=5} = \{f_{O,t-1}(2.1), f_{O_r,t-1}(2.1), f_{O_{rl},t-1}(2.1)\}$, obtaining the node decisions only is sufficient to obtain the final decision of our algorithm $g_{t-1}(\boldsymbol{x}_t)$ for any observation $\boldsymbol{x}_t$ in general for a depth-D tree visiting the nodes $\{\nu_0 = O, \nu_1, \cdots, \nu_D\}$ as

$$\Leftrightarrow g_{t-1}(\boldsymbol{x}_t) = 2 \times \text{sgn}(f_{\nu_i,t-1}(\boldsymbol{x}_t)) - 1 \text{ with probability } \tilde{q}_{i,t-1} \text{ for } 0 \le i \le D, \tag{3.23}$$

where the expert probabilities q are accumulated in

$$\tilde{q} \text{ by } \tilde{q}_{i,t-1} = \sum_{j=1}^{N_q} q_{j,t-1} 1_{\{f_{j,t-1}(\boldsymbol{x}_t) = f_{\nu_i,t-1}(\boldsymbol{x}_t)\}}, 0 \le i \le D.$$

Here, $1_{\{\cdot\}}$ is the indicator function returning 1 if its argument holds, and 0 otherwise. At this point of the chapter, we would like to further explain $O(D)$ computation of the $\tilde{q}_{i,t}$ randomization probabilities.

Let us introduce the combined performance variable $\mathcal{U}_{\nu,t}$ that is defined for any node over the tree but updated in time for only those nodes $\nu \in \{\nu_0 = O, \nu_2, \cdots, \nu_D = leaf\}$ the observation $\boldsymbol{x}_t$ visits as

$$\begin{aligned}
&\mathcal{U}_{\nu,t} \triangleq \exp(-hS_{\nu,t}) = \exp(-hS_{\nu,t-1}) \times \exp(-hs_{\nu,t}), \text{ if } \nu \text{ is a leaf node,} \\
&\mathcal{U}_{\nu,t} \triangleq q_s \mathcal{U}_{\nu_l,t} \mathcal{U}_{\nu_r,t} + (1 - q_s) \exp(-hS_{\nu,t}), \text{ if } \nu \text{ is not a leaf node,}
\end{aligned}$$

where $s_{\nu,t}$ and $S_{\nu,t}$ are the stochastic and the accumulated NP losses of the OLNP running at the node ν and $0 \le q_s \le 1$ is the node split probability of the tree that is related to the expert probabilities

$$q_{i,t} = \frac{U_{i,t-1}}{\sum_{j=1}^{N_q} U_{j,t-1}} \text{ with } 1 - q_s$$

being the probability of choosing the expert of the complete observation space (single piece) at time $t = 1$, as further explained below. Importantly, at the root node $O = \nu_1$, we have

$$\mathcal{U}_{O,t} = \sum_{i=1}^{N_q} U_{i,t}$$

(so the name "combined performance"), and note that these two variables $S_{\nu,t}$,$\mathcal{U}_{\nu,t}$ can be updated bottom up from the leaf ν_D to the root ν_0 at each time for the observation x_t with complexity in the order $O(D)$.

Now, let us consider the node choosing probability $\tilde{q}_{i,t}$, which is verbally the sum of the probabilities of the experts (consider the corresponding partitions over the tree) containing the node ν_i as the leaf in its corresponding pruned tree. For the root node, it is simply

$$\tilde{q}_{0,t} = \frac{(1-q_s)\exp(-hS_{O,t-1})}{\mathcal{U}_{O,t}}$$

and for the other nodes (cf. [103, 97]),

$$\tilde{q}_{i,t} = \frac{q_s}{1-q_s}\tilde{q}_{i-1,t}\mathcal{U}_{\nu_{i,s},t}\mathcal{U}_{\nu_i,t}\left((1-q_s)1_{\{\nu_i=\nu_D\}} + 1_{\{\nu_i\neq\nu_D\}}\right),$$

where $\nu_{i,s}$ is the sibling node of ν_i. As a result, the computation of the lumped probabilities $\tilde{q}_{i,t}$'s can be completed at each time for the observation $\boldsymbol{x}_t$ in two passes of complexity $O(D)$: (1) first, bottom up computation from the leaf ν_D to the root ν_0 for the variables $S_{\nu,t}$,$\mathcal{U}_{\nu,t}$ as described above, (2) and then, top down computation from the root ν_0 to the leaf ν_D to finally get $\tilde{q}_{i,t}$'s along the way. We point out that split probability q_s draws the initial expert probabilities $q_{j,1}$'s as the following. Considering back our depth-2 example, $\mathcal{P}_1 = \{[0,4]\} \to q_{1,1} = 1-q_s$ (single piece), $\mathcal{P}_2 = \{[0,2],(2,4]\} \to q_{2,1} = q_s(1-q_s)^2$ (two pieces), $\mathcal{P}_3 = \{[0,2],(2,3],(3,4]\} \to q_{3,1} = q_s^2(1-q_s)$ (three pieces), $\mathcal{P}_4 = \{[0,1],(1,2],(2,4]\} \to q_{4,1} = q_s^2(1-q_s)$ (three pieces), $\mathcal{P}_5 = \{[0,1],(1,2],(2,3],(3,4]\} \to q_{4,1} = q_s^3$ (four pieces), where the experts with more pieces obtain smaller probabilities, intuitively and as desired. Typically $q_s \leq \frac{1}{2}$. This finalizes the description of the versatile implementation of our proposed randomized algorithm g_t for which a pseudo-code table is also given below.

To conclude, the binary partitioning tree-based implementation of our algorithm (described in this section and Section 3.3.2) has computational complexity $O(D)$ that is linear in the tree depth. Comparing with the complexity $O(N_q)$ of the naive implementation, we achieve immense computational gains in twice logarithmic manner by reducing from $O(N_q)$ to $O(D)$ since the number of prunings of a binary tree is twice exponential in the depth, i.e., $N_q \simeq (1.5)^{2^D}$ [97]. We call this efficient

Algorithm Proposed Tree NP Classifier (Tree OLNP)

1: Set the tree depth D, target false alarm (positive) rate τ, regularization λ, and the parameters β_f, β_μ as well as the constants h.
2: Construct the tree and initialize.
3: **for** $t = 1, 2, \ldots$ **do**
4: Receive the observation x_t and find the visited nodes ν_i's.
5: Calculate the lumped probabilities (top-down) $\hat{q}_{i,t-1}$'s.
6: Calculate the decision using $g_{t-1}(x_t)$ and observe the feedback y_t.
7: Update μ_τ and obtain $\mu_{\tau,t}$.
8: Calculate the instantaneous (top-down) NP loss $s_{\nu,t}$ for each visited node.
9: Obtain the update node classifiers (top-down) $f_{\nu,t}$'s.
10: Obtain the updated combined performance $\mathcal{U}_{\nu,t}$'s for each visited node (bottom-up).
11: **end for**

implementation Tree OLNP.

Two main aspects of the computational efficiency of our algorithm is first the online processing with linear complexity in the number of the processed instances ($O(t)$), as already achieved in Section 3.3.2. The second aspect is achieving complexity that is linear in the tree depth and twice logarithmic in the number of the experts in the ensemble ($O(D) \sim O(\log \log N_q)$, overall: $O(Dt)$), as achieved in this section -however- by introducing the lumped probabilities $\hat{q}_{i,t}$'s which must also be obtained sequentially in the $O(D)$ (linear in depth) manner. In this section, we also provided $O(D)$ computation of the lumped probabilities and a table of our algorithm as a pseudo code. We stress that our main contribution in this section is to incorporate the Neyman-Pearson formulation of classification with false alarm rate controllability into the general information processing framework of the context trees [105, 103, 97].

Our algorithm Tree OLNP with only a root node is a regular OLNP running for the complete observation space, collecting and organizing OLNPs in a piecewise manner. Tree OLNP models nonlinear separations more powerfully as the tree depth increases whereas a regular OLNP is not capable of learning nonlinearity. A limitation for Tree OLNP emerges as the observation ($\boldsymbol{x} \in \mathbb{R}^d$) dimensionality d grows since finding the right way of splittings (partitionings) at the nodes is challenging. This is not an issue when d is small, e.g., $d = 1, 2$ or $d = 3$, and one can simply use even splits (mid-point between min/max values) feature by feature at each depth; but this would require an extremely large tree depth in high dimension. As a solution for high dimension, we use PCA transformation at the nodes for Tree OLNP and then evenly split after projecting to the vector of the largest variance.

To further observe advantages of Tree OLNP over OLNP and NP-NN, we also investigate computational requirements of the individual algorithms (Tree OLNP,

OLNP and NP-NN) in tuning and regular running phases. It is important to mention that since all of these algorithms are online, there is no need to make separation between running and tuning phases but, since NP-NN needs very extensive parameter tuning, we are opt to make this distinction while discussing computational advantages of Tree OLNP. Therefore, we provide detailed computational complexity comparisons among these algorithms in terms of the required number of dot products in Table 3.1 along with the actual realizations in Table 3.2 for various benchmark datasets. Generally, single OLNP has the fastest processing and Tree OLNP is the second fastest. This is for the regular phase of running the algorithms. All these algorithms are online (OLNP, NP-NN and Tree OLNP) and able to process data in real time sequentially without a separate training. However, the single NP-NN (unlike OLNP and Tree OLNP) additionally requires a separate extensive cross validations for M as well as the bandwidth parameter Ω of the RBF kernel for every τ. Namely, our algorithm dynamically determines the right modeling power on the fly (with regards to the amount of available data at each time) during the data stream without having to know the horizon (the total number of instances to be processed), whereas NP-NN achieves the required degree of modeling complexity via extensive cross validations (before the processing starts, and it is not dynamic as it is fixed for all times in the stream once it is set) which not only hinders a fluent online processing but also requires the knowledge of the horizon. Finally, OLNP and NP-NN neither have a dynamically optimal control over the modeling power nor provides competitive performance guarantees; on the contrary, Tree OLNP has those valuable properties. This is reflected in our performance results which we present in Section 3.4.

3.4 Performance Evaluations and Experimental Results

We present performance evaluations of Tree OLNP as well as comparisons with OLNP [22] and NP-NN [96], based on 8 different real and synthetic datasets [83, 82] from various fields (with properties given in Table 3.3) after applying z-score (zero mean unit variance) normalization. For each dataset, class with the largest size is assigned to label -1, and the remaining are considered as label 1. We split each dataset to training (70%), validation (15%) and test (15%). Validation set is used for parameter optimization via cross-validation and algorithms are trained in an online manner on the training set while probing the performance at logarithmically spaced 100 time points. It is important to note that cross validation is only applied to NP-NN whereas OLNP and Tree OLNP do not require extensive parameter tuning. We run each experiment for every target false alarm rate ($\{0.005, 0.01, 0.05, 0.1, 0.2\}$) for 32 times and calculate corresponding NP-scores as NP-score $= \kappa \max(\hat{P}_{\text{fa}}(f) - \tau, 0) + \hat{P}_{\text{nd}}(f)$ of [15] which measures the maximization of realized TPR together with the trackability of the target FPR (TFPR), i.e., TFPR shall match realized FPR. Here, lower NP-score is better. We opted for the default choice $\kappa = 1/\tau$ in our evaluations. To improve the convergence, we augment the training set by concatenating random shuffles of the same set to achieve a sample size of 150×10^3. This augmentation is only for the training set and does not apply to fixed validation or test sets. In addition, we select 15 runs out of 32 with lowest NP-scores in order to discard cases that have poor FPR convergence. We emphasize that this filtering is applied to all datasets. Transient graphs given in Fig. 3.6 are mean values collected from training of selected 15 runs. Similarly, results represented in Table 3.3 are calculated on test sets (more specifically, we use the latest data point in logarithmically spaced performance evaluation sequence) on the same run population.

The classifier parameters $\boldsymbol{w} = [\tilde{\boldsymbol{w}}, b]^T$ (linear projection and the bias for all OLNP, Tree OLNP and NP-NN) are both initialized around 0. For NP-NN, the RBF kernel bandwidth Ω and the projection size M are both cross-validated with $\Omega \in \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}/d$ and $M \in \{2, 5, 10, 20, 40, 80, 100\}$, where d is the observation dimension. We initially set $\hat{\mu}_\tau = 1$. The step sizes β_f and β_μ are cross-validated with $\beta_f \in \{0.1, 0.01\}$ and $\beta_\mu \in \{2.5 \times 10^2, 5 \times 10^2, 1 \times 10^3, 5 \times 10^3, 1 \times 10^4, 1 \times 10^5\}/n^-$. Finally, the split probability parameter is set as $q_s = \frac{1}{2}$. We select depth for Tree OLNP as 8 for the results available in Tab. 3.3 and Fig. 3.7. Effect of different depths on Tree OLNP performance is shown in Fig. 3.6 and discussed below. Performance metrics are (1) visual inspection of the receiver operating characteristics curve (ROC): realized true positive rate (TPR) vs realized

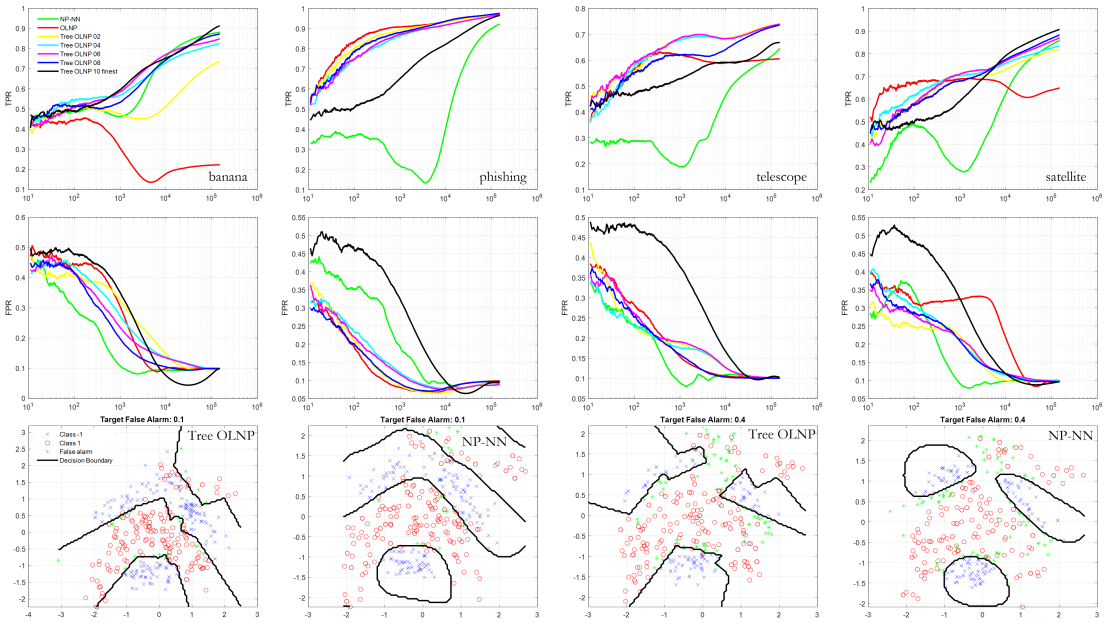


Figure 3.6 The transient behavior (learning TPR and FPR) of the algorithms are shown for TFPR $\tau = 0.1$ with varying depths on the Banana, Phishing, Telescope and Satellite datasets. We also present decision boundaries calculated by NP-NN and Tree OLNP with depth of 6 on visually representative Banana dataset.

false positive (alarm) rate (FPR), (2) area under the ROC curve (AUC), and (3) the NP-score.

