Online Learning: A Comprehensive Survey

Steven C.H. Hoi*, Doyen Sahoo*, Jing Lu*, Peilin Zhao†
*School of Information Systems, Singapore Management University, Singapore
†School of Software Engineering, South China University of Technology, Guangzhou, China
{chhoi,doyens,jing.lu.2014}@smu.edu.sg,peilinzhao@hotmail.com

Abstract

Online learning represents an important family of machine learning algorithms, in which a learner attempts to resolve an online prediction (or any type of decision-making) task by learning a model/hypothesis from a sequence of data instances one at a time. The goal of online learning is to ensure that the online learner would make a sequence of accurate predictions (or correct decisions) given the knowledge of correct answers to previous prediction or learning tasks and possibly additional information. This is in contrast to many traditional *batch learning* or *offline machine learning* algorithms that are often designed to train a model in batch from a given collection of training data instances. This survey aims to provide a comprehensive survey of the online machine learning literatures through a systematic review of basic ideas and key principles and a proper categorization of different algorithms and techniques. Generally speaking, according to the learning type and the forms of feedback information, the existing online learning works can be classified into three major categories: (i) *supervised online learning* where full feedback information is always available, (ii) *online learning with limited feedback*, and (iii) *unsupervised online learning* where there is no feedback available. Due to space limitation, the survey will be mainly focused on the first category, but also briefly cover some basics of the other two categories. Finally, we also discuss some open issues and attempt to shed light on potential future research directions in this field.

Keywords: Online Learning, Online Convex Optimization, Large Scale Machine Learning

1. Introduction

Machine learning plays a crucial role in modern data analytics and emerging artificial intelligence (AI) applications. Traditional machine learning paradigms often work in a batch learning or offline learning fashion (especially for supervised learning), where a collection of data is given to train a model by some learning algorithm and then the model is deployed for inference without (or seldom) performing any updates afterwards. Such learning methods suffer from expensive re-training cost when dealing with new training data, and thus are poorly scalable for real-world applications. In the era of big data, traditional batch learning paradigms become more and more restricted, especially when live data increases and evolves rapidly. Making machine learning scalable and practical has become an open grand challenge in machine learning and AI.

Unlike traditional machine learning, *online learning* is a subfield of machine learning and includes an important family of learning techniques which are devised to learn models incrementally from data in a sequential manner. Online learning overcomes the drawbacks of traditional batch learning in that the model can be updated instantly and efficiently by an online learner when new training data arrives. Besides, online learning algorithms are often easy to understand, simple to implement, and often founded on solid theory with rigorous regret bounds. Along with urgent need of making machine learning practical for real big data analytics, online learning has attracted increasing interest in recent years.

This survey aims to give a comprehensive survey of *online learning*[1] literatures. Online learning has been extensively studied across different fields, ranging from machine learning, data mining, statistics, optimization and applied math, to artificial intelligence and data science. This survey aims to distill the core ideas of online learning methodologies and applications in litera-

[1]The term of "online learning" in this survey is *not* related to "e-learning" in the online education field.

ture. This survey is written mainly for machine learning audiences, and assumes readers with basic knowledge in machine learning. While trying our best to make the survey as comprehensive as possible, it is very difficult to cover every detail since online learning research has been evolving rapidly in recent years. We apologize in advance for any missing papers or inaccuracies in description, and encourage readers to provide feedback, comments or suggestions. Finally, as a supplemental document to this survey, readers may check our updated version online at: `http://libol.stevenhoi.org/survey`.

1.1. What is Online Learning?

Traditional machine learning paradigm often runs a batch learning fashion , e.g., a supervised learning task, where a collection of training data is given in advance to train a model by following some learning algorithm. Such paradigm requires the entire training data set made available prior to the learning task, and the training process is often done in an offline environment due to the expensive training cost. Traditional batch learning methods suffer from some critical drawbacks: (i) low efficiency in both time and space costs; and (ii) poor scalability for large-scale applications because the model often has to be re-trained from scratch for new training data.

In contrast to batch learning algorithms, online learning is a method of machine learning for data arriving in a sequential order, where a learner aims to learn and update the best predictor for future data at every step. Online learning is able to overcome the drawbacks of batch learning in that the predictive model can be updated instantly for any new data instances. Thus, online learning algorithms are far more efficient and scalable for large-scale machine learning tasks in real-world data analytics applications where data are not only large in size, but also arriving at a high velocity.

1.2. Tasks and Applications

Similar to traditional (batch) machine learning methods, online learning techniques can be applied to solve a variety of tasks in a wide range of real-world application domains. Examples of online learning tasks include the following:

Supervised learning tasks: Online learning algorithms can be derived for supervised learning tasks. One common supervised learning task is classification, a task of identifying to which of a set of categories a new data instance belongs to, on the basis of observing a training set of data instances whose category label is given. For example, one commonly studied task in online learning is binary classification which involves only two distinct categories; other types of supervised classification tasks include multi-class classification, multi-label classification, and multiple-instance classification, etc.

In addition to classification tasks, another common supervised learning task in machine learning is regression analysis, which refers to the learning process for estimating the relationships among variables (typically between a dependent variable and one or more independent variables). Online learning techniques are naturally applied for regression analysis tasks, e.g., time series analysis where data instances arrive sequentially.

Unsupervised learning tasks: Online learning algorithms can be applied to solve unsupervised learning tasks. One example case is clustering or cluster analysis — a process of grouping a set of objects such that objects in the same group ("cluster") are more similar to each other than to objects in other groups/clusters. Online clustering aims to perform incremental cluster analysis on a sequence of data instances, which is commonly explored in mining data streams.

Other learning tasks: Online learning techniques can also be used for other kinds of machine learning tasks, such as learning with recommender systems, or reinforcement learning. For example, the learning task with recommender systems aims to produce recommendations, typically through either collaborative or content-based filtering approaches. One family of widely used techniques for recommender systems is collaborative filtering, which is the process of filtering for information by exploiting the collaborations among users. Online learning techniques can be explored for such tasks to improve both efficacy and scalability performances.

Finally, it is important to note that online learning techniques are often used in two major application scenarios. One scenario is to improve efficiency and scalability of some existing machine learning methodology for regular batch machine learning tasks where a full collection of training data must be made available before the learning task. For example, Support Vector Machines (SVM) is a well-known machine learning method for batch classification tasks, in which classical SVM algorithms (e.g., QP or SMO solvers [1]) could suffer from poor scalability for large-scale applications. In literature, a variety of online learning algorithms have been investigated for training SVM in an online (or stochastic) learning manner [2, 3], making it more efficient and scalable than conventional batch SVMs. The other scenario is to apply online learning algorithms to directly tackle online analytics tasks where

data instances naturally arrive in a sequential manner and the target concepts may be drifting or evolving over time. Examples include time series regression, such as stock price prediction, where data arrives periodically and the learner has to make decisions immediately before getting the next instance.

1.3. Taxonomy

To help readers better understand the online learning literatures as a whole, we attempt to construct a taxonomy of online learning, as summarized in Figure 1. In general, from a theoretical perspective, online learning methodologies are founded based on theory and principles from three major theory communities: learning theory, optimization theory, and game theory. From the perspective of specific algorithms, we can further group the existing online learning techniques into different categories according to their specific learning principles and problem settings. Specifically, according to the types of feedback information and the types of supervision in the learning tasks, online learning techniques can be classified into the following three major categories:

- **Online supervised learning:** This is concerned with tasks where full feedback information is always revealed to a learner at the end of each online learning round. It can be further divided into three groups: (i) "'linear online learning" that aims to learn a linear predictive model from a sequence of training data instances; (ii) "nonlinear online learning" that aims to learn a nonlinear predictive model which is either based on kernel methods or other non-linear models; and (iii) non-traditional online learning that addresses other supervised online learning tasks which are different from traditional supervised learning models for classification and regression.

- **Online learning with limited feedback:** This is concerned with tasks where an online learner receives partial feedback information from the environment during the online learning process. For example, consider multi-class classification tasks, at a particular round, the learner makes a prediction of class label for an incoming instance, and then receives the partial feedback indicating whether the prediction is correct or not instead of the explicit class label. For such tasks, the online learner often has to make online predictions or decisions by achieving a tradeoff between the exploitation of disclosed knowledge and the exploration of unknown information.

- **Online unsupervised learning:** This is concerned with online learning tasks where the online learner only receives the sequence of data instances without any additional feedback (e.g., true class label) during the online learning tasks. Unsupervised online learning can be considered as a natural extension of traditional unsupervised learning for dealing with data streams, which is typically studied in batch learning fashion. Examples of unsupervised online learning include online clustering, online representation learning, and online anomaly detection tasks, etc. Unsupervised online learning has less restricted assumptions about data without requiring explicit feedback or label information which could be difficult or expensive to acquire.

This article will conduct a systematic review of existing works for online learning, especially for supervised online learning and online learning with partial feedback. Finally, we note that it is always very challenging to make a precise categorization of all the existing online learning works, and it is likely that the above proposed taxonomy may not fully cover all the existing online learning works in literature, though we have tried our best to cover as much as possible.

1.4. Related Work and Further Reading

This paper attempts to make a comprehensive survey of online learning research works. In literature, there are some related books, PHD theses, and articles published over the past years dedicated to online learning [4, 5], in which many of them also include rich discussions on related works of online learning. For example, the book titled "Prediction, Learning, and Games" [4] gave a nice introduction about some niche subjects of online learning, particularly for online prediction with expert advice and online learning with partial feedback. Another recent work titled "Online Learning and Online Convex Optimization" [5] gave a nice tutorial about basics of online learning and foundations of online convex optimization. In addition, there are also quite a few PHD theses dedicated to addressing different subjects of online learning [6, 7, 8, 9]. Readers are also encouraged to read some older related books, surveys and tutorial notes about online learning and online algorithms [10, 11, 12, 13, 14]. Finally, readers who are interested in applied online learning can explore some open-source toolboxes, including LIBOL [15, 16] and Vowpal Wabbit [17].

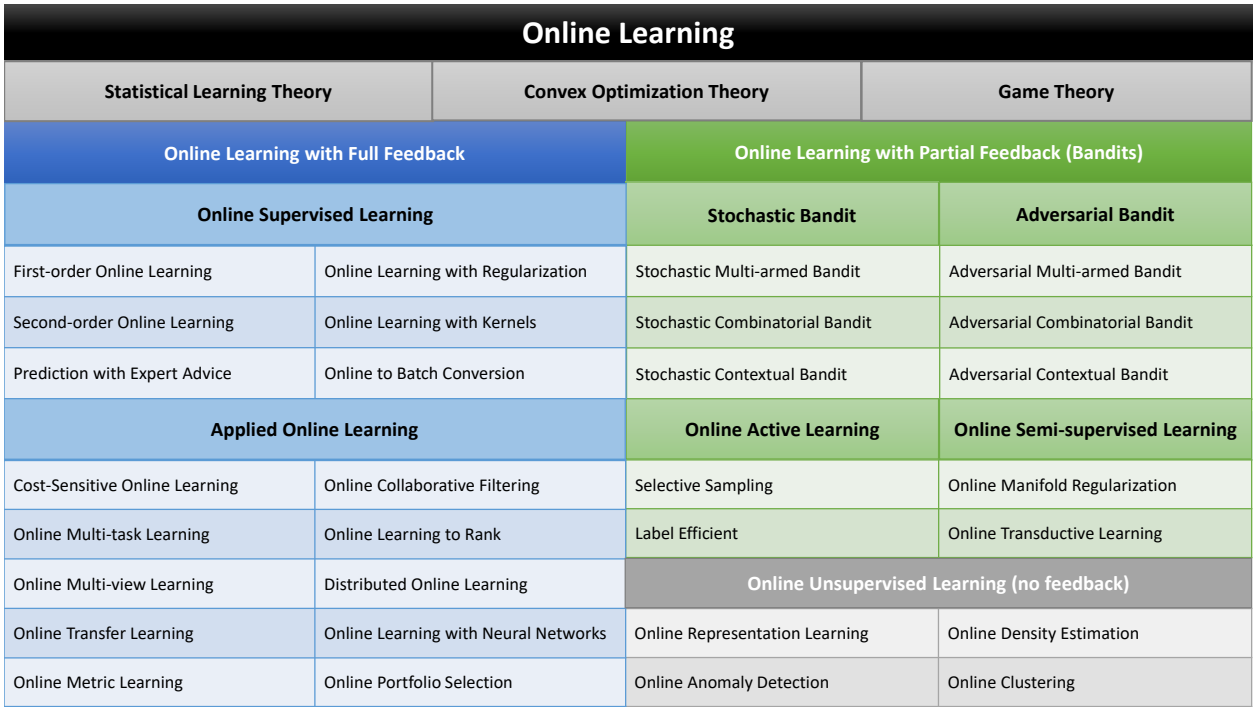

Figure 1: Taxonomy of Online Learning Techniques

2. Problem Formulations and Related Theory

Without loss of generality, we will first give a formal formulation of a classical online learning problem, i.e., binary online classification, and then introduce basics of online convex optimization as the theoretical foundations for many online learning techniques.

2.1. Problem Settings

Online learning takes place in a sequential way. On each round, a learner receives a data instance, and then makes a prediction of the instance, e.g., classifying it into some predefined categories. After making the prediction, the learner receives the true answer about the instance from the environment as a feedback. Based on the feedback, the learner can measure the loss suffered, depending on the difference between the prediction and the answer. Finally, the learner updates its prediction model by some strategy so as to improve predictive performance on future received instances.

Consider spam email detection as a running example of online binary classification, where the learner answers every question in binary: yes or no. The task is supervised binary classification from a machine learning perspective. More formally, we can formulate the problem as follows: consider a sequence of instances/objects represented in a vector space, $\mathbf{x}_t \in \mathbb{R}^d$, where t denotes the t-th round and d is the dimensionality, and we use $y_t \in \{+1, -1\}$ to denote the true class label of the instance. The online binary classification takes place sequentially. On the t-th round, an instance $\mathbf{x}_t$ is received by the learner, which then employs a binary classifier $\mathbf{w}_t$ to make a prediction on $\mathbf{x}_t$, e.g., $\hat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$ that outputs $\hat{y}_t = +1$ if $\mathbf{w}_t^\top \mathbf{x}_t \geq 0$ and $\hat{y}_t = -1$ otherwise. After making the prediction, the learner receives the true class label y_t and thus can measure the suffered loss (e.g. hinge-loss: $\ell_t(\mathbf{w}_t) = \max(0, 1 - y_t \mathbf{w}_t^\top \mathbf{x}_t)$). Whenever the loss is nonzero, the learner updates the prediction model from $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$ by some strategy on the training example $(\mathbf{x}_t, y_t)$. The procedure of Online Binary Classification is summarized in Algorithm 1.

Algorithm 1: Online Binary Classification process.

Initialize the prediction function as $\mathbf{w}_1$;
for $t = 1, 2, \ldots, T$ **do**
  Receive instance: $\mathbf{x}_t \in \mathbb{R}^d$;
  Predict $\widehat{y_t} = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$ as the label of $\mathbf{x}_t$;
  Receive correct label: $y_t \in \{-1, +1\}$;
  Suffer loss: $\ell_t(\mathbf{w}_t)$, which depends on the difference between $\mathbf{w}_t^\top \mathbf{x}_t$ and y_t;
  Update the prediction function $\mathbf{w}_t$ to $\mathbf{w}_{t+1}$;
end for

By running an online learner over T rounds, the regret of the learner is defined as

$$R_T = \sum_{t=1}^{T} \ell_t(\mathbf{w}_t) - \min_{\mathbf{w}} \sum_{t=1}^{T} \ell_t(\mathbf{w}) \quad (1)$$

Here, the second term is the loss suffered by the optimal model $\mathbf{w}^*$, which can be known only in hindsight. Similarly, the number of mistakes made by the online learner can be defined as

$$M_T = \sum_{t=1}^{T} \mathbb{I}(\widehat{y_t} \neq y_t)$$

The goal of online learning is to minimize the regret in the long run. More formally, a good online learning strategy should be able to guarantee low regret even in the worst case.

The methods to solve the described problem setting largely have their theoretical foundations in the fields of statistical learning theory, convex optimization theory, and game theory. Next, we give a brief overview of these topics.

2.2. Statistical Learning Theory

Statistical learning theory, first introduced in the late 1960's, is a powerful tool not only for the theoretical analysis of machine learning problems but also for creating practical algorithms for estimating multidimensional functions. In literature, there are many comprehensive survey articles and books [18, 19].

2.2.1. Empirical Error Minimization

Assuming that the instance $\mathbf{x}_t$ is generated from a fixed but unknown distribution $P(\mathbf{x})$ and the class label y is also generated randomly with a fixed but unknown distribution $P(y|\mathbf{x})$. The joint distribution of labeled data is $P(\mathbf{x}, y) = P(\mathbf{x})P(y|\mathbf{x})$. The goal of a learning problem is to find a prediction function $f(\mathbf{x})$ that minimizes the expected value of the loss function:

$$R(f) = \int \ell(y, f(\mathbf{x}))\mathrm{d}P(x, y)$$

which is also termed as the *Risk Function*. The solution $f^* = \arg\min R(f)$ is the optimal predictor.

In practice, we draw a group of instances $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$ for model training. Assuming that the training data are drawn i.i.d, we could then estimate the value of the risk function:

$$R_{emp}(f) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(\mathbf{x}_n))$$

which is usually termed as Empirical Error or Empirical Risk. The learning problem is to find a function f over a class of functions $\mathcal{F}$ and minimizes the Empirical Error:

$$\hat{f}_n = \arg\min_{f \in \mathcal{F}} R_{emp}(f)$$

The problem is called Empirical Error Minimization (ERM). ERM is the theoretical base for many machine learning algorithms. For example, in the binary classification problem with hinge loss, when $\mathcal{F}$ is the set of linear classifiers, we can rewrite the ERM as

$$R_{emp}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \max(0, 1 - y_n \mathbf{w}^\top \mathbf{x}_n)$$

2.2.2. Error Decomposition

A good model function $\hat{f}_n$ obtained from the training process should behave similarly with the optimal function f^*. We examine their performance difference by the Excess Risk:

$$R(\hat{f}_n) - R(f^*) = \left(R(\hat{f}_n) - \inf_{f \in \mathcal{F}} R(f)\right) + \left(\inf_{f \in \mathcal{F}} R(f) - R(f^*)\right)$$

where the first term, called the Estimation Error, is due to the fact that we only have finite training samples and the sampling is imperfect to approximate the expectation of loss function. While the second term, called the Approximation Error, is due to the restriction of model class $\mathcal{F}$.

In practice, the estimation error increases with the increase of model complexity while the approximation error performs just the opposite, i.e., decreases with the increase of model complexity. Consequently, choosing the complexity of $\mathcal{F}$ is a trade-off of approximation vs. estimation. Unfortunately, this is not an easy task.

2.3. Convex Optimization Theory

Many online learning problems can essentially be (re-)formulated as an Online Convex Optimization (OCO) task. In the following, we introduce some basics of online convex optimization.