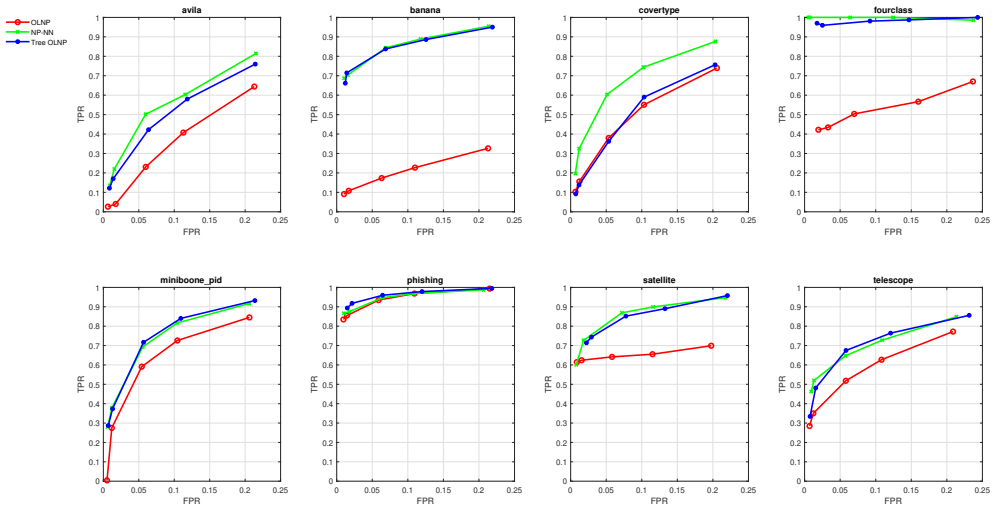


Figure 3.7 Receiver operator characteristics (ROC: realized FPR: realized TPR) curves are shown for all the datasets.

Fig. 3.6 reports the transient behavior (realized TPR vs realized FPR) of OLNP, NP-NN and Tree OLNP with different depths on Banana, Phishing, Telescope and Satellite datasets for TFPR $\tau = 0.1$. Each transient graph is calculated as the mean

of 15 (we select 15 runs out of 32 using NP-scores as explained above) different runs. We also include finest partition of Tree OLNP with depth of 10 (Tree OLNP 10) to the comparison. Finest partition is calculated by setting the split probability of Tree OLNP as $q_s = 1$ and eventually, forcing predictions to be calculated only with the leaf nodes. For all datasets, all models successfully converge to target false alarm rate. In Phishing, Telescope and Satellite datasets, finest partition under performs in terms of TPR compared to other Tree OLNP models. Our proposed algorithm Tree OLNP, mitigates overfitting by favoring the less complicated model when total number of data processed by the ensemble model is low. As more samples are observed by the ensemble model, weights of complex experts increase. Similar to the finest partition of Tree OLNP 10, NP-NN is also a complex algorithm that requires extensive parameter tuning. As seen from Fig. 3.6 NP-NN does not perform as well as Tree OLNP models in terms of TPR when total number of samples processed by the model is low. We also present decision boundaries for Tree OLNP and NP-NN using 2D banana dataset for $\tau \in \{0.1, 0.4\}$. It is important to note that in order to better visualize decision boundaries for Tree OLNP, instead of selecting predictions generated by experts with different probabilities which are correlated with corresponding expert's performance, we always use the prediction generated by the expert with highest probability. This stabilizes decision boundaries near high entropy areas. As shown in Fig. 3.6, Tree OLNP tends to learn nonlinear decision boundaries in piece-wise manner.

Table 3.1 Computational complexity comparisons among among OLNP, Tree OLNP, NP-NN and Tree NP-NN OLNP and Tree NP-NN) algorithms. T is the total number of processed observations and M is typically far larger than the tree depth D, i.e., $M >> D$. Typically, $2 \leq M \leq 100$ depending on the cross validation whereas $2 \leq D \leq 6$.

	Partition	Transformation	Classification	Dot Products Per Sample	Total Dot Products	Complexity
OLNP	0	0	1	1	T	$O(T)$
Tree OLNP	$D-1$	0	D	$2D-1$	$2TD-T$	$O(TD)$
NP-NN	0	Md	$2M$	$Md+2M$	$TMd+2TM$	$O(TMd)$

Considering the detailed analysis given in Table 3.3 and corresponding ROC curves presented in Fig. 3.7, we observe that NP-NN and Tree OLNP, outperforms OLNP in 7 out of 8 datasets. Phishing, is more linear compared to other datasets therefore OLNP, which is linear by its definition, performs as well as Tree OLNP and slightly outperforms NP-NN in terms of AUC as seen in Table 3.3. For Avila and Covertype datasets, NP-NN performs better in terms of AUC compared to Tree OLNP. To be more specific, considering the data presented in Table 3.3, Tree OLNP outperforms NP-NN on 4 out of 8 datasets in terms of AUC. In addition, Tree OLNP performs as well as NP-NN on 6 out of 8 datasets. It is important to emphasize that NP-NN requires extensive hyperparameter tuning in order to perform well on datasets and

because of the usage of RBF kernel, it is prone to overfitting. On the other hand Tree OLNP controls model complexity with the increasing sample size therefore it is less prone to overfitting.

Table 3.2 Actual computations realized corresponding to Table 3.1 in Section 3.3.3: Number of dot products for various benchmark datasets with $\tau \in \{0.005, 0.01, 0.05, 0.1, 0.2\}$ for 5 experts. In NP-NN, due to the required cross validation for Ω and M, we have different number of dot products whereas this number is the same for all τ in Tree OLNP or Tree NP-NN.

Dataset	OLNP	Tree OLNP	NP-NN				
(n^-, n^+, d)	all τ	all τ	0.005	0.01	0.05	0.1	0.2
avila (12295, 8572, 10)	20867	104335	500808	1252020	20032320	25040400	2504040
banana (2924, 2376, 2)	5300	26500	2120000	106000	212000	848000	212000
covertype (297711, 283301, 54)	581012	3486069	650733440	65073344	3253667200	1301466880	325366720
fourclass (555, 307, 2)	862	4310	6896	17240	6896	6896	6896
miniboone_pid (93565, 36499, 50)	130064	650320	135266560	135266560	676332800	33816640	33816640
phishing (6157, 4898, 68)	11055	55275	77385000	61908000	77385000	7738500	77385000
satellite (4399, 2036, 36)	6435	32175	489060	2445300	19562400	2445300	489060
telescope (12332, 6688, 10)	19020	95100	456480	18259200	9129600	2282400	456480

3.5 Final Remarks

In this chapter, we proposed a framework for online NP classifiers which maximizes detection power (minimizes Type II error) under a given false alarm (Type I error) constraint. Ensemble classifier framework, i.e context tree framework, divides the features space using iterative projection to principle components and aims to solve main problem in a piecewise manner. At each region it has OLNP (in this thesis we select OLNP as base classifier but any online classifier can be used). It is possible to represent feature space as different combinations of regions. We call each different combination a partition and every partition represents a piecewise NP classifier (expert) which is used by Tree OLNP. Thanks to linear training time of the context tree framework, proposed ensemble classifier allows us to use simpler models with less parameters to achieve similar performance without any fine tuning effort compared to random Fourier feature based NP-NN (detailed description

Table 3.3 We present performance evaluation of the proposed algorithm Tree OLNP and the state-of-the-art algorithms OLNP and NP-NN. The evaluation metrics are calculated for each target false positive rate (TFPR $\in \{0.005, 0.01, 0.05, 0.1, 0.2\}$) on 8 different datasets. First columns shows the number of features for each dataset with the number of positive (target) and negative (non-target) samples. For each target false alarm, we run algorithms 32 times with shuffling over different permutations of training and test set pairs. Out of 32 runs, we use 15 runs with the lowest NP-scores (as explained in Section 3.4) to calculate the mean and standard deviations of the results as reported in the table. For each shuffle, training, validation and test sets contain 70%, 15% and 15% of the actual set, respectively. For every iteration (15 iterations), we calculate true positive rate (TPR) and NP-score. Therefore we present mean of the calculated scores with their standard deviations. For each dataset, first row is the achieved TPR, second row is the achieved NP-score and the third row is the area under curve (AUC) of the receiver operating characteristics (ROC) curve of TPR vs TFPR. We emphasize that AUC of ROC curves are calculated up to 0.2 target false alarm rate and do not include operating points with false positive rate higher than 0.2.

Dataset	OLNP			NP-NN			Tree OLNP		
(n_-, n_+, d)	TFPR(τ)$\rightarrow$ 0.01	0.05	0.1	0.01	0.05	0.1	0.01	0.05	0.1
avila	TPR$\rightarrow$ 0.040±0.005	0.231±0.026	0.408±0.020	0.220±0.040	0.502±0.022	0.603±0.023	0.170±0.014	0.422±0.016	0.580±0.018
(12295, 8572, 10)	NP-score$\rightarrow$ 1.696±0.213	0.965±0.082	0.721±0.069	1.328±0.831	0.691±0.099	0.551±0.047	1.221±0.103	0.852±0.120	0.604±0.076
	AUC(until 0.2 TFPR)$\rightarrow$ 0.076±0.007			0.119±0.008			0.107±0.009		
banana	0.108±0.015	0.173±0.016	0.227±0.019	0.703±0.080	0.844±0.020	0.889±0.016	0.715±0.029	0.838±0.018	0.886±0.015
(2924, 2376, 2)	1.518±0.254	1.090±0.169	0.875±0.084	0.829±0.324	0.534±0.174	0.291±0.081	0.697±0.235	0.536±0.201	0.369±0.090
	0.045±0.005			0.176±0.011			0.179±0.012		
covertype	0.156±0.010	0.379±0.013	0.550±0.006	0.325±0.022	0.603±0.009	0.744±0.006	0.138±0.016	0.363±0.015	0.590±0.009
(297711, 283301, 54)	1.067±0.101	0.689±0.034	0.481±0.017	0.862±0.159	0.421±0.028	0.282±0.009	1.041±0.123	0.709±0.046	0.442±0.020
	0.101±0.004			0.136±0.003			0.102±0.002		
fourclass	0.434±0.073	0.504±0.078	0.567±0.060	1.000±0.000	1.000±0.000	1.000±0.000	0.959±0.033	0.981±0.021	0.987±0.021
(555, 307, 2)	2.860±1.009	0.892±0.353	1.033±0.245	0.182±0.362	0.320±0.415	0.246±0.261	1.552±1.164	0.857±0.317	0.479±0.143
	0.118±0.022			0.231±0.033			0.222±0.028		
miniboone_pid	0.275±0.020	0.591±0.013	0.726±0.012	0.377±0.011	0.693±0.012	0.817±0.007	0.373±0.026	0.717±0.013	0.840±0.010
(93565, 36499, 50)	0.915±0.089	0.493±0.045	0.319±0.039	0.811±0.094	0.418±0.043	0.225±0.028	0.928±0.118	0.416±0.059	0.252±0.061
	0.132±0.007			0.150±0.005			0.159±0.011		
phishing	0.855±0.010	0.934±0.008	0.968±0.007	0.876±0.013	0.947±0.009	0.968±0.006	0.918±0.011	0.959±0.006	0.979±0.006
(6157, 4898, 68)	0.597±0.176	0.243±0.095	0.125±0.069	0.956±0.371	0.343±0.107	0.121±0.094	1.241±0.291	0.331±0.113	0.222±0.042
	0.196±0.009			0.187±0.007			0.197±0.011		
satellite	0.624±0.027	0.642±0.040	0.655±0.036	0.727±0.036	0.869±0.019	0.899±0.021	0.744±0.022	0.852±0.020	0.890±0.020
(4399, 2036, 36)	0.919±0.378	0.524±0.141	0.495±0.099	1.098±0.407	0.572±0.181	0.259±0.063	2.176±0.308	0.701±0.147	0.437±0.075
	0.124±0.007			0.183±0.017			0.173±0.013		
telescope	0.350±0.018	0.518±0.020	0.627±0.017	0.520±0.049	0.647±0.021	0.728±0.021	0.481±0.032	0.675±0.017	0.764±0.018
(12332, 6688, 10)	0.877±0.179	0.641±0.157	0.456±0.054	0.817±0.301	0.503±0.082	0.364±0.066	1.079±0.361	0.490±0.118	0.445±0.157
	0.121±0.004			0.146±0.012			0.163±0.016		

is available in Chapter 2). Proposed ensemble classifier dynamically controls the complexity of the model by predicting the output as a combination of experts with different complexities. In the beginning of the process, where the total number of processed samples is low, ensemble model favors simpler models. As time passes and more samples are processed by Tree OLNP, it dynamically changes experts weights effectively optimizing its complexity. This helps Tree OLNP to mitigate overfitting and to find the solution with optimal complexity.

4. Active Learning for Online Nonlinear Neyman-Pearson Classification

The main goal of active learning is to find "useful" (impactful in model training) samples in order to improve the convergence of the model to the desired operating point (desired TPR, FPR) [107]. This also resonates with the case, where we have large amount of unlabelled data, we need to train a model using a subset of the given unlabelled data and annotation is costly [108]. For example, while developing a fraud detection model for credit card transactions, high percentage of the collected data will be unlabelled for fraudulent activities. These activities should be analyzed and flagged by business experts which ends up being a costly process. Therefore, important selection of unlabelled samples to achieve high performance after training is necessary [109]. Similar to the aforementioned application (fraudulent activity detection), active learning is used in many various fields including speech recognition [110], text categorization [111, 112], information extraction [113], remote sensing [114, 115, 116], image classification [117, 118] and deep learning [119, 120, 121].

Two different active learning scenarios are studied in the literature which are myopic and batch mode active learning [108]. In myopic learning, our model queries single data at a time while in batch mode learning, multiple samples are queried at once. In this chapter, since the proposed algorithms are online (NP-NN, Tree OLNP) and expecting input as stream of data (one sample at a time), we study myopic active learning methods, where we have pool of training samples and the learning algorithm makes the decision of processing or discarding the sample upon determined active learning criterion [108]. On the other hand, in the pool-based (batch mode) active learning, samples within the batch are selected simultaneously [122], which is out of scope for this chapter's focus. In stream based (myopic) active learning, it is crucial to select the most informative sample based on a criterion in order to optimize performance convergence of the model. There are several different criterions are introduced in the literature such as query-by-committee [123], uncertainity sampling [112], expected error minimization [124, 125, 126], variance reduction [127, 128, 129], variance maximization [130], maximum model change [131, 132, 133, 134] and the

min-max view [135, 136].

The algorithms proposed in this thesis are trained under Neyman-Pearson (NP) framework, where the goal is to achieve a constant controllable user-defined false alarm rate with maximum possible detection power [17]. The first proposed NP classifier in this thesis (explained in Chapter 2) is NP-NN [77, 96], which uses random Fourier features [53] to learn nonlinear decision boundaries while maintaining desired false alarm rate thanks to NP framework. As described in Chapter 3, there are two important disadvantages of NP-NN. First one is the necessity of hyper-parameter tuning. Apart from the default parameters of the perceptron, random Fourier features related parameters are critical in performance optimization of the model. In case of using radial basis function (rbf) as the kernel, the optimum number of dimensions of the kernel space and the width of rbf bandwidth should be selected via extensive hyper-parameter tuning. Detailed description of the hyper-parameter space and corresponding tuning strategy is given in Chapter 2. Second disadvantage of NP-NN is the usage of radial basis function, which has infinite VC dimension [42]. Therefore, RBF effectively makes this approach susceptible to overfitting. In Chapter 3, we proposed context tree based ensemble model which dynamically maps its complexity. Namely, as the total number of samples processed by the model is low, Tree OLNP increases the weight of its simple models. As number of data increases, it favors towards complex models by increasing the weights of the corresponding expert. This mitigates overfitting problem by introducing dynamic control over overall model complexity.

The main contribution of this chapter is introducing active learning for our ensemble NP classifier Tree OLNP to have better convergence to the desired operating point. It is important to note that, since Tree OLNP is an NP classifier, we need to make sure that NP constraints are satisfied (maximizing detection power while upper bounding the false alarm rate by a user defined threshold). In other words, we need to make sure that the target false alarm is satisfied while the convergence of the model is improved.

We provide the problem formulation in Section 4.1 and provide detailed implementation of active learning of our online algorithm Tree OLNP (detailed description available in Chapter 3) in Section 4.2. We compare the performance of proposed active learning method with random sampling (model processes every sample) on real and synthetic datasets in terms of TPR and FPR convergence in Section 4.3. The chapter ends with final discussions and conclusion in Section 4.5.

4.1 Problem Description

We study online binary classification under Neyman-Pearson framework, where proposed model observe input data $\boldsymbol{x}_t \in \mathbb{R}^d$ and decide upon its label $\hat{y}_t \in \{-1, +1\}$ in an online manner. In Chapter 3, we introduced context tree based ensemble Tree OLNP classifier g_t, which is a randomized mixture of N_q piecewise linear NP classifiers f_i (experts). For input $\boldsymbol{x}_t$, prediction of Tree OLNP at time t can be calculated as

$$g_t(\boldsymbol{x}) = 2 \times \text{sgn}(f_{i,t}(\boldsymbol{x})) - 1 \text{ with probability } q_{i,t} \text{ for } 1 \leq i \leq N_q, \tag{4.1}$$

where $f_{i,t}$ is the i^{th} expert and $q_{i,t}$ is the corresponding probability of the expert f_i at time t. This probability is scaled by the cumulative NP performance of the corresponding expert as explained in Chapter 3. In the same chapter we also show mathematical proof that the performance of the ensemble of experts is as good as the performance of the best expert. When total number of processed data is low, ensemble classifier favors simple experts (piecewise OLNPs in our case and simplest expert is a single OLNP itself) and gradually increases the probabilities of complex models to mitigate overfitting. This dynamic behaviour is explained in more detail in Chapter 3. For any $\boldsymbol{x}_t$, we calculate $\hat{y}_{i,t} = f_{i,t}(\boldsymbol{x}_t)$ for $1 \leq i \leq N_q$, where $\hat{y}_{i,t}$ is the prediction of i^{th} expert at time t. Since Tree OLNP is randomized mixture of experts, final decision $\hat{y}_t$ is calculated by selecting one of the expert predictions $\hat{y}_{i,t}$ with probability $q_{i,t}$.

4.2 Active Learning in Neyman Pearson Framework

Experts of the proposed ensemble Tree OLNP model are also binary classifiers whose weights (probabilities) gradually change with respect to their corresponding NP performances and each expert makes prediction as $\hat{y}_{i,t} \in \{-1, +1\}$. At this point we can make the observation that, for an input sample $\boldsymbol{x_t}$, collection of experts can be uncertain about the predicted label. More specifically, in the worst case scenario (highest uncertainity), half of the experts can generate prediction of 1 (target) for input sample $\boldsymbol{x}_t$, while the remaining experts can predict the same input sample as -1 (non-target). In the exact opposite case, namely when there is

no uncertainity between the prediction of experts, all of the experts (piecewise linear OLNPs) generate the same label for the same input. This observation is very critical in introducing active learning to our ensemble model. In this chapter, we propose to use uncertainity between the expert predictions as a measure of "usefullness" of the input sample and decide upon whether including or excluding the target sample from training. This will effectively decrease the total number of necessary samples to achieve the target performance. In the field of active learning, this method is introduced as uncertainity sampling (entropy based sampling) [112]. For Tree OLNP, we need to measure level of uncertainity between the experts. Using probability of experts ($q_{i,t}$) and their corresponding predictions $\hat{y}_{i,t}$, it is possible to calculate level of uncertainity (entropy) as

$$
\begin{aligned}
\mathrm{P}(\hat{y}_t = -1) &= \sum_{i=1}^{N_q} q_{i,t} 1_{\{f_{i,t}(\boldsymbol{x}_t) \leq 0\}} \\
\mathrm{P}(\hat{y}_t = 1) &= \sum_{i=1}^{N_q} q_{i,t} 1_{\{f_{i,t}(\boldsymbol{x}_t) > 0\}} \\
\sigma_t &= -\Big(\mathrm{P}(\hat{y}_t = -1)\log(\mathrm{P}(\hat{y}_t = -1)) \\
&\quad + \mathrm{P}(\hat{y}_t = 1)\log(\mathrm{P}(\hat{y}_t = 1))\Big),
\end{aligned}
\tag{4.2}
$$

where N_q is the total number of experts, namely piecewise OLNPs. At this point of the chapter, we would like to remind that, even though number of experts scales doubly exponentially with the increasing tree depth ($N_q \simeq (1.5)^{2^D}$), thanks to efficient computation explained in Chapter 3, it is possible to decrease the complexity of this computation to $O(n)$. Note that, expert probability $q_{i,t}$ is only calculated for experts that contains the regions that input $\boldsymbol{x}_t$ visits. Remaining probabilities are not calculated. Reader can refer to Chapter 3 for more detailed explanation about regions and their corresponding partitions. Using Equation 4.3, we measure the amount of agreement among the experts with the entropy function of the Bernoulli random variable:

$$
[0,1] \ni H(\theta) = -\theta \log_2 \theta - (1-\theta)\log_2(1-\theta), \tag{4.3}
$$

where $\theta = \mathrm{P}(\hat{y}_t = 1)$. In Figure 4.1, we present how entropy σ_t is changing with respect to $\mathrm{P}(\hat{y}_t = 1)$. We observe the highest entropy value when probability of observing 1 is 0.5, namely, when the experts of the ensemble model are highly uncertain about the end prediction. As suggested in the previous paragraph, we exploit this observation and introduce a threshold ζ to set a criterion about the input sample. According to this criterion value, proposed active learning based algorithm includes or excludes the sample of interest to the training set.

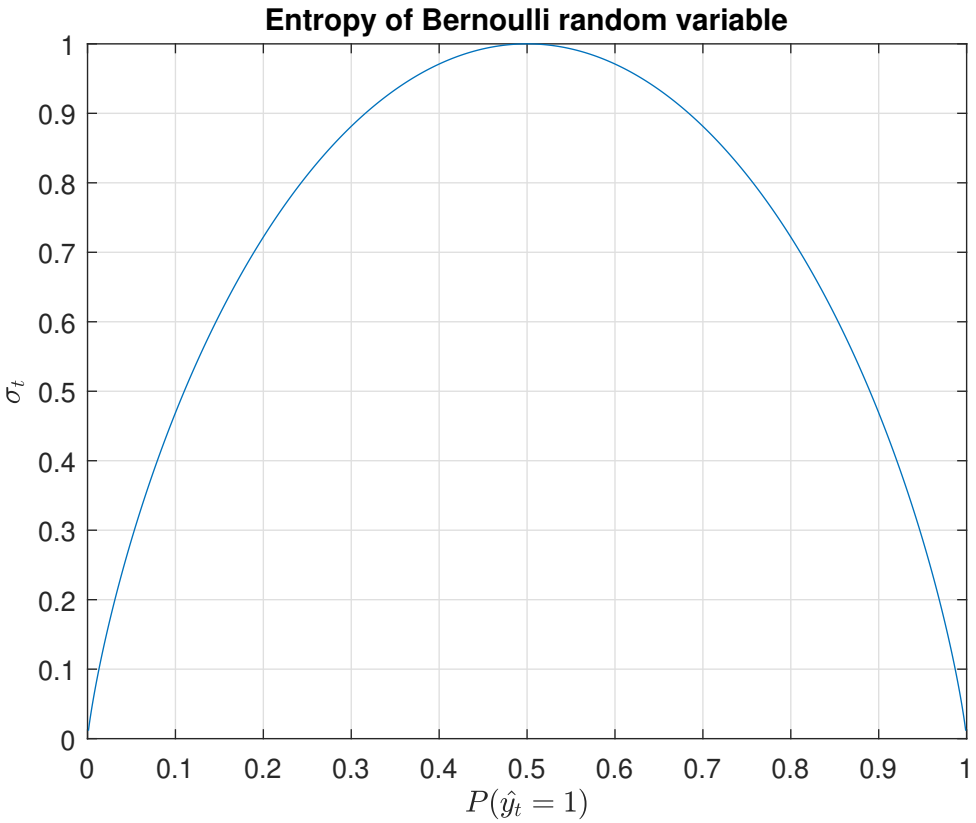


Figure 4.1 Entropy of Bernoulli random variable calculated using 4.3. X axis represents probability of observing $\hat{y} = 1$ (target) in the output. As uncertainty among different experts of the ensemble classifier increases, entropy comes close to 0.5 and we decide upon using the corresponding sample to improve training.

Application of uncertainity sampling forces model to collect more data when experts are more likely to be uncertain. In other words, unlike random sampling, ensemble classifier does not collect data uniformly from the whole feature space. For better explanation, we present Banana dataset with active learning method in Figure 4.2. The negative (non-target) and positive (target) classes are represented with different colors. To emphasize the difference between random sampling and active learning, we marked the first 200 samples that are selected for the learning. As seen in Figure 4.2, random sampling does not make any differentiation between the samples hence you can see that all marked samples are scattered through the whole sample space. However, in active learning case, we intentionally force ensemble model to sample from regions with high uncertainty (high entropy). Therefore, marked samples are closer to the decision boundaries.

Even though active learning seems to improve the performance convergence by selecting more useful samples, there are several problems that we want to mention in using proposed method with hierarchically organized NP models.

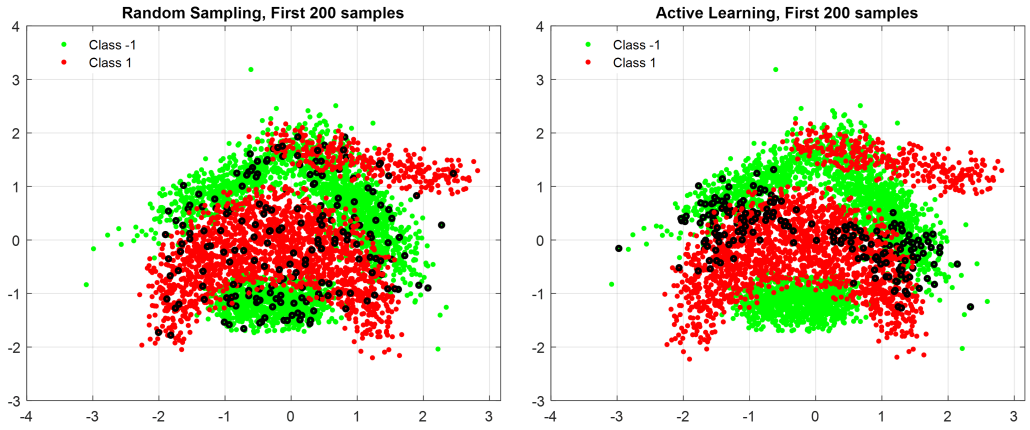


Figure 4.2 We use visually representative Banana dataset in order to compare sampling behaviour between random and active learning methods. Samples marked with black edges represent the first 200 samples processed by Active Tree OLNP under different active learning approaches. Random sampling collects data from all feature space while active learning has a tendency to collect data from the proximity of the decision boundaries.

1. Tree OLNP is an NP classifier which maximizes the detection power while upper bounding the false alarm rate by a threshold provided by the user. However, estimated false alarm should be calculated over independent and identically distributed random variables (without applying any selection to the data). Namely, when we apply uncertainity sampling, we are creating a selection bias by only using samples from a sub-region of whole feature space. Introduced bias causes the active learning-based data selection method to distort the natural statistics by only using samples which are close to the decision boundaries and prevents the measurement and achievement of the desired false alarm level.

2. Active learning approach outlined above is about learning to combine hierarchical experts effectively, namely, not about learning/maturing of each expert algorithm $f_{i,t-1}$ in an online manner. Therefore, the problem discussed in item 1 is also applicable here. In terms of inactive but general learning, our goal is to ensure that every single expert (not only their hierarchical ensemble) is learning the statistics of the data source. Our proposed active learning-based data selection, prevents this by limiting the learning problem only to the ensemble of the experts. This corrupts the original data statistics.

3. Proposed active learning based data selection corrupts data statistics in individual expert $f_{i,t-1}$ level as mentioned in item 2. Therefore, any criterion based on combination of these experts is also invalid. Because, used experts are not converging to the desired operating point.

4. Finally, it is not possible to observe all sides of the data space in a balanced way, since the data selection process, which occurs as a result of active learning, affects or even corrupts the original data statistics. For example, if the class distribution of a problem is not regionally uniform across the feature space, proposed active learning based algorithm prevents ensemble model from learning the actual class cost for the desired false alarm rate.

Algorithm Active Tree OLNP

```
Initialize exploration probability p_k and entropy threshold ζ.
for t = 1,2,... do
    z ← Z ~ U(0,1)
    if z < p_k then
        Include x_t in model training
    else
        Calculate σ using (4.3).
        if σ > ζ then
            Include x_t in model training
        else
            Exclude x_t from model training
        end if
    end if
end for
```

In order to solve the problems mentioned above, we propose to use exploration-exploitation trade-off which is widely studied in optimization of min-max problems such as multi-armed bandits and active sampling [137, 138]. According to this framework, during the training of Tree OLNP model, we use the input sample with an exploration probability of p_k and we apply active learning sampling method with a probability of $1-p_k$. In this approach, as we increase the exploration probability p_k total amount of samples used in learning will also increase but we will also be solving the 4 problems we mentioned above. We propose active learning based sampling algorithm containing the discovery (exploration) probability p_k and entropy threshold ζ for uncertainity sampling below. In the next section, we evaluate random and active learning based sampling methods by comparing achieved performances and used training samples.

4.3 Experiments

In this section, we evaluate performance of proposed active learning based Tree OLNP with random sampling on 6 different datasets. The details of the used datasets are given in Table 4.1. Since Tree OLNP is an online algorithm, the total number of samples in the training sets has been augmented 15×10^4 by repetitive concatenation of the dataset to ensure FPR convergence. It is important to note that augmentation is only applied to the training set and not for the test set. In our experiments, we use Tree OLNP with depth of 5 and selected the exploration probability $p_k \in \{0.1, 0.2, 0.3\}$. We repeat our experiments for multiple entropy threshold ($\zeta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$) values and share the results for all datasets in figures 4.3, 4.4, 4.5 and 4.6. Within the same visualization, we also present performance of random sampling where there is no criterion about including or excluding the incoming sample to the training. Visual explanation of the difference between including or excluding samples and their corresponding subset is given in Figure 4.2. The order of samples processed by Tree OLNP in each dataset is different. In order to handle randomness introduced by the sample order and show significance of the proposed model, we repeat all experiments 32 times and report aggregated results. At this point, we would like to give more detailed explanation of aggregation for visualizing the transient behaviour. There is randomness (because of the exploration probability p_k and shuffling of the input data stream) in our active learning algorithm. This causes our experiments to return different number of total training samples in each run (except for random sampling). In order to show mean transient behaviour using all experiments, we need to calculate point-wise average up to length of the shortest run (i.e run that have the least number of training samples).

Datasets	Number of Dimensions	n_+	n_-
banana	2	2376	2924
telescope	10	6688	12332
miniboone	50	36499	93565
phisihing	68	11055	6157
satellite	36	6435	4399
avila	10	20867	12295

Table 4.1 Details of the datasets used in performance evaluation of active learning based sampling compared to random sampling

4.4 Discussion

For aggregated results shown on the right hand side, we use 2 different metrics together with the total number of training samples used in the learning phase. These metrics are the true positive rate (TPR, detection power) and the NP score, respectively. It should be noted here that the goal in NP classification is to keep the false positive rate (FPR) below a user-specified threshold τ while minimizing the miss rate (maximizing the detection power). Therefore, both aspects of this target (minimum miss rate and FPR constraint) must be considered when evaluating the performance of any NP classifiers [15]. As explained in more detail in Chapter 2, it is possible to combine TPR and FPR as

$$\text{NP score} = \kappa \max(\hat{P}_{\text{fa}}(f) - \tau, 0) + \hat{P}_{\text{nd}}(f), \tag{4.4}$$

where $\kappa = 1/\tau$ is set to penalize relative divergence from target false alarm rate [15, 18].

According to our results in Figure 4.3 (discovery probability $p_k = 0.1$), in spite of extremely sparse data processing (majority of the samples are excluded because of the entropy criterion ζ), at similar TPR values, there is at least a 3-fold acceleration in data processing (compared to the random curve, thanks to active learning). On the other hand, no loss is observed in TFPR convergence. In the case of active learning, convergence of TPR is not completed yet, but this is an expected behaviour since our model selects only useful samples and stops unless its necessary. In a stream-based data processing environment, this will not be an issue since new data will always be available. We also observe that the difference between the end TPR values for $\zeta = 0.1$ is only 0.02 which is less then the variance of different runs. We observe higher number of training samples for $p_k = 0.3$ compared to $p_k = 0.1$ which is expected. Note that increasing discovery (exploration) probability will decrease the advantage gained from active sampling in favor of better false alarm convergence. The performance difference in transient response is not as high as the first case but it is still better than random sampling.