An online convex optimization task typically consists of two major elements: a convex set $\mathcal{S}$ and a convex cost function $\ell_t(\cdot)$. At each time step t, the online algorithm decides to choose a weight vector $\mathbf{w}_t \in \mathcal{S}$; after that, it suffers a loss $\ell_t(\mathbf{w}_t)$, which is computed based on a convex cost function $\ell_t(\cdot)$ defined over $\mathcal{S}$. The goal of the online algorithm is to choose a sequence of decisions $\mathbf{w}_1, \mathbf{w}_2, \ldots$ such that the regret in hindsight can be minimized.

More formally, an online algorithm aims to achieve a low regret R_T after T rounds, where the regret R_T is defined as:

$$R_T = \sum_{t=1}^{T} \ell_t(\mathbf{w}_t) - \inf_{\mathbf{w}^* \in \mathcal{S}} \sum_{t=1}^{T} \ell_t(\mathbf{w}^*), \tag{2}$$

where $\mathbf{w}^*$ is the solution that minimizes the convex objective function $\sum_{t=1}^{T} \ell_t(\mathbf{w})$ over $\mathcal{S}$.

For example, consider an online binary classification task for training online Support Vector Machines (SVM) from a sequence of labeled instances $(\mathbf{x}_t, y_t), t = 1, \ldots, T$, where $\mathbf{x}_t \in \mathcal{R}^d$ and $y_t \times \{+1, -1\}$. One can define the loss function $\ell(\cdot)$ as $\ell_t(\mathbf{w}_t) = \max(0, 1 - y_t \mathbf{w}^\top \mathbf{x})$ and the convex set S as $\{\forall \mathbf{w} \in \mathcal{R}^d | \|\mathbf{w}\| \leq C\}$ for some constant parameter C. There are a variety of algorithms to solve this problem.

For a comprehensive treatment of this subject, readers are referred to the books in [5, 20]. Below we briefly review three major families of online convex optimization (OCO) methods, including first-order algorithms, second-order algorithms, and regularization based approaches.

2.3.1. First-order Methods

First order methods aim to optimize the objective function using the first order (sub) gradient information. Online Gradient Descent (OGD)[21] can be viewed as an online version of Stochastic Gradient Descent (SGD) in convex optimization, and is one of the simplest and most popular methods for convex optimization.

At every iteration, based on the loss suffered on instance $\mathbf{x}_t$, the algorithm takes a step from the current model to update to a new model, in the direction of the gradient of the current loss function. This update gives us $\mathbf{u} = \mathbf{w}_t - \eta_t \nabla \ell_t(\mathbf{w}_t)$. The resulting update may push the model to lie outside the feasible domain. Thus, the algorithm projects the model onto the feasible domain, i.e., $\Pi_{\mathcal{S}}(\mathbf{u}) = \arg\min_{\mathbf{w} \in \mathcal{S}} \|\mathbf{w} - \mathbf{u}\|$ (where $\Pi_{\mathcal{S}}$ denotes the projection operation). OGD is simple and easy to implement, but the projection step sometimes may be computationally intensive which depends on specific tasks. In theory [21], a simple OGD algorithm achieves sublinear regret $O(\sqrt{T})$ for an arbitrary sequence of T convex cost functions (of bounded gradients), with respect to the best single decision in hindsight.

2.3.2. Second-order Methods.

Second-order methods aim to exploit second order information to speed up the convergence of the optimization. A popular approach is the Online Newton Step Algorithm. The Online Newton Step [22] can be viewed as an online analogue of the Newton-Raphson method in batch optimization. Like OGD, ONS also performs an update by subtracting a vector from the current model in each online iteration. While the vector subtracted by OGD is the gradient of the current loss function based on the current model, in ONS the subtracted vector is the inverse Hessian multiplied by the gradient, i.e., $A_t^{-1} \nabla \ell_t(\mathbf{w}_t)$ where A_t is related to the Hessian. A_t is also updated in each iteration as $A_t = A_{t-1} + \nabla \ell_t(\mathbf{w}_t) \nabla \ell_t(\mathbf{w}_t)^\top$. The updated model is projected back to the feasible domain as $\mathbf{w}_{t+1} = \Pi_{\mathcal{S}}^{A_t}(\mathbf{w}_t - \eta A_t^{-1} \nabla \ell_t(\mathbf{w}_t))$, where $\Pi_{\mathcal{S}}^{A}(\mathbf{u}) = \arg\min_{\mathbf{w} \in \mathcal{S}} (\mathbf{w} - \mathbf{u})^\top A (\mathbf{w} - \mathbf{u})$. Different from OGD where the projection is made under the Euclidean norm, ONS projects under the norm induced by the matrix A_t. Although ONS's time complexity $O(n^2)$ is higher than OGD's $O(n)$, it guarantees a logarithmic regret $O(\log T)$ under relatively weaker assumptions of exp-concave cost functions.

2.3.3. Regularization

Unlike traditional convex optimization, the aim of Online Convex Optimization is to optimize the regret. Traditional approaches (termed as Follow the Leader (FTL)) can be unstable, leading to high regret (e.g. linear regret) in the worst case [20]. This motivates the need to stabilize the approaches through regularzation. Here we discuss the common regularization approaches.

Follow-the-Regularized-Leader (FTRL) The idea of Follow-the-Regularized-Leader (FTRL) [23, 24] is to stablize the prediction of the Follow-the-Leader (FTL) [25, 26] by adding a regularization term $R(\mathbf{w})$ which is strongly convex, smooth and twice differentiable. The idea is to solve the following optimization problem in each iteration:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in \mathcal{S}} \left[\eta \sum_{s=1}^{t} \nabla \ell_s(\mathbf{w}_s)^\top \mathbf{w} + R(\mathbf{w}) \right]$$

where $\mathcal{S}$ is the feasible convex set and η is the learning rate. In theory, the FTRL algorithm in general achieves a sublinear regret bound $O(\sqrt{T})$.

Online Mirror Descent (OMD). OMD is an online version of the Mirror Descent (MD) method [27, 28] in batch convex optimization. The OMD algorithm behaves like OGD, in that it updates the model using a simple gradient rule. However, it generalizes OGD as it performs updates in the dual space. This duality is induced by the choice of the regularizer: the gradient of the regularization serves as a mapping from $\mathbb{R}^d$ to itself. Due to this transformation by the regularizer, OMD is

able to obtain better bounds in terms of the geometry of the space.

In general, OMD has two variants of algorithms: lazy OMD and active OMD. The lazy version keeps track of a point in Euclidean space and projects it onto the convex feasible domain only when making prediction, while the active version keeps a feasible model all the time, which is a direct generalization of OGD. Unlike OGD, the projection step in OMD is based on the Bregman Divergence $\mathcal{B}_R$, i.e., $\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathcal{S}} \mathcal{B}_R(\mathbf{w}\|\mathbf{v}_{t+1})$, where $\mathbf{v}_{t+1}$ is the updated model after the gradient step. In general, the lazy OMD has the same regret bound as FTRL. The active OMD also has a similar regret bound. When $R(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$, OMD recovers OGD. If we use other functions as R, we can also recover some other interesting algorithms, such as the Exponential Gradient (EG) algorithm below.

Exponential Gradient (EG). Let $R(\mathbf{w}) = \mathbf{w}\ln\mathbf{w}$ be the negative entropy function and the feasible convex domain be the simplex $\mathcal{S} = \Delta_d = \{\mathbf{w} \in \mathbb{R}_+^d | \sum_i w_i = 1\}$, then OMD will recover the Exponential Gradient (EG) algorithm [29]. In this special case, the induced projection is the normalization by the $L1$ norm, which indicates

$$w_{t+1,i} = \frac{w_{t,i}\exp[-\eta(\nabla\ell_t(\mathbf{w}_t))_i]}{\sum_j w_{t,j}\exp[-\eta(\nabla\ell_t(\mathbf{w}_t))_j]}$$

As a special case of OMD, the regret of EG is bounded by $O(\sqrt{T})$.

Adaptive (Sub)-Gradient Methods. In the previous algorithms, the regularization function R is always fixed and data independent, during the whole learning process. Adaptive (Sub)-Gradient (AdaGrad) algorithm [30] is an algorithm that can be considered as online mirror descent with adaptive regularization, i.e., the regularization function R can change over time. The regularizer R at the t-th step, is actually the function $R(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_{A_t^{1/2}}^2 = \frac{1}{2}\mathbf{w}^\top A_t^{1/2}\mathbf{w}$, which is constructed from the (sub)-gradients received before (and including) the t-th step. In each iteration the model is updated as:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathcal{S}} \left\|\mathbf{w} - [\mathbf{w}_t - \eta A_t^{-\frac{1}{2}}\nabla\ell_t(\mathbf{w}_t)]\right\|_{A_t^{\frac{1}{2}}}^2$$

where A_t is updated as:

$$A_t = A_{t-1} + \nabla\ell_t(\mathbf{w}_t)\nabla\ell_t(\mathbf{w}_t)^\top$$

We also note that there are also other emerging online convex optimization methods, such as Online Convex Optimization with long term constraints [31], which assumes that the constraints are only required to be satisfied in long term, and Online ADMM [32] which is an online version for the Alternating Direction Method of Multipliers (ADMM) [33, 34] and is particularly suitable for distributed optimization applications. The RESCALEDEXP algorithm [35], proposed recently, does not use any prior knowledge about the loss functions and does not require the tuning of learning rate.