In Figure 4.4, we present advantages of active learning based sampling on Telescope dataset. Similar to the results achieved on Banana dataset, the affect of active learning is clearly visible, especially for $p_k = 0.1$. For all ζ (entropy threshold) values, we observe improvement in Tree OLNP's TPR convergence. To be more precise, around TPR$= 0.63$ (for random sampling), for all ζ values we observe TPR$= 0.72$. By looking at the end points of the transient graphs, we can also conclude that

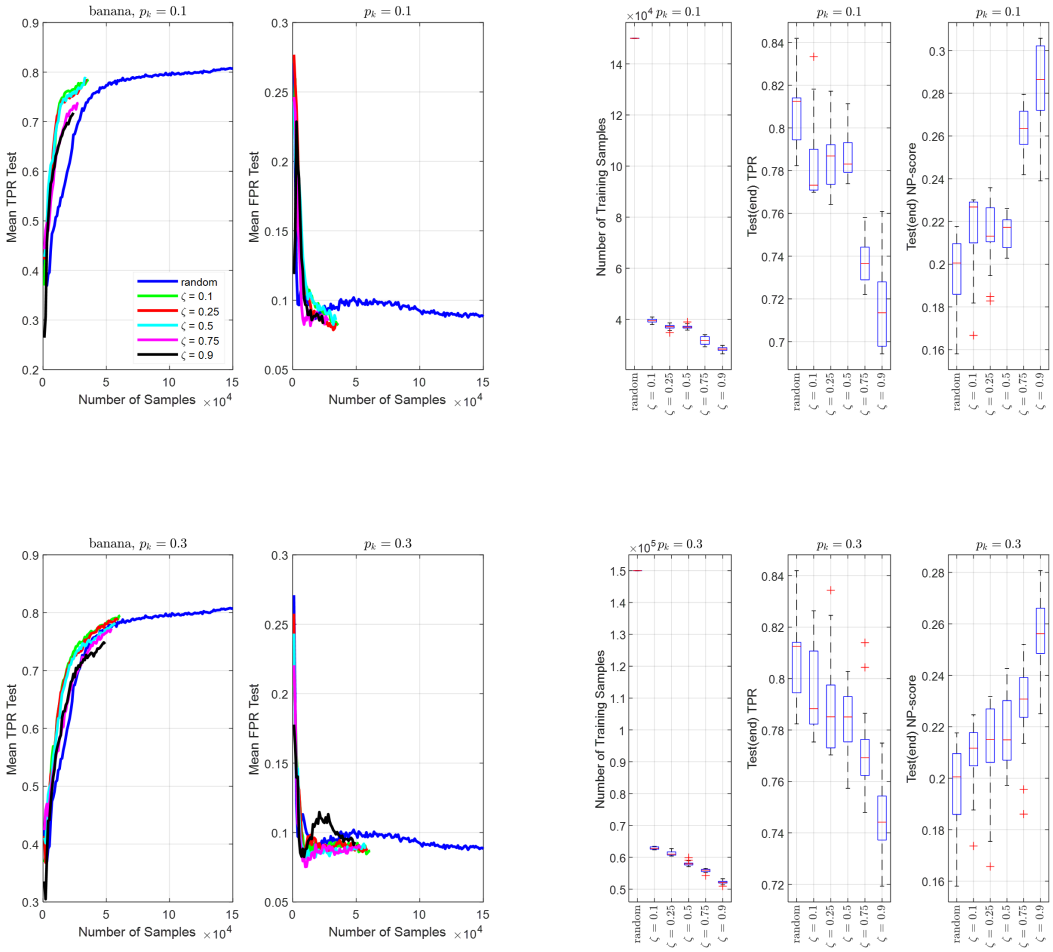


Figure 4.3 Transient behaviour of active Tree OLNP with $p_k \in \{0.1, 0.3\}$ and $\zeta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ for Banana dataset.

introducing active learning is not creating a major performance loss compared to the case where all samples are used (random sampling). Considering the FPR convergence in different cases, we observe ripples which is cause by the constant step size used in stochastic gradient descent updates. This can be solved by introducing a step size that decreases with increasing number of samples but in the scope of this chapter, since we did not encounter any convergence problems, we used constant step size. For $p_k = 0.1$ we also observe that for the experiments with $\zeta > 0.25$, TPR decreases and advantages of active learning starts to diminish.

For Miniboone_pid dataset, we observe similar performance to Banana and Telescope datasets. Considering the total samples used, for all datasets, random sampling uses all samples hence all of the experiments contain 15×10^4 samples. This number decreases drastically for active learning based sampling cases and it is inversely proportional to p_k as expected. We observe performance drop for $\zeta > 0.5$ and the convergence performance is low in $p_k = 0.3$ case compare to $p_k = 0.1$.

In Figure 4.6, we compare two methods using Phishing, Satellite and Avila datasets.

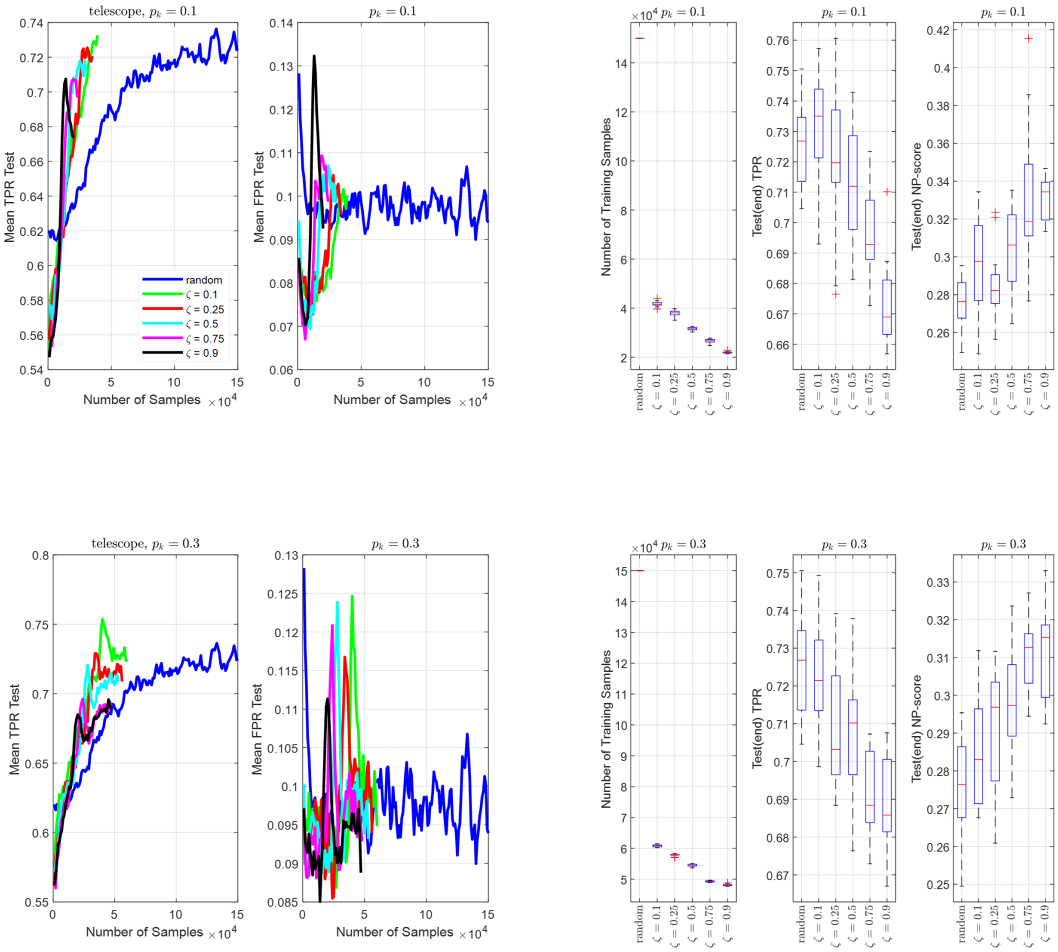


Figure 4.4 Transient behaviour of active Tree OLNP with $p_k \in \{0.1, 0.3\}$ and $\zeta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ for Telescope dataset.

In Phishing dataset, for all ζ values, we observe better TPR convergences in active learning based sampling compared to random sampling. Even when the whole samples are used by the random sampling, active learning successfully achieves similar TPR and NP-score.

4.5 Final Remarks

In this section, an active learning method for a context tree-based unified NP classifier (active Tree OLNP) is presented. Our proposed method decides whether or not to add the input sample to the training set according to the uncertainty (entropy) calculated from the prediction differences of the experts in the combined NP classifier. At the same time, the active learning method is skipped with a certain

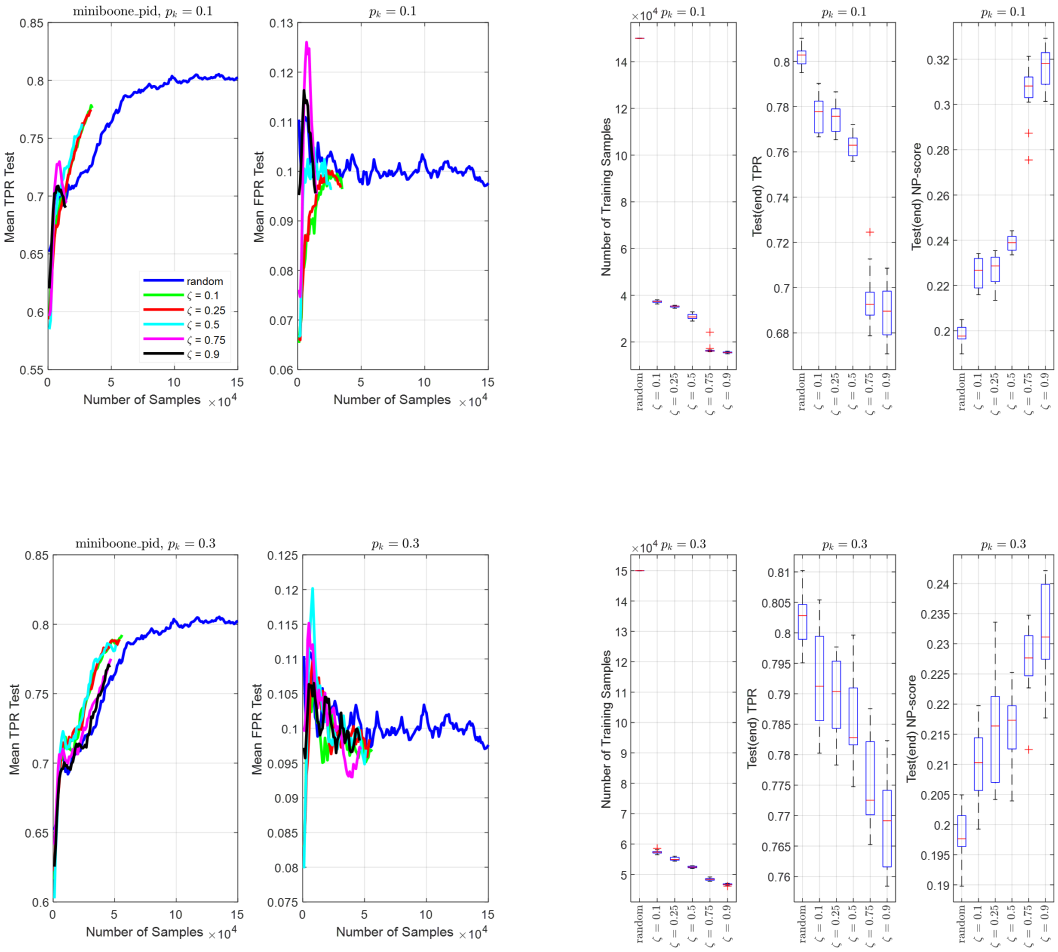


Figure 4.5 Transient behaviour of active Tree OLNP with $p_k \in \{0.1, 0.2, 0.3\}$ and $\zeta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ for Miniboone_pid dataset.

probability (exploration, discovery probability) in order to achieve the target false positive rate set by user. The main purpose of this modification is to prevent collecting samples only from the regions close to the decision boundaries with active learning but to collect from whole feature space. We present convergence improvement with the proposed active learning based sampling on 6 different datasets. We conduct the experiments for different discovery probabilities and entropy thresholds (ζ) and present the results in terms of number of samples used in experiments, TPR and NP-score.

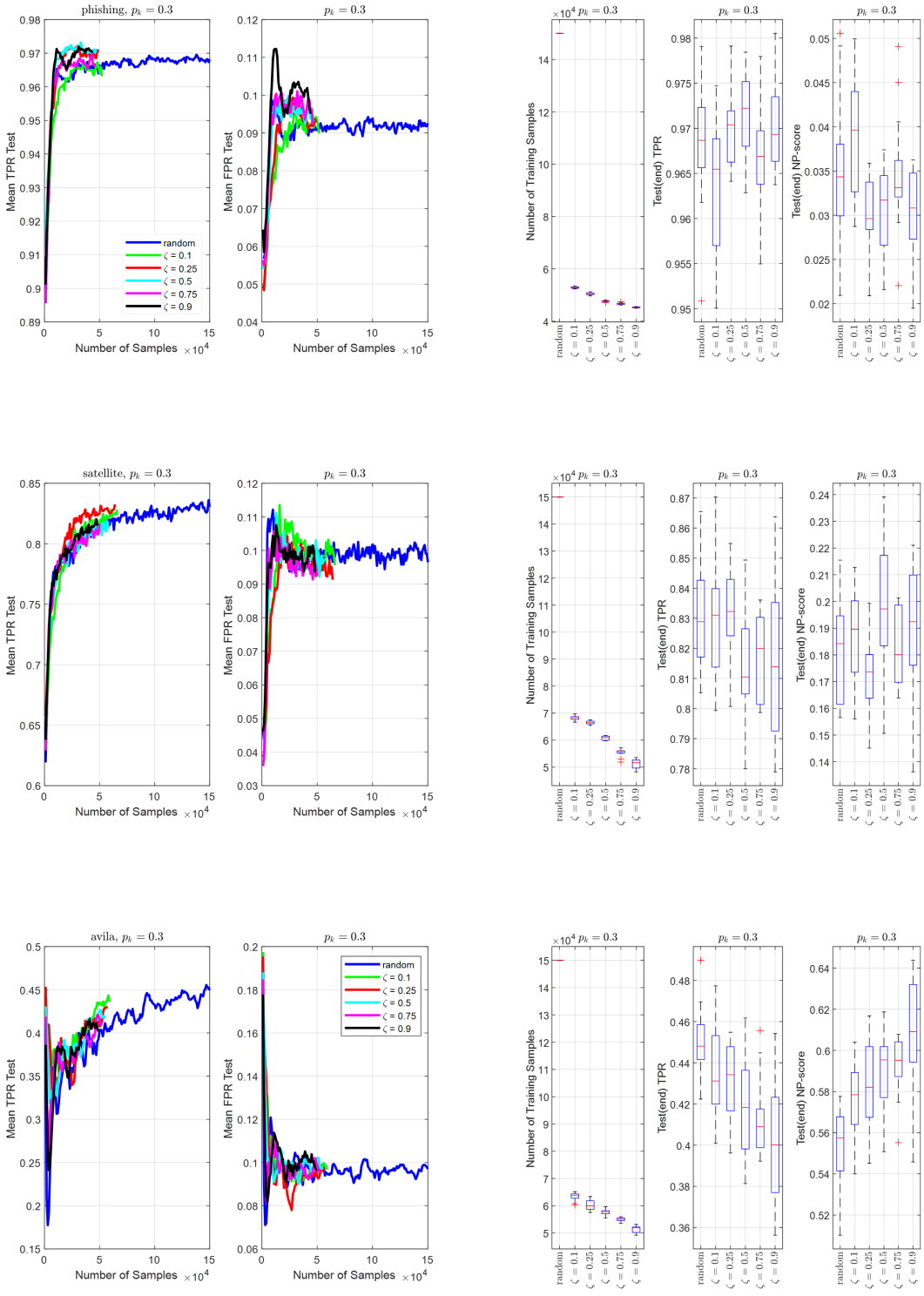


Figure 4.6 Transient behaviour of active Tree OLNP with $p_k = 0.3$ and $\zeta \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ for Phishing, Satellite and Avila datasets.

5. Video Application

Anomaly detection can be considered as detecting single or group of samples that do not represent statistics of the corresponding population [1]. Detecting these samples (anomalies, abnormal samples) is very crucial for business operations since they may contain useful information about an important flaw or a potential risk in the system/process. This drives research in anomaly detection to be popular especially about applications such as fraud detection [139], health condition monitoring [140], network security [141] and surveillance systems [142].

There are several challenges in detecting anomalies from raw video data. First challenge is implementing an accurate feature extraction system. Images collected from consecutive video frames are highly correlated and this needs to be taken into consideration while training the anomaly detection model. Second problem is the definition of an abnormal sample as we discussed in Chapter 1. An object/action may or may not be abnormal depending on the context.

Anomaly detection can be considered as a binary classification problem, where the class prior probabilities of abnormal (target) and normal (non-target) are highly unbalanced. However, in most of the cases, observing or acquiring abnormal samples are very uncommon and availability of target (abnormal) class is assumed to be limited to none. Under these circumstances, we can approach anomaly detection problem with 2 different methods. First option is using unsupervised learning, where we do not have access to abnormal data (i.e target class is not available) in the training set and we need to learn the existing manifold with only non-target samples. This can be addressed by estimating minimum volume set to cover at least α percent of the existing manifold (hence effectively achieving α false alarm rate under uniform target distribution) [13]. This comes with the assumption that the unseen samples (unavailable abnormal samples) are distributed uniformly towards the feature space and learning the aforementioned minimum volume set would solve the problem. Second option is supervised machine learning, where we both have target and non-target samples (even they have highly unbalanced prior probabilities) and we need to formulate the problem as a binary classification problem by

considering different class weights of the samples from different classes. In this thesis, we focus on online Neyman-Pearson classifiers, where we assume that samples from both classes (independent of their priors) are available to the proposed model. At this point of the section, I would like to remind our discussion about connection between anomaly detection and Neyman-Pearson classification as we have discussed in Chapter 1.

We have 2 publicly available and 1 manually curated video data sets for evaluating Tree OLNP. Last data set (SUrveillance) is created by our team in Sabancı University [143]. The goal of this chapter is to extract features that can be used in anomaly detection problem and use the extracted features in our proposed ensemble NP classifier model, Tree OLNP, in order to show that proposed model is a good candidate for detecting anomalies from video stream in an online manner.

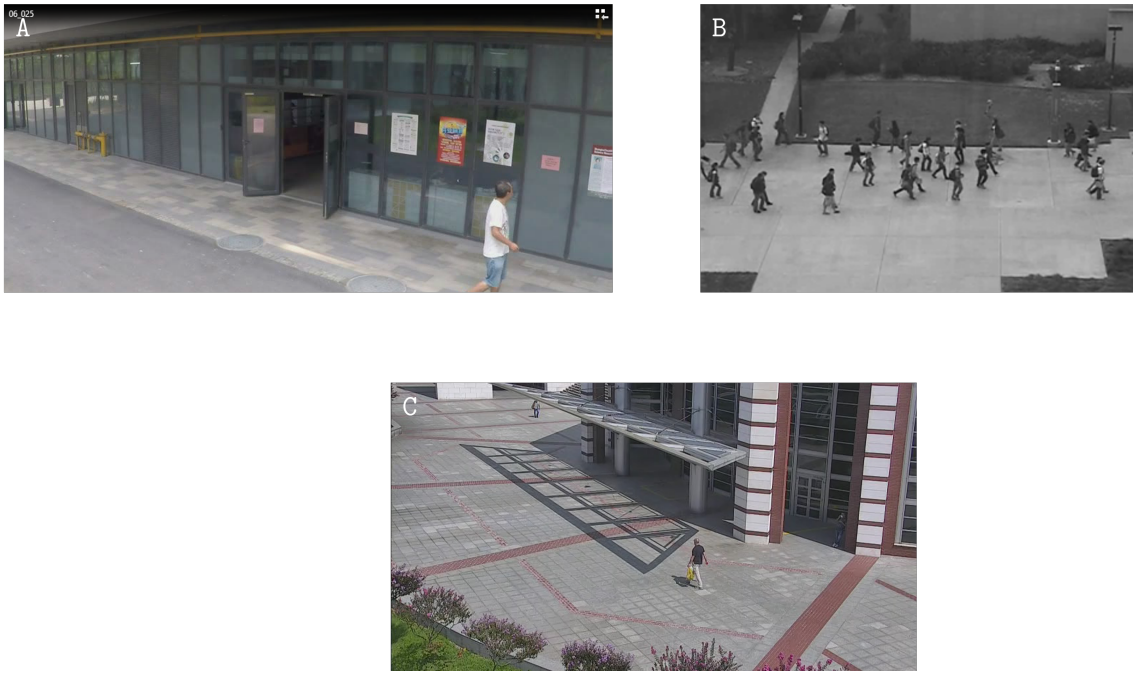


Figure 5.1 We present example scenes from Shanghai Tech (A), UCSD Ped 2 (B) and SUrveillance (C) data set which is used in evaluation of Tree OLNP for detecting abnormal events from video data. Images size of each data set are 856×480, 360×240 and 800×450, respectively.

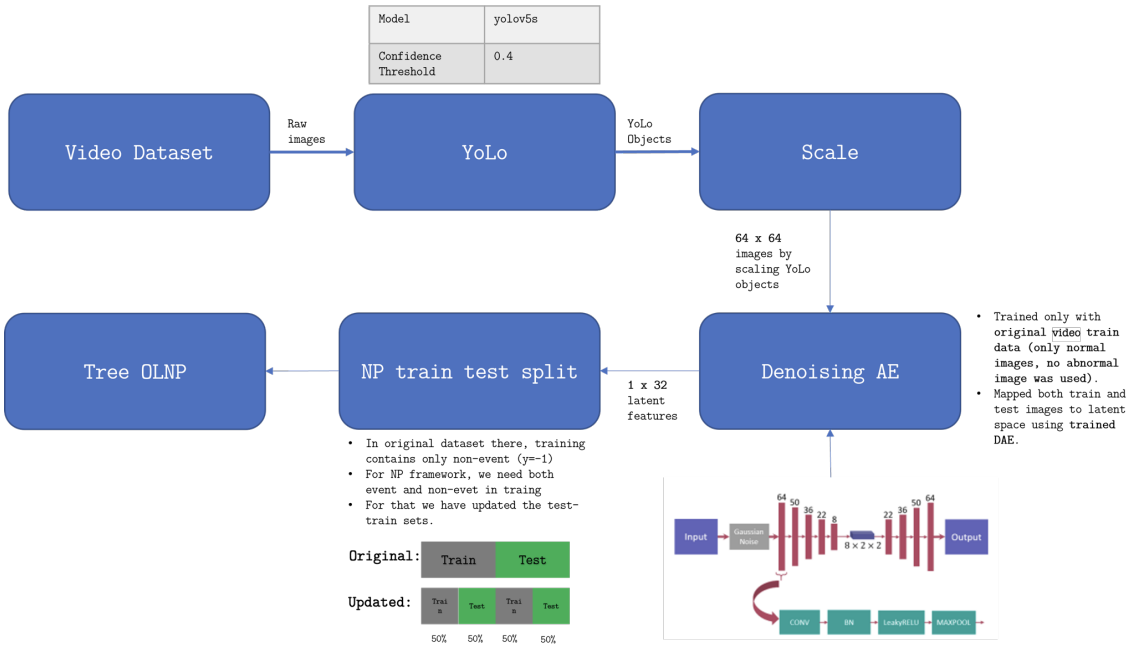


Figure 5.2 Video anomaly detection pipeline starts with object detection using YoLov5 (detailed description is available in Section 5.1) from raw video data (Shanghai Tech, UCSD ped2, SUrveillance). Generated objects are scaled and given to a Denoising Autoencoder in order to map scaled YoLo objects to low dimension latent space. Tree OLNP uses latent features to solve the NP problem.

5.1 Problem Description

In this chapter, we consume raw video image and aim to detect abnormal objects/actions using proposed ensemble NP classifier Tree OLNP (for detailed explanation please refer to Chapter 3). We use Shanghai Tech 6 [144] (Shanghai Tech has 12 different scenes, we select scene 6 for our analysis) and UCSD Ped 2 [145] for initial evaluation of Tree OLNP. In addition, we also use SUrveillance dataset [143] which is generated by our team at Sabancı University. Visual samples for each dataset and their corresponding frame sizes are available in Figure 5.1. Note that these datasets are used for training models to solve video anomaly detection problem, therefore their training data only contain non-target labels.

Each collected image (frame) from video is high dimensional and highly correlated (consecutive frames in a video have correlated pixel values). We need to extract important features from video data and use it as an input to Tree OLNP. We train the pipeline for extracting features in an offline manner. Focus of this thesis to propose online nonlinear NP classifier hence we assume that input samples (features extracted from video stream) for Tree OLNP are readily available in real time.

Therefore, once the feature extraction pipeline is available, it will be able to generate stream of data which can be consumed by Tree OLNP for video anomaly detection. We present full pipeline for anomaly detection from video data in Figure 5.2 (denoising autoencoder architecture image is taken from [146]) and explain each block in more detail in below.

Video anomaly detection pipeline starts with object detection using YoLov5 (which has same architecture with YoLo4) object detector [43]. We set the confidence threshold (hyperparameter) of YoLov5 as 0.4 and only allow "person, car, skateboard, wheelchair, bag, wheelbarrow (with and without motor), bicycle, cat, dog" objects to be detected in order limit the object space. Note that every detected object (we will call as YoLo objects) are at different sizes based on the detection window sized calculated by YoLov5. Next step is our pipeline is scaling of the individual YoLo objects to 64×64 so that we can use a denoising auto encoder (DAE) to map scaled objects to lower dimensional latent space. We use the same DAE architecture introduced in [146] as shown in Figure 5.3 (Image is taken from [146]).

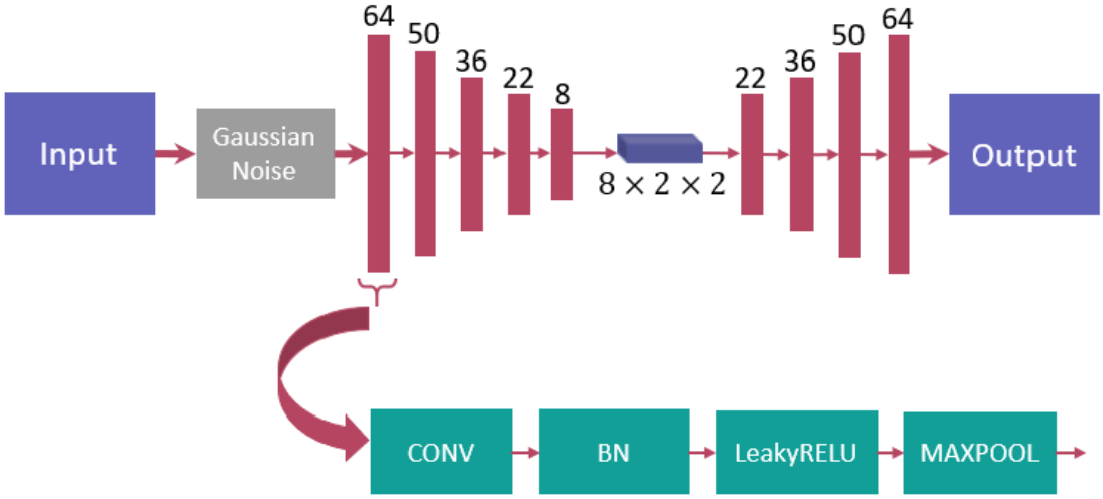


Figure 5.3 Architecture of the DAE. BN: batch normalization.

In the process of training DAE given in Figure 5.4, we created validation set using 15% of training YoLo objects. Note that, each sample in our training set are YoLo objects but, we also save their corresponding timestamps from the initial raw video as metadata prior to object detection part. Therefore, we apply all of the splits (train, validation and test) using these timestamps to make sure there is no leakage between created sets (redefining the aforementioned sets will be necessary in solving the anomaly detection problem in NP framework). Namely, for this chapter, all of the splits are applied through time stamps, not through random shuffling. In Figure 5.4, we present transient training behaviour of DAE. During the training,

we set batch size as 32 and total epoch as 100. At each epoch, to better visualize overfitting/underfitting, we evaluate DAE with the validation set. Figure 5.4 shows that for UCSD Ped 2, DAE successfully mapped YoLo objects to latent space but for Shanghai Tech and SUrveillance, there is a small difference between the validation error and training error. The difference is not much but, this issue should be addressed if the performance of Tree OLNP for detecting abnormal scenes is not satisfactory.

The latent space of DAE has size of 32, namely, using the DAE shown in Figure 5.3, we map detected YoLo objects to a 32 dimensional latent space. Even though Figure 5.4 is a good representation of successful training, we also present input and their reconstructed images in Figure 5.5. Visual comparison with the reconstructed and actual images show that, latent variables are good candidates for further statistical analysis.

In addition to the latent 32 dimensions, as suggested by the authors of [146], we use an additional feature called SPNR, which is calculated as

$$PSNR(\boldsymbol{z}, \boldsymbol{z}_{rec}) = 10log\frac{\max(\boldsymbol{z})}{MSE(\boldsymbol{z}, \boldsymbol{z}_{rec})}, \tag{5.1}$$

where $\max(\boldsymbol{z})$ is the maximum pixel value for the input image. SPNR explains reconstruction error for any input image $\boldsymbol{z}$ and contains much information about the abnormal cases. Since DAE is trained only on non-target data, it will have high reconstruction error for abnormal samples (not available in training set). Adding SPNR with concatenation to the latent features created by DAE, we end up with the input (for Tree OLNP) $\mathbb{R}^d \ni \boldsymbol{x}$, where $d = 33$ (images are mapped to latent space as $\mathcal{H}(\boldsymbol{x}_t) = \boldsymbol{x}_t$ with autoencoder $\mathcal{H}$). In the next section, we will describe how this problem is adopted to NP framework and present evaluation results.

5.2 Experiments

In this thesis we focus on online nonlinear NP classifiers, namely models that maximizes detection power while upper bounding the false alarm rate by a user defined threshold [17]. It is possible to solve the anomaly detection problems by assuming abnormal (target) samples are uniformly distributed through out the feature space and solve MVS problem as explained in Chapter 1. However, for the case of video

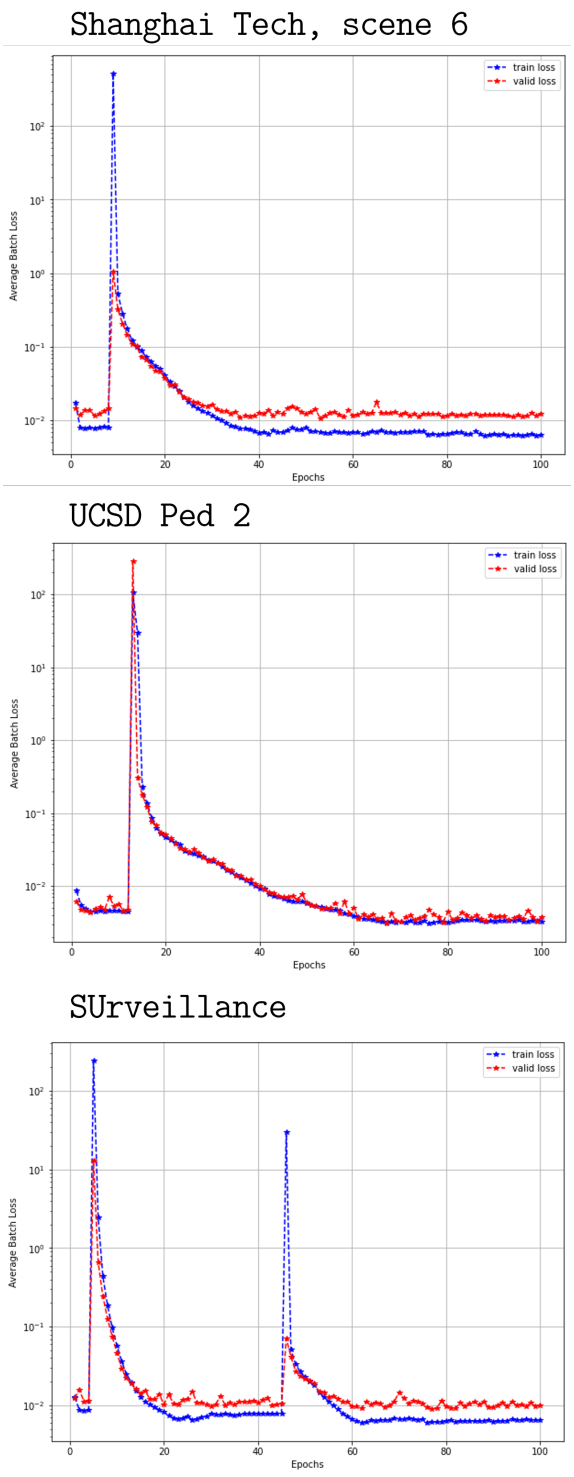


Figure 5.4 Learning curves containing training and validation losses for Shanghai Tech scene 6, UCSD Ped 2 and SUrveillance datasets. We train denoising autoencoder (DAE) for 100 epochs and for each data point, we present training and batch losses to better visualize learning performance throughout training process. DAE architecture seems to generalize better for UCSD Ped 2 dataset compared to Shanghai Tech and SUrveillance datasets considering the difference between train and validation losses.