2.4. Game Theory

Game theory is closely related to online learning. Actually, an online prediction problem can be represented as a repeated game between the predictor and the environment [4]. Consider the online classification problem for example. During each iteration, the algorithm chooses one class from a finite number of classes and the environment chooses the cost vector (the true class label). As the environment is stable, i.e. not played by the adversary, the algorithm tries to perform as well as the best fixed strategy.

The game theory under the simplest assumptions, full feedback and stable environment, can be used to represent conventional online classification problem, while various settings in game theory are related to many other online learning problems. For example, the chosen cost vector by the environment might be partly observed by the predictor, or the environment might be operated by the adversary who tries to maximize the loss of the predictor. In this section, we will introduce the basic concepts and algorithms of the game theory which will facilitate our later discussion.

2.4.1. K-Person Normal Form Games

In a game, there are K players ($1 < K < \infty$) and a player $k \in \{1, ..., K\}$ has N_k possible actions to choose from. The players' actions can be represented by a vector $\mathbf{i} = (i_1, ..., i_K)$, where $i_k \in \{1, ..., N_k\}$ is the action of player k. The loss suffered by the player k is denoted by $\ell^k(\mathbf{i})$ since the loss is related to not only the action of player k but the action of all the other players. During each iteration of the game, each player tries to take actions that minimizes its own loss.

In a mixed strategy, we assume that the player k draws its action from a probability distribution $\mathbf{p}^k \in \mathbb{R}^{N_k}$. And the action of all the players is a random vector $\mathbf{I} = (I_1, ..., I_K)$. Thus, we can calculate the expected loss of player k as,

$$\mathbb{E}\ell^k(\mathbf{I}) = \sum_{i_1=1}^{N_1}\cdots\sum_{i_K=1}^{N_K} p_{i_1}^1 \times \cdots \times p_{i_K}^K \ell^k(i_1, ..., i_K)$$

2.4.2. Nash Equilibrium

A strategy of all players $\pi = \mathbf{p}^1 \times \cdots \times \mathbf{p}^K$ is a Nash equilibrium if for any new strategy π' defined by replacing the action distribution of any player k in π with any probability distribution $\mathbf{q}^k$, we have

$$\mathbb{E}\ell^k(\mathbf{I}_\pi) \leq \mathbb{E}\ell^k(\mathbf{I}_{\pi'})$$

This definition indicates that in a Nash Equilibrium, if all other player keeps the same strategy, a player can not achieve a lower loss by only changing its own strategy. Given others strategy, in a Nash Equilibrium, everyone gets its own optimal strategy. In a game, there may be more than one Nash Equilibrium depending on the structure of the game and the loss functions.

2.4.3. Two-person Zero-Sum Games

Zero-Sum means that for any action $\mathbf{i}$, the sum of losses of all players is zero, i.e.

$$\sum_{k=1}^{K} \ell^k(\mathbf{i}) = 0$$

This indicates that the game is purely competitive and a player's loss results in another player's gain. The zero sum game is usually seen in real world. For example, a shooter winning a score can also be viewed as the loss of a goalie. In research, zero sum game can represent online learning in adversary setting.

The simplest setting of zero-sum game is call the Two-person Zero-sum Game where a player only plays against one opponent, i.e. $K = 2$ and $\ell^1(\mathbf{i}) = -\ell^2(\mathbf{i})$ [36]. Player 1's loss is just player 2's gain. This indicates that we only need one matrix $A \in \mathbb{R}^{N_1 \times N_2}$ to store the gain of player 1 in all actions, where $A_{a,b}$ is the gain of player 1 when player 1 chooses action a and player 2 chooses action b and $-A_{a,b}$ is the gain of player 2.

Given the strategies of the two players, $\mathbf{p}^1$ and $\mathbf{p}^2$, the expected gain of player 1 is $\mathbf{p}^1 \cdot A\mathbf{p}^2$. Player 1 would like to maximize this term while player 2 would like to minimize it. Finally they reach the Nash Equilibrium $\pi = \mathbf{p}^1_* \times \mathbf{p}^2_*$ and the gain of player 1 is $V = \mathbf{p}^1_* \cdot A\mathbf{p}^2_*$. Note that a zero sum game can be unfair. In other words, we are not expecting $V = 0$.

Minimax optimal strategy is a randomized strategy that has the best guarantee on its expected gain, over choices of the opponent. In other words, player 1 plays the optimal strategy assuming that player 2 knows player 1 very well, i.e.

$$\max_{\mathbf{p}^1} \min_{\mathbf{p}^2} \mathbf{p}^1 \cdot A\mathbf{p}^2$$

where $\mathbf{p}^1$ and $\mathbf{p}^2$ are under the constraint of probability distribution vectors.

Theorem 1. *In a two-person zero-sum game, when two players both follow the minimax optimal strategy, they reach the same optimal value*

$$V = \max_{\mathbf{p}^1} \min_{\mathbf{p}^2} \mathbf{p}^1 \cdot A\mathbf{p}^2 = \min_{\mathbf{p}^2} \max_{\mathbf{p}^1} \mathbf{p}^1 \cdot A\mathbf{p}^2$$

Actually, a two-person zero-sum game has a unique game value V. And any pair of optimal strategies $\pi = \mathbf{p}^1_* \times \mathbf{p}^2_*$ that achieves the value $V = \mathbf{p}^1_* \cdot A\mathbf{p}^2_*$ is a Nash equilibrium.

2.4.4. General-Sum Games

In a general-sum game, the sum of the players' gain can be non-zero for some actions, which indicates that there are some strategies that benefit all the players. In this situation, a Nash Equilibrium is a stable pair of strategies which is optimal for any player as long as the other player does not change its behavior. Different from the zero-sum game, there is no unique game value V in a general-sum game.

3. Supervised Online Learning

3.1. Overview

In this section, we survey a family of "supervised online learning" algorithms which define the fundamental approaches and principles for online learning methods.

We first discuss linear online learning methods, where a target model is a linear function. More formally, consider an input domain $\mathcal{X}$ and an output domain $\mathcal{Y}$ for a learning task, we aim to learn a hypothesis $f : \mathcal{X} \mapsto \mathcal{Y}$, where the target model f is linear. For example, consider a typical linear binary classification task, our goal is to learn a linear classifier $f : \mathcal{X} \mapsto \{+1, -1\}$ as follows: $f(\mathbf{x}_t; \mathbf{w}) = sgn(\mathbf{w} \cdot \mathbf{x}_t)$, where $\mathcal{X}$ is typically a d-dimensional vector space $\mathbb{R}^d$, $\mathbf{w} \in \mathcal{X}$ is a weight vector specified for the classifier to be learned, and $sgn(z)$ is an indicator function that outputs +1 when $z > 0$ and -1 otherwise. We review two major types of linear online learning algorithms: first-order online learning and second-order online learning algorithms. Following this, we discuss Prediction with expert advice, and Online Learning with Regularization. This is followed by reviewing nonlinear online learning using kernel based methods. We discuss a variety of kernel-based online learning approaches, their computational challenges, and several approximation strategies for efficient learning. We end this section by discussing the theory for converting using online learning algorithms to learn a batch model that can generalize well.

3.2. First-order Online Learning

In the following, we survey a family of first-order linear online learning algorithms, which exploit the first order information of the model during learning process.

3.2.1. Perceptron

Perceptron [37, 38, 39] is the oldest algorithm for online learning. The running of the algorithm for online binary classification task is outlined in Algorithm 2

Algorithm 2: Perceptron

INIT: $\mathbf{w}_1 = 0$
for $t = 1, 2, \ldots, T$ **do**
  Given an incoming instance $\mathbf{x}_t$, predict $\hat{y}_t = f_t(\mathbf{x}_t) = sign(\mathbf{w}_t \cdot \mathbf{x}_t)$;
  Receive the true class label $y_t \in \{+1, -1\}$;
  if $\hat{y}_t \neq y_t$ **then**
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$;
  end if
end for

In theory, by assuming the data is separable with some margin γ, the Perceptron algorithm makes at most $(\frac{R}{\gamma})^2$ mistakes, where the margin γ is defined as $\gamma = \min_{t \in [T]} |\mathbf{x}_t \cdot \mathbf{w}^*|$ and R is a constant such that $\forall t \in [T], \|\mathbf{x}_t\| \leq R$. The larger the margin γ is, the tighter the mistake bound will be.

In literature, many variants of Perceptron algorithms have been proposed. One simple modification is the "normalized Perceptron" algorithm that differs only in the updating rule as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \frac{\mathbf{x}_t}{\|\mathbf{x}_t\|}$$

The mistake bound of the "normalized Perceptron" algorithm can be improved from $(\frac{R}{\gamma})^2$ to $(\frac{1}{\gamma})^2$ for the separable case due to the normalization effect.

3.2.2. Winnow

Unlike the Perceptron algorithm that uses additive updates, Winnow [40] employs multiplicative updates. The problem setting is slightly different from the Perceptron: $\mathcal{X} = \{0, 1\}^d$ and $y \in \{0, 1\}$. The goal is to learn a classifier $f(x_1, \ldots, x_n) = x_{i_1} \vee \ldots \vee x_{i_k}$ called monotone disjunction, where $i_k \in 1, \ldots, d$. The separating hyperplane for this classifier is given by $x_{i_1} + \ldots + x_{i_k}$. The Winnow algorithm is outlined in Algorithm 3.

The Winnow algorithm has a mistake bound $\alpha k(\log_\alpha \theta + 1) + n/\theta$ where $\alpha > 1$ and $\theta \geq 1/\alpha$ and the target function is a k-literal monotone disjunction.