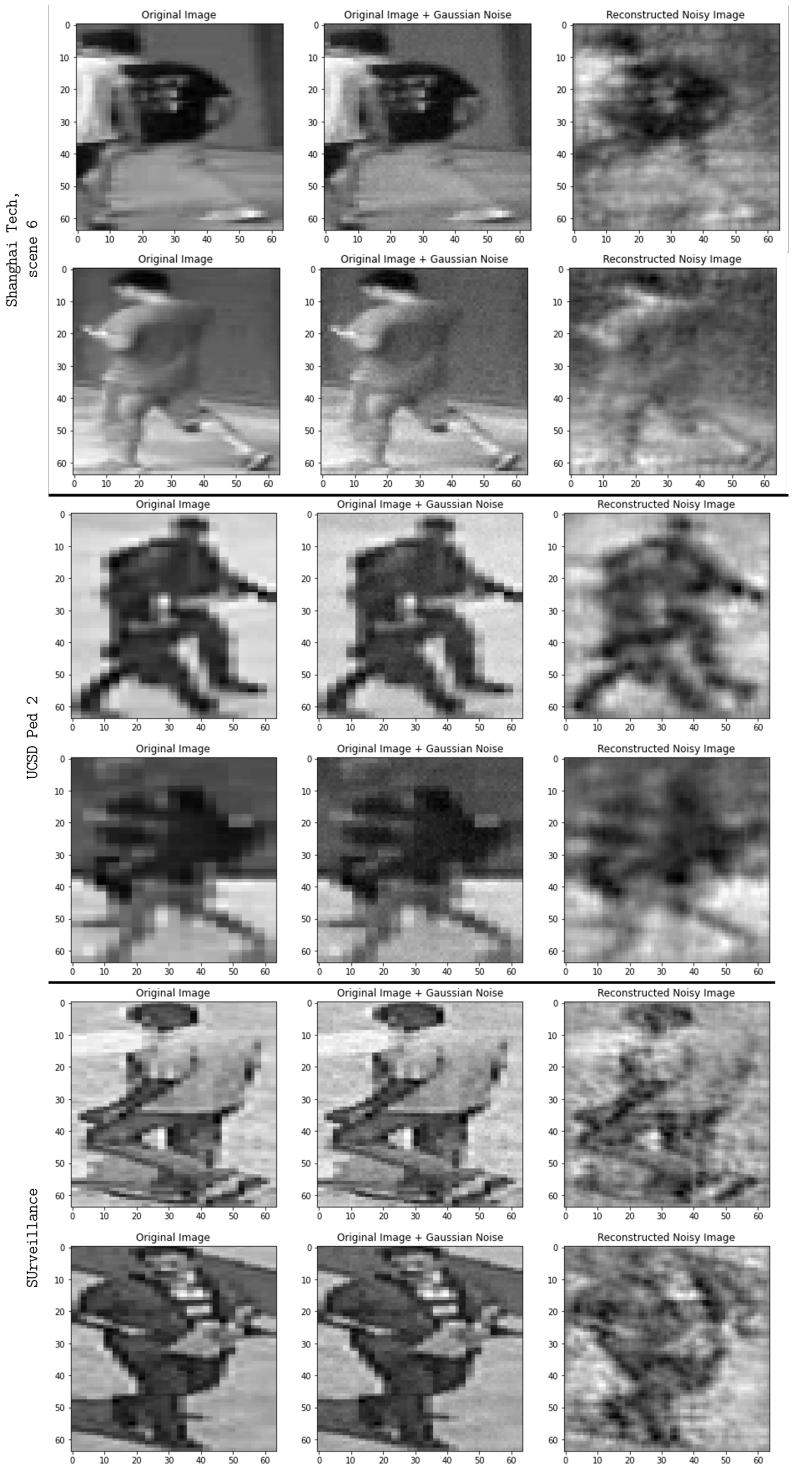


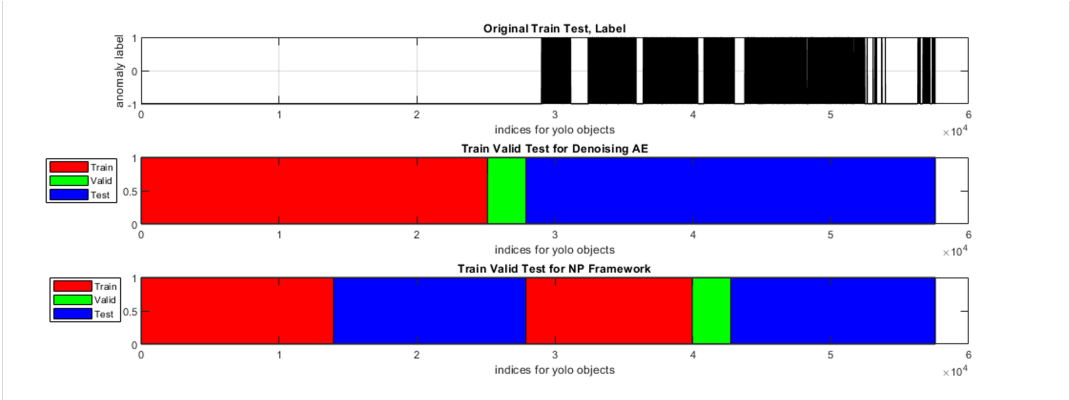
Figure 5.5 Figures in the first column on the left are example images from Shanghai Tech, UCSD Ped 2 and SUrveillance datasets. We add Gaussian noise $\mathcal{N}(0, 0.01)$ to the initial images in order to create generate training input for DAE. Last images present reconstruction of the noisy image by DAE.

anomaly detection problem, where statistically meaningful low dimensional latent features are created via encoding with denoising auto encoder, uniform generation of target samples in 33 dimensional space is challenging and can affect performance

of Tree OLNP. Therefore we are opt to use the second option, where we need target samples in our training set. Based on the timestamps of the mapped YoLo objects, we create new train, validation and test sets for Tree OLNP. An example data separation for creating train, validation and test sets for UCSD Ped 2 is shown in Figure 5.6. X-axes in the first 3 graph represent YoLo object index (not time). However, since the frames which we detect the YoLo objects from are in order, YoLo indices and time values are correlated. First figure shows target ($y = 1$) and non-target ($y = -1$) labels. As explained, since there is no abnormal samples in the training set of UCSD Ped 2, we do not observe any $y = 1$ in the training phase. Second graph shows set separation for DAE. Note that there is no abnormal data is available in training or validation sets. Once the training of DAE is completed (please refer Figure 5.4 for training behaviour of DAE on each dataset), we use trained model to encode all images ($\mathcal{H}(\boldsymbol{z}_t) = \boldsymbol{x}_t$) to 33 dimensional space (32 latent dimension + SPNR). At this stage of the pipeline, we should train Tree OLNP with input $\boldsymbol{x}_t$. Unfortunately, we need target data in training set. In order to resolve this issue, without violating temporal correlations and using metadata of individual YoLo objects for redefining new subsets, we generate new training, validation and test sets shown in the third figure. Note that second and third figures share the same indices in terms of YoLo objects, but their corresponding set (training, validation or test) are different. This difference is visualized by color coding. After redefining the sets, we also present prior class distributions for all of the sets. We do not need separate validation set in training Tree OLNP, therefore in the learning phase, validation and training sets are concatenated.

Figure 5.7 presents evaluation of Tree OLNP on all datasets. We share ROC curves on the left column with their corresponding AUC values. Each point in these ROC curves are individual Tree OLNP classifiers. Namely, each of these classifiers aim to solve different NP problem (with respect to the target false alarm rate given in x-axis). We repeat the experiments for every false alarm rate 32 times and calculated the statistics of the operating point by taking the average of 32 TPRs and FPRs. In the figures given on the right column, we present transient behaviour of Tree OLNP for $\tau = 0.1$, for all datasets. Blue line represents mean of 32 individual experiments whereas dashed red line represents corresponding standard deviation for that point.

Each input $\boldsymbol{x}$ contains 33 dimensions where the first 32 are the latent variables collected from DAE and last feature is SPNR which is calculated as 5.1. Since we also save the meta data of each YoLo object $\boldsymbol{z}_t$ (64 image), it is also possible for us to track back and visualize the actual abnormal objects/actions in their frames. We present examples of true positive and false positive frames for each data set in Figure 5.8. For Shanghai Tech 6, guy riding a bicycle (abnormal behaviour for the context



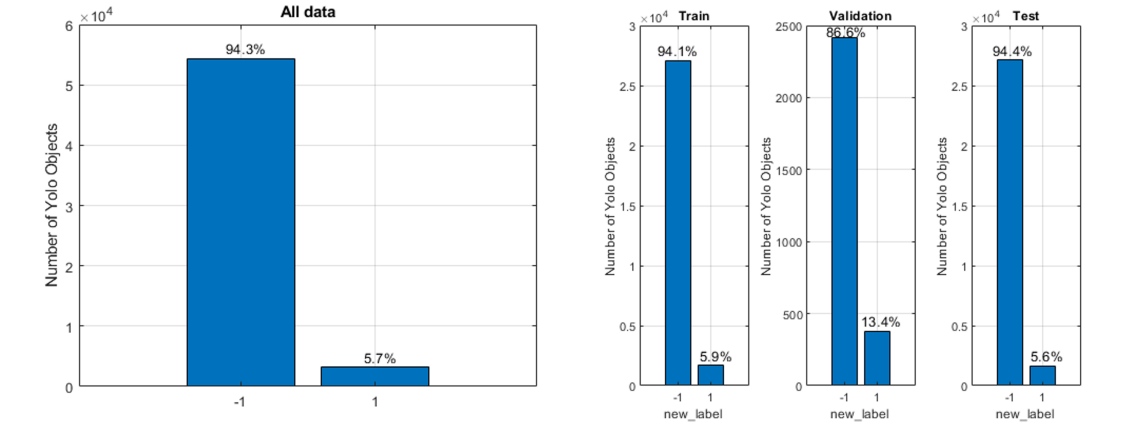


Figure 5.6 Training, validation and test sets definitions for UCSD Ped 2 dataset. Top figure visualizes how target vs non-target is changing with time and their corresponding distribution is given on the bottom left figure. We train DAE using the default setup, where training and validation does not contain any abnormal samples. However, Tree OLNP requires abnormal samples ($y = 1$) in training phase (even its frequency is very low). We handle target unavailability by redefining training, validation and test sets as shown on the third figure. Figure on bottom right shows class prior probabilities for the newly generated sets.

of video) is successfully detected by our algorithm. For the same video, another guy with a backpack is also labeled as abnormal by our algorithm, which is in fact a false alarm. Note that existence of a backpack should create difference compared to statistics of people (without any additional accessory) therefore it is reasonable to have such prediction error. In UCSD Ped 2 dataset, people with bikes and skateboards are successfully labeled as abnormal. In the false positive frames, we observe running people and a static object on the background are labelled as abnormal. The reason for latter error is the localization method of our pipeline. We detect objects and then encode them using DAE, which does not have too much information about the proximity environment. We left studying different architectures in DAE as a future research topic to better capture environment information to improve overall performance. In the latest dataset (SUrveillance) we observe similar successful

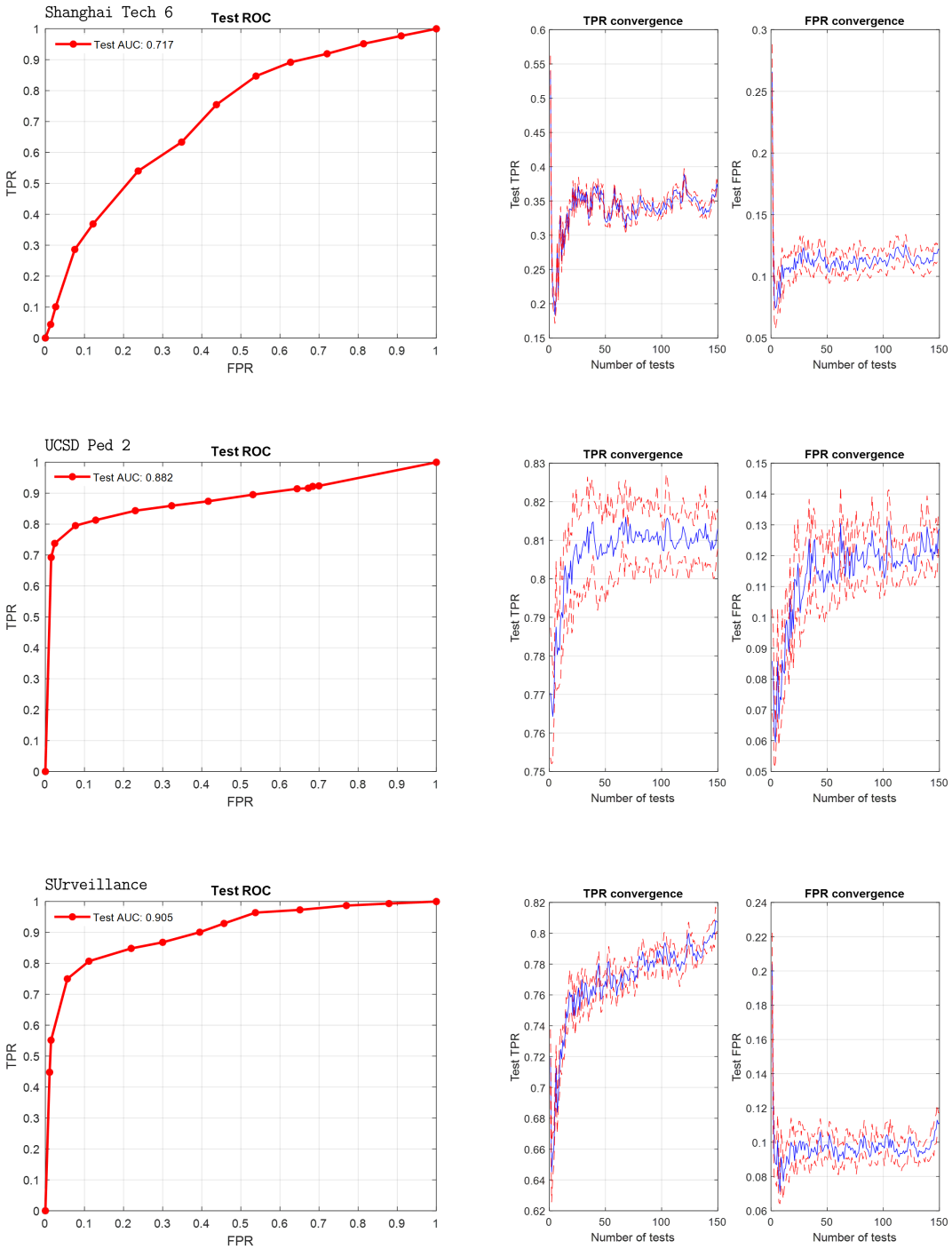


Figure 5.7 We present ROC curves of Tree OLNP for all video datasets with its transient TPR and FPR behaviour for $\tau = 0.1$ Each point in ROC curve is average of 32 different experiments and each point represents different NP classifier. Similarly, transient curves are also average of 32 individual runs where blue lines represent mean of the run and dashed red lines represent their corresponding standard deviations. AUCs are available as an additional information in ROC curves indicating that Tree OLNP shows good performance.

results as Shanghai Tech 6 and UCSD Ped 2. Algorithm successfully identifies vehicles and people carrying backpacks as abnormal samples (in the labelling process,

we indicate that people carrying additional accessories such as bags and etc. are abnormal. Therefore discussion for this detection is different than UCSD Ped 2 and Shanghai Tech 6).

Figure 5.8 Using Tree OLNP on Shanghai Tech 6, UCSD Ped 2 and SUrveillance, we calculate performance of our algorithm with ROC curves and transient behaviours of TPR and FPR in Figure 5.7. Meta data of individual objects are also saved therefore it is also possible to track back calculated TPR and FPR values and visualize actual TP and FP objects in their corresponding frames. In above figure we share examples of these frames for each data set.

5.3 Application of Active Learning

In Chapter 4, we introduce active learning based sampling in order to increase convergence rate of the ensemble classifier. The main idea of the proposed method

is to quantify the uncertainity (by calculating entropy of the predictions of the experts in the ensemble classifier using Equation 4.3) between the predictions of the experts (piecewise OLNPs) and decide upon including or excluding incoming sample $\boldsymbol{x}_t$ in training. We calculate the final decision by comparing calculated uncertainity amount with a parameter ζ (entropy threshold).

After projection of video data to latent space, video anomaly detection is also same as the data we investigate in the previous chapters of this thesis. Therefore, same idea we use in active learning for convergence improvement is also applicable to video anomaly detection pipeline. We conduct our experiments on all video sets (Shanghai Tech 6, UCSD Ped 2 and SUrveillance) with Tree OLNP of depth 4 with target false alarm $\tau = 0.1$. We repeat all runs 32 times and reported achieved TPR, FPR values in addition with the total number of training samples used. Note that these results are calculated on the test sets not on the training sets. In Figure 5.9, we present results of individual runs as boxplots. For all datasets, in the first figure, total number of samples used for random sampling method is always 15×10^4. As we increase the value of ζ, active learning sampling based method becomes more selective hence uses less number of sample during training. Please refer to Chapter 4 for more detailed explanation about entropy based sampling method. In all datasets, random sampling managed to achieve higher TPR compared to active learning approaches. However, differences are not more than 3%. In addition, for Shanghai Tech 6 and UCSD Ped 2 datasets, we observe better false alarm controllability compared to random sampling case, since the NP-score for active learning cases are lower (lower NP-score is better [15]) than the default case (random sampling).

5.4 Final Remarks

In this section, a video processing pipeline for detecting abnormal objects/behaviours is presented. Training phase of the proposed pipeline (up to Tree OLNP), is conducted in an offline manner. Since our proposed ensemble NP classifier (Tree OLNP) requires input data as a stream of input samples, we opt to train feature extraction part first in an offline manner. Pipeline starts with detecting objects using YoLov5 [43] object detector. Then detected objects (cropped images) are scaled to 64×64 in order to be used as an input to denoising autoencoder described in [146]. We train the network using normal data (not containing any abnormal samples). Once training is completed, all data is projected to latent space using

trained DAE. Latent features (collected from the bottleneck layer of DAE) are used as an input to Tree OLNP. One modification we applied at this point is recreation of training, validation and test sets. Proposed Tree OLNP requires target samples in the training set whereas they are not available as the original problem is anomaly detection and only contains non-target samples in its training set. In Figure 5.6, we present how new sets are created without violating temporal correlation between the samples and not allowing any information leakage. We present ROC curves in addition to transient TPR and FPR graphs. For all datasets, Tree OLNP successfully solves NP problem and achieve high AUC. We also observe that our proposed model successfully converge to desired false alarm rate in all experiments. As a continuation to our analysis, we also apply active learning based sampling introduced in 4 to improve convergence behaviour of Tree OLNP. We observe that by selecting for "useful" samples, thanks to uncertainity sampling introduced in Chapter 4, Tree OLNP is able to achieve almost as good as results compared to random sampling case, which requires all available samples. This drastically decreased the number of necessary samples to achieve desired performance, effectively increasing the efficiency of the ensemble model (up to 3 fold).

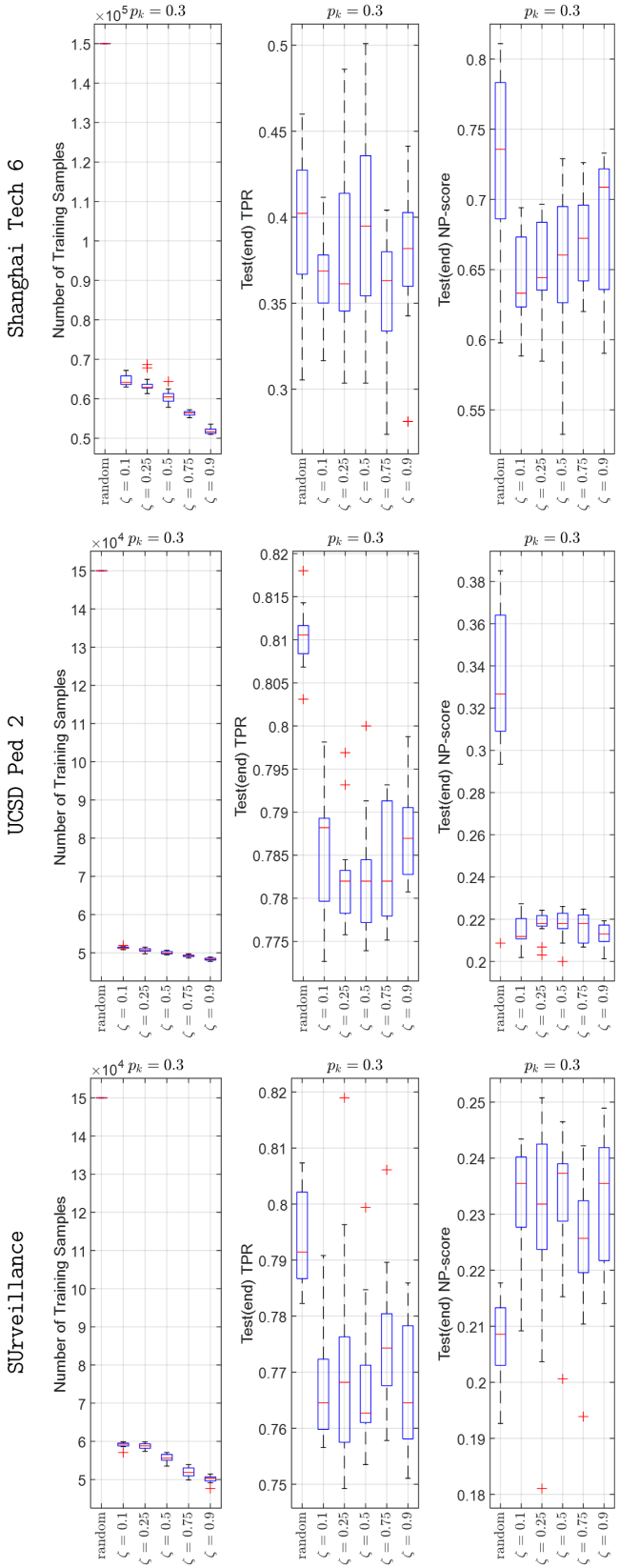


Figure 5.9 For Shanghai Tech 6, UCSD Ped 2 and SUrveillance datasets, we applied active learning based sampling method to improve performance convergence as explained in Chapter 4. We set exploration (discovery) probability $p_k = 0.3$ and share distribution of total used samples in training with TPR and FPR values for different entropy threshold ζ values.

6. Conclusion

In this thesis, we approached anomaly detection problem with Neyman-Pearson (NP) classification framework, which requires maximization of detection power while upper-bounding the false alarm rate by a user defined threshold. Introduced algorithms are computationally scalable to voluminous data with online processing. Firstly, we propose a single hidden layer feed forward neural network based (SLFN) based NP classifier. In the first hidden layer, inner products of random Fourier features converge exponentially to true kernel with sinusoidal activations. With the stochastic gradient descent, network continues to learn optimum mapping by updating hidden layer parameters. Output layer follows as a perceptron with identity activation. Considering the full architecture, we update network parameters with stochastic gradient descent minimizing Lagrange loss. Achieved model is computationally scalable, nonlinear and satisfies the desired false alarm rate while maximizing the detection power. Comparison of the proposed NP classifier with other state-of-the-art algorithms show that NP-NN outperforms online NP classifier OLNP in terms of FPR convergence and TPR. We also observe that proposed NP-NN performs at least as good as state of the art batch NP algorithms, which are not online, i.e not scalable to voluminous data.

One disadvantage of NP-NN is the usage of radial basis function (RBF) kernel, which requires extensive hyperparameter tuning and makes the model sensitive to overfitting. In order to handle this, we propose context tree based ensemble NP model (Tree OLNP), which dynamically controls model complexity in data driven manner. Context tree framework divides the feature space in to regions and in each region, we train individual OLNP (in the scope of this thesis we use OLNP as base classifier but, any online algorithm can be used here). We can represent the whole feature space as different combinations of the created regions. Each of this combination, i.e partition, represents piecewise NP models (experts) with different complexities. Ensemble model favors simpler experts when total number of processed data is low and it starts increasing weight of more complex models (experts) as number of processed data increases. Number of experts in the ensemble model increases doubly

exponentially with the increasing context tree depth, but thank to hierarchical organization of experts (tree structure) and efficient computation algorithm, learning can be decreased to linear time. We present that Tree OLNP is capable of achieving similar performance as NP-NN with less parameters and without any parameter tuning effort. More importantly, proposed simplification does not violates any NP constraints and model can successfully achieve target false alarm rate.

Active learning is a sampling method to select "useful" samples from stream of data to increase convergence behaviour of the model. We propose uncertainty sampling based active learning approach to Tree OLNP to have better transient behaviour. Uncertainity is quantified by calculating the entropy using the prediction generated by the experts (piecewise OLNPs) of Tree OLNP. While increasing the convergence performance is critical, since Tree OLNP is an NP algorithm, it should satisfy target false alarm rate. Active learning based sampling prevents model from learning actual statistics of the classes since it forces to use samples from the proximity of the decision boundaries. In order to solve this issue, we propose to use exploration-exploitation framework where Tree OLNP learns class costs in the exploration state and updates model parameters in exploitation. Even though exploration decreases the performance advantage of active learning, we are able to satisfy target false alarm rate. In this thesis, we use constant exploration probability and investigate performance improvement of Tree OLNP, which is a greedy approach. The convergence performance can further be improved by introducing Thompson sampling [147] or applying upper confidence bound (UCB) algorithms [148], which are common approaches in improving convergence behaviour in optimization of min-max problems such as multi-armed bandits.

Furthermore, we present an application of proposed Tree OLNP classifier about anomaly detection from video stream. We developed a pipeline, starting with object detection from videos. Detected objects are scaled and then encoded using denoising autoencoder (DAE) to lower dimensional latent space. We use latent features as an input to Tree OLNP and detect abnormal objects/actions. In addition, we also apply active learning based sampling to show that convergence in performance is also applicable to video data. The DAE architecture for encoding detected objects can further be optimized for better performance. Another important component in the pipeline that is affecting the overall performance is the training of DAE. In this thesis we use the architecture and training parameters proposed in [146]. However, one important difference is, after video frames are mapped to latent space, in [146], authors uses Gaussian mixture models to detect anomalies, whereas we feed latent variables to NP framework. Considering the differences in the pipeline, architecture and training parameters of DAE should be revisited for maximum performance.

BIBLIOGRAPHY

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[2] Sailesh Kumar. Survey of current network intrusion detection techniques. *Washington Univ. in St. Louis*, pages 1–18, 2007.

[3] Weiming Hu, Xuejuan Xiao, Dan Xie, and Tieniu Tan. Traffic accident prediction using vehicle tracking and trajectory analysis. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, volume 1, pages 220–225. IEEE, 2003.

[4] Yisroel Mirsky, Asaf Shabtai, Bracha Shapira, Yuval Elovici, and Lior Rokach. Anomaly detection for smartphone data streams. *Pervasive and Mobile Computing*, 35:83–107, 2017.

[5] Phuong Hanh Tran, Kim Phuc Tran, Truong Thu Huong, Cédric Heuchenne, Phuong HienTran, and Thi Minh Huong Le. Real time data-driven approaches for credit card fraud detection. In *Proceedings of the 2018 International Conference on E-Business and Applications*, pages 6–9. ACM, 2018.

[6] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Trans. Knowl. Data Eng.*, 19(5):631–645, 2007.

[7] Yu Zheng, Huichu Zhang, and Yong Yu. Detecting collective anomalies from multiple spatio-temporal datasets across different domains. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, page 2. ACM, 2015.

[8] Nasser M Nasrabadi. Hyperspectral target detection: An overview of current and future challenges. *IEEE Signal Processing Magazine*, 31(1):34–44, 2013.

[9] Irving S Reed and Xiaoli Yu. Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(10):1760–1770, 1990.

[10] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

[11] Heesung Kwon and Nasser M Nasrabadi. Kernel rx-algorithm: A nonlinear anomaly detector for hyperspectral imagery. *IEEE transactions on Geoscience and Remote Sensing*, 43(2):388–397, 2005.

[12] Laura Rettig, Mourad Khayati, Philippe Cudré-Mauroux, and Michał Piórkowski. Online anomaly detection over big data streams. In *Applied Data Science*, pages 289–312. Springer, 2019.

[13] Sakti Saurav, Pankaj Malhotra, Vishnu TV, Narendhar Gugulothu, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Online anomaly detection with concept drift adaptation using recurrent neural networks. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 78–87. ACM, 2018.

[14] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[15] Clayton Scott. Performance measures for neyman–pearson classification. *IEEE Transactions on Information Theory*, 53(8):2852–2863, 2007.

[16] Xin Tong, Yang Feng, and Anqi Zhao. A survey on neyman-pearson classification and suggestions for future research. *Wiley Interdisciplinary Reviews: Computational Statistics*, 8(2):64–81, 2016.

[17] H Vincent Poor. *An introduction to signal detection and estimation*. Springer Science & Business Media, 2013.

[18] Mark A Davenport, Richard G Baraniuk, and Clayton D Scott. Tuning support vector machines for minimax and neyman-pearson classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(10):1888–1898, 2010.

[19] Xu-Ying Liu and Zhi-Hua Zhou. Learning with cost intervals. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 403–412. ACM, 2010.

[20] Xin Tong, Yang Feng, and Jingyi Jessica Li. Neyman-pearson classification algorithms and np receiver operating characteristics. *Science advances*, 4(2):eaao1659, 2018.

[21] Ao Zhang, Nan Li, Jian Pu, Jun Wang, Junchi Yan, and Hongyuan Zha. Tau-fpl: Tolerance-constrained learning in linear time. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] Gilles Gasso, Aristidis Pappaioannou, Marina Spivak, and Léon Bottou. Batch and online learning algorithms for nonconvex neyman-pearson classification. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–19, 2011.

[23] Adam Cannon, James Howse, Don Hush, and Clint Scovel. Learning with the neyman-pearson and min-max criteria. *Los Alamos National Laboratory, Tech. Rep. LA-UR*, pages 02–2951, 2002.

[24] David Casasent and Xue-wen Chen. Radial basis function neural networks for nonlinear fisher discrimination and neyman–pearson classification. *Neural Networks*, 16(5-6):529–535, 2003.

[25] Clayton Scott and Robert Nowak. A neyman-pearson approach to statistical learning. *IEEE Transactions on Information Theory*, 51(11):3806–3819, 2005.

[26] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[27] Harry L Van Trees. *Detection, estimation, and modulation theory, part I: detection, estimation, and linear modulation theory*. John Wiley & Sons, 2004.

[28] Alex J Bowers and Xiaoliang Zhou. Receiver operating characteristic (roc) area under the curve (auc): A diagnostic measure for evaluating the accuracy of predictors of education outcomes. *Journal of Education for Students Placed at Risk (JESPAR)*, 24(1):20–46, 2019.

[29] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[30] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[31] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[32] Bernhard Schölkopf. The kernel trick for distances. In *Advances in neural information processing systems*, pages 301–307, 2001.

[33] James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

[34] David de la Mata-Moya, María Pilar Jarabo-Amores, Jaime Martín de Nicolás, and Manuel Rosa-Zurera. Approximating the neyman–pearson detector with 2c-svms. application to radar detection. *Signal Processing*, 131:364–375, 2017.

[35] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Cost-sensitive boosting. *IEEE Transactions on pattern analysis and machine intelligence*, 33(2):294–309, 2010.

[36] Harris Drucker, Donghui Wu, and Vladimir N Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, 10(5):1048–1054, 1999.

[37] Mehrdad Mahdavi, Tianbao Yang, and Rong Jin. Stochastic convex optimization with multiple objectives. In *Advances in Neural Information Processing Systems*, pages 1115–1123, 2013.

[38] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[39] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[40] Manqi Zhao and Venkatesh Saligrama. Anomaly detection with score functions based on nearest neighbor graphs. In *Advances in neural information processing systems*, pages 2250–2258, 2009.

[41] Hirofumi Uzawa. Iterative methods for concave programming. *Studies in linear and nonlinear programming*, 6:154–165, 1958.

[42] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[43] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[44] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[45] Fang Feng, Kuan-Ching Li, Jun Shen, Qingguo Zhou, and Xuhui Yang. Using cost-sensitive learning and feature selection algorithms to improve the performance of imbalanced classification. *IEEE Access*, 8:69979–69996, 2020.