Algorithm 3: Winnow

INIT: $\mathbf{w}_1 = \mathbf{1}^d$, constant $\alpha > 1$ (e.g.,$\alpha = 2$)
for $t = 1, 2, \ldots, T$ **do**
  Given an instance $\mathbf{x}_t$, predict $\hat{y}_t = \mathbb{I}_{\mathbf{w}_t \cdot \mathbf{x}_t \geq \theta}$ (outputs 1 if statement holds and 0 otherwise);
  Receive the true class label $y_t \in \{1, 0\}$;
  if $\hat{y}_t = 1, y_t = 0$ **then**
    set $w_i = 0$ for all $x_{t,i} = 1$ ("elimination" or "demotion"),
  end if
  if $\hat{y}_t = 0, y_t = 1$ **then**
    set $w_i = \alpha w_i$ for all $x_{t,i} = 1$ ("promotion").
  end if
end for

3.2.3. Passive-Aggressive Learning (PA)

This is a popular family of first-order online learning algorithms which generally follows the principle of margin-based learning [41]. Specifically, given an instance $\mathbf{x}_t$ at round t, PA formulates the updating optimization as follows:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 \quad s.t. \ \ell_t(\mathbf{w}) = 0 \tag{3}$$

where $\ell_t(\mathbf{w}) = \max(0, 1 - y_t \mathbf{w} \cdot \mathbf{x}_t)$ is the hinge loss. The above resulting update is passive whenever the hinge loss is zero, i.e., $\mathbf{w}_{t+1} = \mathbf{w}_t$ whenever $\ell = 0$. In contrast, whenever the loss is nonzero, the approach will force $\mathbf{w}_{t+1}$ aggressively to satisfy the constraint regardless of any step-size; the algorithm is thus named as "Passive-Aggressive" (PA) [41]. More specifically, PA ensures the updated classifier $\mathbf{w}_{t+1}$ should stay as close as to the previous classifier ("passiveness") and every incoming instance should be classified by the updated classifier correctly ("aggressiveness"). The regular PA algorithm assumes training data is always separable, which may not be true for noisy training data from real-world applications. To overcome the above limitation, two variants of PA relax the assumption as follows:

$$\begin{aligned}
&\text{PA} - \text{I} : \mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 + C\xi \\
&\text{subject to} \quad \ell_t(\mathbf{w}) \leq \xi \ \text{ and } \ \xi \geq 0 \\
&\text{PA} - \text{II} : \mathbf{w}_{t+1} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 + C\xi^2 \\
&\text{subject to} \quad \ell_t(\mathbf{w}) \leq \xi
\end{aligned} \tag{4}$$

where C is a positive parameter to balance the trade-off between "passiveness" (first regularization term) and "aggressiveness" (second slack-variable term). By solving the three optimization tasks, we can derive the

closed-form updating rules of three PA algorithms:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t y_t \mathbf{x}_t, \quad \tau_t = \begin{cases} \ell_t / \|\mathbf{x}_t\|^2 & \text{(PA)} \\ \min\{C, \ell_t / \|\mathbf{x}_t\|^2\} & \text{(PA-I)} \\ \frac{\ell_t}{\|\mathbf{x}_t\|^2 + \frac{1}{2C}} & \text{(PA-II)} \end{cases}$$

It is important to note a major difference between PA and Perceptron algorithms. Perceptron makes an update only when there is a classification mistake. However, PA algorithms aggressively make an update whenever the loss is nonzero (even if the classification is correct). In theory [41], PA algorithms have comparable mistake bounds as the Perceptron algorithms, but empirically PA algorithms often outperform Perceptron significantly. The PA algorithms are outlined in Algorithm 4.

Algorithm 4: Passive Aggressive Algorithms

INIT: $\mathbf{w}_1$, Aggressiveness Parameter C;
for $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t \in \mathbb{R}^d$, predict $\hat{y}_t$ using $\mathbf{w}_t$;
  Suffer loss $\ell_t(\mathbf{w}_t)$;
  Set $\tau = \begin{cases} \ell_t / \|\mathbf{x}_t\|^2 & \text{(PA)} \\ \min\{C, \ell_t / \|\mathbf{x}_t\|^2\} & \text{(PA-I)} \\ \frac{\ell_t}{\|\mathbf{x}_t\|^2 + \frac{1}{2C}} & \text{(PA-II)} \end{cases}$
  Update $\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t y_t \mathbf{x}_t$;
end for

3.2.4. Online Gradient Descent (OGD)

Many online learning problems can be formulated as an online convex optimization task, which can be solved by applying the OGD algorithm. Consider the online binary classification as an example, where we use the hinge loss function, i.e., $\ell_t(\mathbf{w}) = \max(0, 1 - y_t \mathbf{w} \cdot \mathbf{x}_t)$. By applying the OGD algorithm, we can derive the updating rule as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t y_t \mathbf{x}_t \tag{5}$$

Algorithm 5: Online Gradient Descent

INIT: $\mathbf{w}_1$, convex set $\mathcal{S}$, step size η_t;
for $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t \in \mathbb{R}^d$, predict $\hat{y}_t$ using $\mathbf{w}_t$;
  Suffer loss $\ell_t(\mathbf{w}_t)$;
  Update $\mathbf{w}_{t+1} = \Pi_{\mathcal{S}}(\mathbf{w}_t - \eta_t \nabla \ell_t(\mathbf{w}_t))$
end for

where η_t is the learning rate (or step size) parameter. The OGD algorithm is outlined in Algorithm 5, where any generic convex loss function can be used. $\Pi_{\mathcal{S}}$ is the projection function to constrain the updated model to lie in the feasible domain.

OGD and PA share similar updating rules but differ in that OGD often employs some predefined learning rate scheme while PA chooses the optimal learning rate τ_t at each round (but subject to a predefined cost parameter C). In literature, different OGD variants have been explored for online learning tasks to improve either theoretical bounds or practical issues, such as adaptive OGD [42], and mini-batch OGD [43], amongst others.

3.2.5. Other first-order algorithms

In literature, there are also some other first-order online learning algorithms, such as Approximate Large Margin Algorithms (ALMA) [44] which is a large margin variant of the p-norm Perceptron algorithm, and the Relaxed Online Maximum Margin Algorithm (ROMMA) [45]. Many of these algorithms often follow the principle of large margin learning. The metaGrad algorithm [46] tries to adapt the learning rate automatically for faster convergence.

3.3. Second-Order Online Learning

Unlike the first-order online learning algorithms that only exploit the first order derivative information of the gradient for the online optimization tasks, second-order online learning algorithms exploit both first-order and second-order information in order to accelerate the optimization convergence. Despite the better learning performance, second-order online learning algorithms often fall short in higher computational complexity. In the following we present a family of popular second-order online learning algorithms.

3.3.1. Second Order Perceptron (SOP)

SOP algorithm [47] is able to exploit certain geometrical properties of the data which are missed by the first-order algorithms.

For better understanding, we first introduce the whitened Perceptron algorithm, which strictly speaking, is not an online learning method. Assuming that the instances $\mathbf{x}_1, ..., \mathbf{x}_T$ are preliminarily available, we can get the correlation matrix $M = \sum_{t=1}^{T} \mathbf{x}_t \mathbf{x}_t^\top$. The whitened Perceptron algorithm is simply the standard Perceptron run on the transformed sequence $(M^{-1/2}\mathbf{x}_1, y_1), ..., (M^{-1/2}\mathbf{x}_T, y_T)$. By reducing the correlation matrix of the transformed instances, the whitened Perceptron algorithm can achieve significantly better mistake bound.

SOP can be viewed as an online variant of the whitened Perceptron algorithm. In online setting

the correlation matrix M can be approximated by the previously seen instances. SOP is outlined in Algorithm 6

Algorithm 6: SOP

INIT: $\mathbf{w}_1 = 0$, X_0=[], $\mathbf{v}_0 = 0$, $k = 1$
for $t = 1, 2, \ldots, T$ **do**
  Given an incoming instance $\mathbf{x}_t$, set $S_t = [X_{k-1}\ \mathbf{x}_t]$,
  predict $\hat{y}_t = f_t(\mathbf{x}_t) = sign(\mathbf{w}_t \cdot \mathbf{x}_t)$, where
  $\mathbf{w}_t = (aI_n + S_t S_t^\top)^{-1}\mathbf{v}_{k-1}$
  Receive the true class label $y_t \in \{+1, -1\}$;
  if $\hat{y}_t \neq y_t$ **then**
    $\mathbf{v}_k = \mathbf{v}_{k-1} + y_t\mathbf{x}_t$, $X_k = S_t$, $k = k + 1$.
  end if
end for

Here $a \in \mathbb{R}^+$ is a parameter that guarantees the existence of the matrix inverse.

3.3.2. Confidence Weighted Learning (CW)

The CW algorithm [48] is motivated by the following observation: the frequency of occurrence of different features may differ a lot in an online learning task. (For example) The parameters of binary features are only updated when the features occur. Thus, the frequent features typically receive more updates and are estimated more accurately compared to rare features. However, no distinction is made between these feature types in most online algorithms. This indicates that the lack of second order information about the frequency or confidence of the features can hurt the learning.

In the CW setting, we model the linear classifier with a Gaussian distribution, i.e., $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu} \in \mathbb{R}^d$ is the mean vector and $\Sigma \in \mathbb{R}^{d\times d}$ is the covariance matrix. When making a prediction, the prediction confidence $M = \mathbf{w} \cdot \mathbf{x}$ also follows a Gaussian distribution: $M \sim \mathcal{N}(\mu_M, \Sigma_M)$, where $\mu_M = \boldsymbol{\mu} \cdot \mathbf{x}$ and $\Sigma_M = \mathbf{x}^\top\Sigma\mathbf{x}$.