[46] Xiaoming Xi, Zhilou Yu, Zhaolei Zhan, Yilong Yin, and Cuihuan Tian. Multi-task cost-sensitive-convolutional neural network for car detection. *IEEE Access*, 7:98061–98068, 2019.

[47] Jiwen Lu, Venice Erin Liong, and Jie Zhou. Cost-sensitive local binary feature learning for facial age estimation. *IEEE Transactions on Image Processing*, 24(12):5356–5368, 2015.

[48] Jianwu Wan and Feng Zhu. Cost-sensitive canonical correlation analysis for semi-supervised multi-view learning. *IEEE Signal Processing Letters*, 27:1330–1334, 2020.

[49] Mingxia Liu, Linsong Miao, and Daoqiang Zhang. Two-stage cost-sensitive learning for software defect prediction. *IEEE Transactions on Reliability*, 63(2):676–686, 2014.

[50] Venkatesh Saligrama and Zhu Chen. Video anomaly detection based on local statistical aggregates. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2112–2119. IEEE, 2012.

[51] Huseyin Ozkan, Ozgun Soner Pelvan, and Suleyman S Kozat. Data imputation through the identification of local anomalies. *IEEE Transactions on Neural Networks and Learning Systems*, 26(10):2381–2395, 2015.

[52] Liangxiao Jiang, Chaoqun Li, and Shasha Wang. Cost-sensitive bayesian network classifiers. *Pattern Recognition Letters*, 45:211–216, 2014.

[53] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[54] Clayton D Scott and Robert D Nowak. Learning minimum volume sets. *Journal of Machine Learning Research*, 7:665–704, 2006.

[55] Alfred O Hero. Geometric entropy minimization (gem) for anomaly detection and localization. *Advances in Neural Information Processing Systems*, pages 585–592, 2007.

[56] Venkatesh Saligrama and Manqi Zhao. Local anomaly detection. *Artificial Intelligence and Statistics*, pages 969–983, 2012.

[57] Yuting Chen, Jing Qian, and Venkatesh Saligrama. A new one-class svm for anomaly detection. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3567–3571, 2013.

[58] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[59] Régis Vert and Jean-Philippe Vert. Consistency and convergence rates of one-class svms and related algorithms. *Journal of Machine Learning Research*, 7:817–854, 2006.

[60] Xuedan Miao, Ying Liu, Haiquan Zhao, and Chunguang Li. Distributed online one-class support vector machine for anomaly detection over networks. *IEEE Transactions on Cybernetics*, (99):1–14, 2018.

[61] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks. *IEEE International Conference on Advanced Information Networking and Applications Workshops*, pages 990–995, 2009.

[62] Roy L Streit. A neural network for optimum neyman-pearson classification. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 685–690. IEEE, 1990.

[63] Xin Tong. A plug-in approach to Neyman-Pearson classification. *Journal of Machine Learning Research*, 14:3011–3040, 2013.

[64] María-Pilar Jarabo-Amores, David De la Mata-Moya, Roberto Gil-Pita, and Manuel Rosa-Zurera. Radar detection with the Neyman–Pearson criterion using supervised-learning-machines trained with the cross-entropy error. *EURASIP Journal on Advances in Signal Processing*, 2013(1):44, 2013.

[65] Shuchen Kong, Weiwei Shen, Yingbin Zheng, Ao Zhang, Jian Pu, and Jun Wang. False positive rate control for positive unlabeled learning. *Neurocomputing*, 367:13–19, 2019.

[66] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[67] Cewu Lu, Jianping Shi, and Jiaya Jia. Abnormal event detection at 150 fps in matlab. *IEEE International Conference on Computer Vision*, pages 2720–2727, 2013.

[68] Huseyin Ozkan, Fatih Ozkan, and Suleyman S Kozat. Online anomaly detection under markov statistics with controllable type-i error. *IEEE Transactions on Signal Processing*, 64(6):1435–1445, 2015.

[69] Jing Lu, Steven CH Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. Large scale online kernel learning. *Journal of Machine Learning Research*, 17(1):1613–1655, 2016.

[70] Fatih Porikli and Huseyin Ozkan. Data driven frequency mapping for computationally scalable object detection. *IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 30–35, 2011.

[71] Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13(Oct):3103–3131, 2012.

[72] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. *Advances in Neural Information Processing Systems*, pages 342–350, 2009.

[73] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. *Advances in Neural Information Processing Systems*, pages 2627–2635, 2014.

[74] Kurt Cutajar, Edwin V Bonilla, Pietro Michiardi, and Maurizio Filippone. Random feature expansions for deep gaussian processes. *International Conference on Machine Learning*, pages 884–893, 2017.

[75] Siamak Mehrkanoon and Johan AK Suykens. Deep hybrid neural-kernel networks using random fourier features. *Neurocomputing*, 298:46–54, 2018.

[76] Jiaxuan Xie, Fanghui Liu, Kaijie Wang, and Xiaolin Huang. Deep kernel learning via random fourier features, 2019.

[77] Basarbatu Can, Mine Kerpicci, and Huseyin Ozkan. Online kernel-based nonlinear neyman-pearson classification. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1618–1622. IEEE, 2021.

[78] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[79] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. *Advances in Neural Information Processing Systems*, pages 476–484, 2012.

[80] Marthinus Du Plessis, Gang Niu, and Masashi Sugiyama. Convex formulation for learning from positive and unlabeled data. *International Conference on Machine Learning*, pages 1386–1394, 2015.

[81] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[82] Dheeru Dua and Casey Graff. Uci machine learning repository (2017). *URL http://archive. ics. uci. edu/ml*, 37, 2017.

[83] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011.

[84] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[85] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. *Large scale kernel machines*, 3(1):301–320, 2007.

[86] Xing Hu, Shiqiang Hu, Lingkun Luo, and Guoxiang Li. Abnormal event detection in crowded scenes via bag-of-atomic-events-based topic model. *Turkish Journal of Electrical Engineering & Computer Sciences*, 24(4):2638–2653, 2016.

[87] Weixin Luo, Wen Liu, Dongze Lian, Jinhui Tang, Lixin Duan, Xi Peng, and Shenghua Gao. Video anomaly detection with sparse coding inspired deep neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[88] Nesibe Yalcin, Gülay Tezel, and Cihan Karakuzu. Epilepsy diagnosis using artificial neural network learned by pso. *Turkish Journal of Electrical Engineering & Computer Sciences*, 23(2):421–432, 2015.

[89] Chunfeng Lian, Mingxia Liu, Jun Zhang, and Dinggang Shen. Hierarchical fully convolutional network for joint atrophy localization and alzheimer's disease diagnosis using structural mri. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):880–893, 2018.

[90] Morteza Arab and Mohammad Karim Sohrabi. Proposing a new clustering method to detect phishing websites. *Turkish Journal of Electrical Engineering & Computer Sciences*, 25(6):4757–4767, 2017.

[91] Noam Segev, Maayan Harel, Shie Mannor, Koby Crammer, and Ran El-Yaniv. Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1811–1824, 2016.

[92] Nouh Guidoum, Faouzi Soltani, and Amar Mezache. Two novel radar detectors for spiky sea clutter with the presence of thermal noise and interfering targets. *Turkish Journal of Electrical Engineering & Computer Sciences*, 28(3):1599–1611, 2020.

[93] Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems*, 23(8):1177–1193, 2012.

[94] Necati Demir and Gökhan Dalkiliç. Modified stacking ensemble approach to detect network intrusion. *Turkish Journal of Electrical Engineering & Computer Sciences*, 26(1):418–433, 2018.

[95] Bin Gu, Victor S Sheng, Keng Yeow Tay, Walter Romano, and Shuo Li. Cross validation through two-dimensional solution surface for cost-sensitive svm. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1103–1121, 2016.

[96] Basarbatu Can and Huseyin Ozkan. A neural network approach for online nonlinear neyman-pearson classification. *IEEE Access*, 8:210234–210250, 2020.

[97] Frans MJ Willems, Yuri M Shtarkov, and Tjalling J Tjalkens. The context-tree weighting method: Basic properties. *IEEE transactions on information theory*, 41(3):653–664, 1995.

[98] I Papageorgiou, Ioannis Kontoyiannis, L Mertzanis, A Panotopoulou, and M Skoularidou. Revisiting context-tree weighting for bayesian inference. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 2906–2911. IEEE, 2021.

[99] Fei Mi and Boi Faltings. Adaptive sequential recommendation using context trees. In *IJCAI*, pages 4018–4019, 2016.

[100] Marco Cuturi and Jean-Philippe Vert. The context-tree kernel for strings. *Neural Networks*, 18(8):1111–1123, 2005.

[101] Nikolaos Tziortziotis, Christos Dimitrakakis, and Konstantinos Blekas. Cover tree bayesian reinforcement learning. *Journal of Machine Learning Research*, 15:2313–2335, 2014.

[102] Suleyman S Kozat, Andrew C Singer, and Georg Christoph Zeitler. Universal piecewise linear prediction via context trees. *IEEE Transactions on Signal Processing*, 55(7):3730–3745, 2007.

[103] Huseyin Ozkan, N Denizcan Vanli, and Suleyman S Kozat. Online classification via self-organizing space partitioning. *IEEE Transactions on Signal Processing*, 64(15):3895–3908, 2016.

[104] Mohammadreza Mohaghegh Neyshabouri, Kaan Gokcesu, Hakan Gokcesu, Huseyin Ozkan, and Suleyman Serdar Kozat. Asymptotically optimal contextual bandit algorithm using hierarchical structures. *IEEE transactions on neural networks and learning systems*, 30(3):923–937, 2018.

[105] Mine Kerpicci, Huseyin Ozkan, and Suleyman Serdar Kozat. Online anomaly detection with bandwidth optimized hierarchical kernel density estimators. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[106] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[107] Burr Settles. Active learning literature survey. 2009.

[108] Yazhou Yang and Marco Loog. A benchmark and comparison of active learning for logistic regression. *Pattern Recognition*, 83:401–415, 2018.

[109] Fabrizio Carcillo, Yann-Aël Le Borgne, Olivier Caelen, and Gianluca Bontempi. Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization. *International Journal of Data Science and Analytics*, 5(4):285–300, 2018.

[110] Xiaojin Zhu. *Semi-supervised learning with graphs*. Carnegie Mellon University, 2005.

[111] Deng Cai and Xiaofei He. Manifold adaptive experimental design for text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):707–719, 2011.

[112] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[113] Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS workshop on cost-sensitive learning*, volume 1. Vancouver, CA:, 2008.

[114] Cheng Deng, Xianglong Liu, Chao Li, and Dacheng Tao. Active multi-kernel domain adaptation for hyperspectral image classification. *Pattern Recognition*, 77:306–315, 2018.

[115] Alim Samat, Jun Li, Sicong Liu, Peijun Du, Zelang Miao, and Jieqiong Luo. Improved hyperspectral image classification by active learning using pre-designed mixed pixels. *Pattern Recognition*, 51:43–58, 2016.

[116] Devis Tuia, Jordi Muñoz-Marí, and Gustavo Camps-Valls. Remote sensing image segmentation by active queries. *Pattern Recognition*, 45(6):2180–2192, 2012.

[117] Jian Wu, Victor S Sheng, Jing Zhang, Hua Li, Tetiana Dadakova, Christine Leon Swisher, Zhiming Cui, and Pengpeng Zhao. Multi-label active learning algorithms for image classification: Overview and future promise. *ACM Computing Surveys (CSUR)*, 53(2):1–35, 2020.

[118] Chenying Liu, Jun Li, and Lin He. Superpixel-based semisupervised active learning for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(1):357–370, 2018.

[119] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2016.

[120] Juan Mario Haut, Mercedes E Paoletti, Javier Plaza, Jun Li, and Antonio Plaza. Active learning with convolutional neural networks for hyperspectral image classification using a new bayesian approach. *IEEE Transactions on Geoscience and Remote Sensing*, 56(11):6440–6461, 2018.

[121] Xiangyong Cao, Jing Yao, Zongben Xu, and Deyu Meng. Hyperspectral image classification with convolutional neural network and active learning. *IEEE Transactions on Geoscience and Remote Sensing*, 58(7):4604–4616, 2020.

[122] Shayok Chakraborty, Vineeth Balasubramanian, and Sethuraman Panchanathan. Adaptive batch mode active learning. *IEEE transactions on neural networks and learning systems*, 26(8):1747–1760, 2014.

[123] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

[124] Yuhong Guo and Russell Greiner. Optimistic active-learning using mutual information. In *IJCAI*, volume 7, pages 823–829, 2007.

[125] Alex Holub, Pietro Perona, and Michael C Burl. Entropy-based active learning for object recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[126] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. int. conf. on machine learning, 2001.

[127] Andrew I Schein and Lyle H Ungar. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265, 2007.

[128] Kai Yu, Jinbo Bi, and Volker Tresp. Active learning via transductive experimental design. In *Proceedings of the 23rd international conference on Machine learning*, pages 1081–1088, 2006.

[129] Tong Zhang and F Oles. The value of unlabeled data for classification problems. In *Proceedings of the Seventeenth International Conference on Machine Learning,(Langley, P., ed.)*, volume 20, page 0. Citeseer, 2000.

[130] Yazhou Yang and Marco Loog. A variance maximization criterion for active learning. *Pattern Recognition*, 78:358–370, 2018.

[131] Wenbin Cai, Yexun Zhang, Ya Zhang, Siyuan Zhou, Wenquan Wang, Zhuoxiang Chen, and Chris Ding. Active learning for classification with maximum model change. *ACM Transactions on Information Systems (TOIS)*, 36(2):1–28, 2017.

[132] Alexander Freytag, Erik Rodner, and Joachim Denzler. Selecting influential examples: Active learning with expected model output changes. In *European conference on computer vision*, pages 562–577. Springer, 2014.

[133] Christoph Käding, Alexander Freytag, Erik Rodner, Andrea Perino, and Joachim Denzler. Large-scale active learning with approximations of expected model output changes. In *German Conference on Pattern Recognition*, pages 179–191. Springer, 2016.

[134] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. *Advances in neural information processing systems*, 20, 2007.

[135] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Semisupervised svm batch mode active learning with applications to image retrieval. *ACM Transactions on Information Systems (TOIS)*, 27(3):1–29, 2009.

[136] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. *Advances in neural information processing systems*, 23, 2010.

[137] Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems*, 27, 2014.

[138] Sebastian Raisch and Alexander Zimmermann. A process perspective on the exploration–exploitation paradox. *The Oxford handbook of organizational paradox*, page 315, 2017.

[139] Waleed Hilal, S Andrew Gadsden, and John Yawney. Financial fraud:: A review of anomaly detection techniques and recent advances. 2022.

[140] Kilian Hendrickx, Wannes Meert, Yves Mollet, Johan Gyselinck, Bram Cornelis, Konstantinos Gryllias, and Jesse Davis. A general anomaly detection framework for fleet-based condition monitoring of machines. *Mechanical Systems and Signal Processing*, 139:106585, 2020.

[141] Wen Xu, Julian Jang-Jaccard, Amardeep Singh, Yuanyuan Wei, and Fariza Sabrina. Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset. *IEEE Access*, 9:140136–140146, 2021.

[142] Elvan Duman and Osman Ayhan Erdem. Anomaly detection in videos using optical flow and convolutional autoencoder. *IEEE Access*, 7:183914–183923, 2019.

[143] Huseyin Ozkan. Performans garantili ve büyük veriye Ölçeklenebilir Çevrimiçi makine Öğrenimi anomali tespit yöntemlerinin geliştirilmesi ve bilgisayarlı görü geniş alan gözetlemesine uygulanması, 2022.

[144] Weixin Luo, Wen Liu, and Shenghua Gao. A revisit of sparse coding based anomaly detection in stacked rnn framework. In *Proceedings of the IEEE international conference on computer vision*, pages 341–349, 2017.

[145] Vijay Mahadevan, Weixin Li, Viral Bhalodia, and Nuno Vasconcelos. Anomaly detection in crowded scenes. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 1975–1981. IEEE, 2010.

[146] Yuqi Ouyang and Victor Sanchez. Video anomaly detection by estimating likelihood of representations. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 8984–8991. IEEE, 2021.

[147] Neha Gupta, Ole-Christoffer Granmo, and Ashok Agrawala. Thompson sampling for dynamic multi-armed bandits. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 1, pages 484–489. IEEE, 2011.

[148] Alexandra Carpentier, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, and Peter Auer. Upper-confidence-bound algorithms for active learning in multi-armed bandits. In *International Conference on Algorithmic Learning Theory*, pages 189–203. Springer, 2011.