Similar to the PA update strategy, the update rule in round t can be obtained by solving the following convex optimization problem:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu}\in\mathbb{R}^d} \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma)\|\mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\right) \quad s.t.\ \Pr[y_t M_t \geq 0] \geq \eta \tag{6}$$

The objective function means that the new distribution should stay close to the previous distribution so that the classifier does not forget the information learnt from previous instances, where the distance between the two distributions is measured by the KL divergence. The constraint means that the new classifier should classify the new instance $\mathbf{x}_t$ correctly with probability higher than a predefined threshold parameter $\eta \in (0, 1)$.

Note that this is only the basic form of confidence weighted algorithms and has several drawbacks. 1) Similar to the hard margin PA algorithm, the constraint forces the new instance to be correctly classified, which makes this algorithm very sensitive to noise. 2) The constraint is in a probability form. It is easy to solve a problem with the constraint $g(\boldsymbol{\mu}_M, \Sigma_M) < 0$. However, a problem with a probability form constraint is only solvable when the distribution is known. Thus, this method faces difficulty in generalizing to other online learning tasks where the constraint does not follow a Gaussian distribution.

3.3.3. Adaptive Regularization of Weight Vectors (AROW)

AROW [49] is a variant of CW that is designed for non-separable data. This algorithm adopts the same Gaussian distribution assumption on classifier vector $\mathbf{w}$ while the optimization problem is different. By recasting the CW constraint as regularizers, the optimization problem can be formulated as:

$$\begin{aligned} C(\boldsymbol{\mu}, \Sigma) = \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma)\|\mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\right) \\ +\lambda_1\ell(y_t, \boldsymbol{\mu}\cdot\mathbf{x}_t) + \lambda_2\mathbf{x}_t^\top\Sigma\mathbf{x}_t \end{aligned} \tag{7}$$

where $\ell(y_t, \boldsymbol{\mu}\cdot\mathbf{x}_t) = (\max(0, 1 - y_t\boldsymbol{\mu}\cdot\mathbf{x}_t))^2$ is the squared-hinge loss. During each iteration, the update rule is obtained by solving the optimization problem:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu}\in\mathbb{R}^d}(C(\boldsymbol{\mu}, \Sigma))$$

which balances the three desires. First, the parameters should not change radically on each round, since the current parameters contain information about previous examples (first term). Second, the new mean parameters should predict the current example with low loss (second term). Finally, as we see more examples, our confidence in the parameters should generally grow (third term). λ_1 and λ_2 are two positive parameters that control the weight of the three desires.

Besides the robustness to noisy data, another important advantage of AROW is its ability to be easily generalized to other online learning tasks, such as Confidence Weighted Online Collaborative Filtering algorithm [50] and Second-Order Online Feature Selection [51].

3.3.4. Soft Confidence weighted Learning (SCW)

This is a variant of CW learning in order to deal with non-separable data [52, 53]. Different from AROW which directly adds loss and confidence regularization,

and thus loses the adaptive margin property, SCW exploits adaptive margin by assigning different margins for different instances via a probability formulation. Consequently, SCW tends to be more efficient and effective.

Specifically, the constraint of CW can be rewritten as $y_t(\boldsymbol{\mu} \cdot \mathbf{x}_t) \geq \phi \sqrt{\mathbf{x}_t^\top \Sigma \mathbf{x}_t}$. Thus, the loss function can be defined as:$\ell(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t)) = \max(0, \phi \sqrt{\mathbf{x}_t^\top \Sigma \mathbf{x}_t} - y_t(\boldsymbol{\mu} \cdot \mathbf{x}_t))$. The original CW optimization can be rewritten as:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu} \in \mathbb{R}^d} \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\right)$$
$$\text{subject to } \ell(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t)) = 0$$

Inspired by soft-margin PA variants, SCW generalized the CW into two soft-margin formulations:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu} \in \mathbb{R}^d} \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\right) + C\ell(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t))$$
$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu} \in \mathbb{R}^d} \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\right) + C\ell^2(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t))$$

where $C \in \mathbb{R}^+$ is a parameter controls the aggressiveness of this algorithm, similar to the C in PA algorithm. The two algorithms are termed "SCW-I" and "SCW-II".

3.3.5. Other second-order algorithms

The confidence weighted idea also works for other online learning tasks such as multi-class classification [54], active learning [55] and structured-prediction [56]. There are many other online learning algorithms that adopt second order information: IELLIP [57] assumes the objective classifier $\mathbf{w}$ lies in an ellipsoid and incrementally updates the ellipsoid based on the current received instance. Other approaches include New variant of Adaptive Regularization (NAROW) [58] and the Normal Herding method via Gaussian Herding (NHERD) [59]. Recently, Sketched Online Newton [60] made significant improvements to speed-up second order online learning.

3.4. Prediction with Expert Advice

This is an important online learning subject [61] with many applications. A general setting is as follows. A learner has N experts to choose from, denoted by integers $1, \ldots, N$. At each time step t, the learner decides on a distribution $\mathbf{p}_t$ over the experts, where $p_{t,i} \geq 0$ is the weight of each expert i, and $\sum_{i=1}^N p_{t,i} = 1$. Each expert i then suffers some loss $\ell_{t,i}$ according to the environment. The overall loss suffered by the learner is $\sum_{i=1}^N p_{t,i}\ell_{t,i} = \mathbf{p}_t^\top \ell_t$, i.e., the weighted average loss of the experts with respect to the distribution chosen by the learner.

Typically we assume that the loss suffered by any expert is bounded. More specifically, it is assumed that $\ell_{t,i} \in [0, 1]$ without loss of generality. Besides this condition, no assumptions will be made on the form of the loss, or about the how they are generated. Suppose the cumulative losses experienced by each expert and the forecaster are calculated respectively as follows:

$$L_{t,i} = \sum_{s=1}^t \ell_{s,i}, \quad L_t = \sum_{s=1}^t \mathbf{p}_t^\top \ell_t.$$

The loss difference between the forecaster and the expert is known as the "regret", i.e.,

$$R_{t,i} = L_t - L_{t,i}, \quad i = 1, \ldots, N.$$

The goal of learning the forecaster is to make the regret with respect to each expert as small as possible, which is equivalent to minimizing the overall regret, i.e.,

$$R_T = \max_{1 \leq i \leq N} R_{T,i} = L_T - \min_{1 \leq i \leq N} L_{T,i}$$

In general, online prediction with expert advice is to design an ideal forecaster that can achieve a vanishing per-round regret, a property known as the Hannan consistency [62], i.e.,

$$R_T = o(T) \Leftrightarrow \lim_{T \to \infty} \frac{1}{T}\left(L_T - \min_{1 \leq i \leq N} L_{T,i}\right)$$

A learner achieving this is called a Hannan consistent forecaster [63].

3.4.1. Weighted Majority Algorithms

The weighted majority algorithm (WM) is a simple but effective algorithm that makes a binary prediction based on a series of expert advices [64, 65]. The simplest version is shown in Algorithm 7, where $\beta \in (0, 1)$ is a user specified discount rate parameter.

Algorithm 7: Weighted Majority

INIT: Initialize the weights $p_1, p_2, ...p_N$ of all experts to $1/N$.
for $t = 1, 2, \ldots, T$ **do**
  Get the prediction $x_1, ..., x_N$ from N experts.
  Output 1 if $\sum_{i:x_i=1} p_i \geq \sum_{i:x_i=0} p_i$ otherwise output 0.
  receive the true value. If the i-th expert made a mistake, $p_i = p_i * \beta$
end for

3.4.2. Randomized Multiplicative Weights Algorithms

This algorithm works under the same assumption that the expert advices are all binary [66]. While the prediction is random. The algorithm gives the prediction 1 with probability of $\gamma = \frac{\sum_{i:x_i=1} p_i}{\sum_i^N p_i}$ and 0 with probability of $1 - \gamma$.

3.4.3. Hedge Algorithm

One of the well-known approaches for online prediction with expert advice is the Hedge algorithm [67], which can be viewed as a direct generalization of Littlestone and Warmuth's weighted majority algorithm [64, 65]. The working of Hedge algorithm is shown in Algorithm 8

Algorithm 8: Hedge Algorithm

INIT: $\beta \in [0, 1]$, initial weight vector $\mathbf{w}_1 \in [0, 1]^N$ with $\sum_{i=1}^N w_{1,i} = 1$
for $t = 1, 2, \ldots, T$ **do**
  set distribution $\mathbf{p}_t = \frac{\mathbf{w}_t}{\sum_{i=1}^N w_{t,i}}$;
  Receive loss $\ell_t \in [0, 1]^N$ from environment;
  Suffer loss $\mathbf{p}_t^\top \ell_t$;
  Update the new weight vector to $w_{t+1,i} = w_{t,i}\beta^{\ell_{t,i}}$
end for

The algorithm maintains a weight vector whose value at time t is denoted $\mathbf{w}_t = (w_{t,1}, \ldots, w_{t,N})$. At all times, all weights are nonnegative. All of the weights of the initial weight vector $\mathbf{w}_1$ must be nonnegative and sum to one, which can be considered as a prior over the set of experts. If it is believed that one expert performs the best, it is better to assign it the most weight. If no prior is known, it is better to set all the initial weights equally, i.e., $w_{1,i} = 1/N$ for all i. The algorithm uses the normalized distribution to make prediction, i.e., $\mathbf{p}_t = \mathbf{w}_t / \sum_{i=1}^N w_{t,i}$. After the loss ℓ_t is disclosed, the weight vector $\mathbf{w}_t$ is updated using a multiplicative rule $w_{t+1,i} = w_{t,i}\beta^{\ell_{t,i}}, \quad \beta \in [0, 1]$, which implies that the weight of expert i will exponentially decrease with the loss $\ell_{t,i}$. In theory, the Hedge algorithm is proved to be Hannan consistent.

3.4.4. Other Algorithms

Besides Hedge, there are some other algorithms for online prediction with expert advice under more challenging settings, including exponentially weighted average forecaster (EWAF) and Greedy Forecaster (GF) [63]. We will mainly discuss EWAF, which is shown in Algorithm 9

Algorithm 9: EWAF

INIT: a poll of experts $f_i, \quad i = 1, \ldots, N$ and $L_{0,i} = 0, \ i = 1, \ldots, N$, and learning rate η
for $t = 1, 2, \ldots, T$ **do**
  The environment chooses the next outcome y_t and the expert advice $\{f_{t,i}\}$;
  The expert advice is revealed to the forecaster
  The forecaster chooses the prediction $\hat{p}_t = \frac{\sum_{i=1}^N \exp(-\eta L_{t-1,i}) f_{t,i}}{\sum_{i=1}^N \exp(-\eta L_{t-1,i})}$
  The environment reveals the outcome y_t;
  The forecaster incurs loss $\ell(\hat{p}_t, y_t)$ and;
  Each expert incurs loss $\ell(f_{t,i}, y_t)$
  The forecaster update the cumulative loss $L_{t,i} = L_{t-1,i} + \ell(f_{t,i}, y_t)$
end for

The difference between EWAF and Hedge is that the loss in Hedge is the inner product between the distribution and the loss suffered by each expert, while for EWAF, the loss is between the prediction and the true label, which can be much more complex.

3.5. Online Learning with Regularization

Traditional online learning methods learn a classifier $\mathbf{w} \in \mathbb{R}^d$ where the magnitude of each element $|\mathbf{w}^j|$ weights the importance of each feature, which are often non-zero. When dealing with high dimensional data, traditional online learning methods suffer from expensive computational time and space costs. This drawback is often addressed using regularization by performing Sparse online learning, which aims to exploit the sparsity property with real-world high-dimensional data. Specifically, a batch sparse learning problem can be formalized as:

$$P(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell_t(\mathbf{w}) + \phi_s(\mathbf{w})$$

where ϕ_s is a sparsity-inducing regularizer. For example, when choosing $\phi_s = \lambda \|\mathbf{w}\|_0$, it is equivalent to imposing a hard constraint on the number of nonzero elements in $\mathbf{w}$. Instead of choosing ℓ_0-norm which is hard to be optimized, a more commonly used regularizer is ℓ_1-norm, i.e., $\phi_s = \lambda \|\mathbf{w}\|_1$, which can induce sparsity of the weight vector but does not explicitly constrain the number of nonzero elements. The following reviews some popular sparse online learning methods.

3.5.1. Truncated Gradient Descent

A straightforward idea to sparse online learning is to modify Online Gradient Descent and round small coef-

ficients of the weight vector to 0 after every K iterations:

$$\mathbf{w}_{t+1} = T_0(\mathbf{w}_t - \eta \nabla \ell_t(\mathbf{w}_t), \theta)$$

where the function $T_0(\mathbf{v}, \theta)$ performs an element-wise rounding on the input vector: if the j-th element v^j is smaller than the threshold θ, set $v^j = 0$. Despite its simplicity, this method struggles to provide satisfactory performance because the aggressive rounding strategy may ignore many useful weights which may be very small due to low frequency of appearance.

Motivated by addressing the above limitation, the Truncated Gradient Descent (TGD) method [68] explores a less aggressive version of the truncation function:

$$\mathbf{w}_{t+1} = T_1(\mathbf{w}_t - \eta \nabla \ell_t(\mathbf{w}_t), \eta g_i, \theta)$$

$$\text{where } T_1(v^j, \alpha, \theta) = \begin{cases} \max(0, v^j - \alpha) & \text{if } v^j \in [0, \theta] \\ \min(0, v^j + \alpha) & \text{if } v^j \in [-\theta, 0] \\ v^j & \text{otherwise} \end{cases}$$

where $g_i > 0$ is a parameter that controls the level of aggressiveness of the truncation. By exploiting sparsity, TGD achieves efficient time and space complexity that is linear with respect to the number of nonzero features and independent of the dimensionality d. In addition, it is proven to enjoy a regret bound of $O(\sqrt{T})$ for convex loss functions when setting $\eta = O(1/\sqrt{T})$.

3.5.2. Forward Looking Subgradients (FOBOS)

Consider the objective function in the t-th iteration of a sparse online learning task as $\ell_t(\mathbf{w}) + r(\mathbf{w})$, FOBOS [69] assumes f_t is a convex loss function (differentiable), and r is a sparsity-inducing regularizer (non-differentiable). FOBOS updates the classifier in the following two steps:

(1) Perform Online Gradient Descent: $\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \eta_t \nabla \ell_t(\mathbf{w}_t)$

(2) Project the solution in (i) such that the projection stays close to the interim vector $\mathbf{w}_{t+\frac{1}{2}}$ and (ii) has a low complexity due to r:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \{\frac{1}{2}\|\mathbf{w} - \mathbf{w}_{\frac{1}{2}}\|^2 + \eta_{t+\frac{1}{2}} r(\mathbf{w})\}$$

When choosing ℓ_1-norm as the regularizer, the above optimization can be solved with the closed-form solution for each coordinate:

$$w^j_{t+1} = \text{sgn}(w^j_{t+\frac{1}{2}}) \left[|w^j_{t+\frac{1}{2}}| - \eta_{t+\frac{1}{2}} \right]_+$$

The FOBOS algorithm with ℓ_1-norm regularizer can be viewed as a special case of TGD, where the truncation threshold $\theta = \infty$, and the truncation frequency $K = 1$. When $\eta_{t+\frac{1}{2}} = \eta_{t+1}$ and $\eta_t = O(1/\sqrt{t})$, this algorithm also achieves $O(\sqrt{T})$ regret bound.

3.5.3. Regularized Dual Averaging (RDA)

Motivated by the theory of dual-averaging techniques [70], the RDA algorithm [71] updates the classifier by:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \left\{ \bar{\mathbf{g}}_t^\top \mathbf{w} + \Psi(\mathbf{w}) + \frac{\beta_t}{t} h(\mathbf{w}) \right\}$$

where $\Psi(\mathbf{w})$ is the original sparsity-inducing regularizer, i.e., $\Psi(\mathbf{w}) = \lambda \|\mathbf{w}\|_1$; $h(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$ is an auxiliary strongly convex function and $\bar{\mathbf{g}}_t$ is the averaged gradients of all previous iterations, i.e., $\bar{\mathbf{g}} = \frac{1}{t}\sum_{\tau=1}^{t} \nabla \ell_\tau(\mathbf{w}_\tau)$. Setting the step size $\beta_t = \gamma\sqrt{t}$, one can derive the closed-form solution:

$$w^j_{t+1} = \begin{cases} 0 & \text{if } |\bar{g}_t{}^j| < \lambda \\ -\frac{\sqrt{t}}{\gamma}(\bar{g}_t{}^j - \lambda \text{sgn}(\bar{g}_t{}^j)) & \text{otherwise} \end{cases}$$

To further pinpoint the differences between RDA and FOBOS, we rewrite FOBOS in the same notation as RDA:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \left\{ \mathbf{g}_t^\top \mathbf{w} + \Psi(\mathbf{w}) + \frac{1}{2\alpha_t}\|\mathbf{w} - \mathbf{w}_t\|_2^2 \right\}$$

Specifically, RDA differs from FOBOS in several aspects. First, RDA uses the averaged gradient instead of the current gradient. Second, $h(\mathbf{w})$ is a global proximal function instead of its local Bregman divergence. Third, the coefficient for $h(\mathbf{w})$ is $\beta_t/t = \gamma/\sqrt{t}$ which is $1/\alpha_t = O(\sqrt{t})$ in FOBOS. Fourth, the truncation of RDA is a constant λ, while the truncation in FOBOS $\eta_{t+\frac{1}{2}}$ decrease with a factor $\sqrt{t}$. Clearly, RDA uses a more aggressive truncation threshold, thus usually generates significantly more sparse solutions. RDA also ensures the $O(\sqrt{T})$ regret bound.

3.5.4. Adaptive Regularization

One major issue with both FOBOS and RDA is that the auxiliary strongly convex function $h(\mathbf{w})$ may not fully exploit the geometry information of underlying data distribution. Instead of choosing $h(\mathbf{w})$ as an ℓ_2-norm $\frac{1}{2}\|\mathbf{w}\|^2$ in RDA or a Mahalanobis norm $\|\cdot\|_{H_t}$ in FOBOS, [72] proposed a data-driven adaptive regularization for $h(\mathbf{w})$, i.e.,

$$h_t(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top H_t \mathbf{w}$$

where $H_t = (\sum_{\tau=1}^{t} \mathbf{g}_\tau \mathbf{g}_\tau^\top)^{\frac{1}{2}}$ accumulates the second order info from the previous instances over time. Replacing

the previous $h(\mathbf{w})$ in both RDA and FOBOS by the temporal adaptation function $h_t(\mathbf{w})$, [72] derived two generalized algorithms (Ada-RDA and Ada-FOBOS) with the solutions as follows respectively.

Ada-RDA:

$$w_{t+1}^j = \begin{cases} 0 & \text{if } |\bar{g}_t{}^j| < \lambda \\ -\frac{t}{\beta H_{t,jj}}(\bar{g}_t{}^j - \lambda \text{sgn}(\bar{g}_t{}^j)) & \text{otherwise} \end{cases}$$

Ada-FOBOS:

$$w_{t+1}^j = \text{sgn}(w_t^i - \frac{\alpha_t}{H_{t,jj}} g_t^j) \left[|w_t^i - \frac{\alpha_t}{H_{t,jj}} g_t^j| - \frac{\alpha_t \lambda}{H_{t,ii}} \right]_+ \quad (8)$$

In the above, H_t is approximated by a diagonal matrix since computing the root of a matrix is computationally impractical in high-dimensional data.

3.5.5. Online Feature Selection

Online feature selection [73, 74, 75, 76] is closely related to sparse online learning in that they both aim to learn an efficient classifier for very high dimensional data. However, the sparse learning algorithms aim to minimize the ℓ-1 regularized loss, while the feature selection algorithms are motivated to explicitly address the feature selection issue and thus impose a hard constraint on the number of non-zero elements in classifier. Because of these similarities, they share some common strategies such as truncation and projection.

3.5.6. Others

Two stochastic methods were proposed in [77] for ℓ_1-regularized loss minimization. The Stochastic Coordinate Descent (SCD) algorithm randomly selects one coordinate from d dimensions and update this single coordinate with the gradient of the total loss of all instances. The Stochastic Mirror Descent Made Sparse (SMIDAS) algorithm combines the idea of truncating the gradient with mirror descent algorithm, i.e., truncation is performed on the vector in dual space. The disadvantage of the two algorithms is that their computational complexity depends on the dimensionality d. Besides, the two algorithms are designed in batch learning setting, i.e., they assume all instances are known prior to the learning task. Besides, there are also some recent sparse online learning algorithms proposed [78, 79], which combine the ideas of sparse learning, second order online learning, and cost-sensitive classification together to make the online algorithms scalable for high-dimensional class-imbalanced learning tasks.

3.6. Online Learning with Kernels

We now survey a family of "Nonlinear Online Learning" algorithms for learning a nonlinear target function, where the nonlinearity is induced by kernels. We take the typical nonlinear binary classification task as an example. Our goal is to learn a nonlinear classifier $f : \mathbb{R}^d \to \mathbb{R}$ from a sequence of labeled instances $(\mathbf{x}_t, y_t), t = 1, ..., T$, where $\mathbf{x}_t \in \mathbb{R}^d$ and $y_t \in \{+1, -1\}$. We build the classification rule as: $\widehat{y_t} = \text{sgn}(f(\mathbf{x}_t))$, where $\widehat{y_t}$ is the predicted class label. We measure the classification confidence of certain instance $\mathbf{x}_t$ by $|f(\mathbf{x}_t)|$. Similar to the linear case, for an online classification task, one can define the hinge loss function $\ell(\cdot)$ for the t-th instance using the classifier at the t-th iteration:

$$\ell((\mathbf{x}_t, y_t); f_t) = \max(0, 1 - y_t f_t(\mathbf{x}_t))$$

Formally speaking, an online nonlinear learner aims to achieve the lowest regret $R(T)$ after time T, where the regret function $R(T)$ is defined as follows:

$$R(T) = \sum_{t=1}^{T} \ell_t(f_t) - \inf_f \sum_{t=1}^{T} \ell_t(f), \quad (9)$$

where $\ell_t(\cdot)$ is the loss for the classification of instance $(\mathbf{x}_t, y_t)$, which is short for $\ell((\mathbf{x}_t, y_t); \cdot)$. We denote by f^* the optimal solution of the second term, i.e., $f^* = \arg\min_f \sum_{t=1}^{T} \ell_t(f)$

In the following, we first introduce online kernel methods and then survey a family of scalable online kernel learning algorithms organized into two major categories: (i) budget online kernel learning using *budget maintenance* strategies and (ii) budget online kernel learning using *functional approximation* strategies. Then we briefly introduce some approaches for online learning multiple kernels. Without loss of generality, we will adopt the above online binary classification setting for the discussions in this section.

3.6.1. Online Kernel Methods

We refer to the output f of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \to \mathbb{R}\}$. Here $\mathcal{H}$ a Reproducing Kernel Hilbert Space (**RKHS**) endowed with a kernel function $\kappa(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ [18] implementing the inner product$\langle \cdot, \cdot \rangle$ such that: 1) κ has the reproducing property $\langle f, \kappa(\mathbf{x}, \cdot) \rangle = f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^d$; 2) $\mathcal{H}$ is the closure of the span of all $\kappa(\mathbf{x}, \cdot)$ with $\mathbf{x} \in \mathbb{R}^d$, that is, $\kappa(\mathbf{x}, \cdot) \in \mathcal{H}$ $\forall \mathbf{x} \in \mathcal{X}$. The inner product $\langle \cdot, \cdot \rangle$ induces a norm on $f \in \mathcal{H}$ in the usual way: $\|f\|_{\mathcal{H}} := \langle f, f \rangle^{\frac{1}{2}}$. To make it clear, we denote by $\mathcal{H}_\kappa$ an RKHS with explicit dependence on kernel κ. Throughout the analysis,

we assume $\kappa(\mathbf{x}, \mathbf{x}) \leq X^2$, $\forall \mathbf{x} \in \mathbb{R}^d$, where $X \in \mathbb{R}^+$ is a constant.

The goal of training a batch SVM classifier $f(\mathbf{x})$ is formulated as the following optimization:

$$\min_{f \in \mathcal{H}_\kappa} \frac{\lambda}{2}\|f\|^2_{\mathcal{H}} + \frac{1}{T}\sum_{t=1}^{T} \ell(f(\mathbf{x}_t); y_t) \tag{10}$$

where $\lambda > 0$ is a regularization parameter used to control model complexity. According to the Representer Theorem [80], the optimal solution of the above convex optimization problem lies in the span of T kernels, i.e., those centered on the training points. Consequently, the goal of a typical online kernel learning algorithm is to learn the kernel-based predictive model $f(\mathbf{x})$ for classifying a new instance $\mathbf{x} \in \mathbb{R}^d$ as follows: $f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \kappa(\mathbf{x}_t, \mathbf{x})$, where T is the number of processed instances, α_t denotes the coefficient of the t-th instances, and $\kappa(\cdot, \cdot)$ denotes the kernel function. We define support vector (SV) as the instance whose coefficient α is nonzero. Thus, we rewrite the previous classifier as $f(\mathbf{x}) = \sum_{i \in \mathcal{SV}} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$, where $\mathcal{SV}$ is the set of SV's and i is its index. We use the notation $|\mathcal{SV}|$ to denote the SV set size.

In literature, different online kernel methods have been proposed. We begin by introducing the simplest one, that is, the kernelized Perceptron algorithm.

Kernelized Perceptron. Running the Perceptron algorithm using a kernel based model gives us the kernelized perceptron. The Kernelized Perceptron algorithm [81] is outlined in Algorithm 10

Algorithm 10: Kernelized Perceptron

INIT: $f_0 = 0$
for $t = 1, 2, \ldots, T$ **do**
  Given an incoming instance $\mathbf{x}_t$, predict
  $\hat{y}_t = \text{sgn}(f_t(\mathbf{x}_t))$;
  Receive the true class label $y_t \in \{+1, -1\}$;
  if $\hat{y}_t \neq y_t$ **then**
    $\mathcal{SV}_{t+1} = \mathcal{SV}_t \cup (\mathbf{x}_t, y_t)$, $f_{t+1} = f_t + y_t \kappa(\mathbf{x}_t, \cdot)$;
  end if
end for

The algorithm works similarly to the linear Perceptron algorithm, except that the inner product in the linear classifier, i.e., $f_t(\mathbf{x}_t) = \sum_i \alpha_i \mathbf{x}_i^\top \mathbf{x}_t$, is replaced by a kernel function in the kernel Percetron.

Kernelized OGD. The OGD algorithm can be extended with kernels [82], as shown in Algorithm 11. Here, $\eta_t > 0$ is the learning rate parameter, and ℓ'_t is used to denote the derivative of loss function with respect to the classification score $f_t(\mathbf{x}_t)$.

Algorithm 11: Kernelized OGD

INIT: $f_0 = 0$
for $t = 1, 2, \ldots, T$ **do**
  Given an incoming instance $\mathbf{x}_t$, predict
  $\hat{y}_t = \text{sgn}(f_t(\mathbf{x}_t))$;
  Receive the true class label $y_t \in \{+1, -1\}$;
  if $\ell_t(f_t) > 0$ **then**
    $\mathcal{SV}_{t+1} = \mathcal{SV}_t \cup (\mathbf{x}_t, y_t)$,
    $f_{t+1} = f_t - \eta_t \nabla \ell_t(f_t(\mathbf{x}_t)) = f_t - \eta_t \ell'_t \kappa(\mathbf{x}_t, \cdot)$;
  end if
end for

Others. The kernel trick implies that the inner product between any two instances can be replaced by a kernel function, i.e., $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j), \forall i, j$, where $\Phi(\mathbf{x}_t) \in R^D$ denotes the feature mapping from the original space to a new D-dimensional space which can be infinite. Using the kernel trick, many existing linear online learning algorithms can be easily extended to the kernelized variants, such as the kernelized Perceptron and kernelized OGD as well as the kernel PA variants [41]. However, some algorithms that use complex update rules are non-trivial to be converted into kernelized versions, such as the Confidence Weighted algorithms [48]. Finally, some online kernel learning methods also attempt to make more effective updates at each iteration. For example, the Double Updating Online Learning (DUOL) [83, 84, 85] improves the efficacy of traditional online kernel learning methods by not only updating the weight of the newly added SV, but also the weight for one existing SV.

Next, we discuss budgeting techniques to speed-up the computation of kernel-based online learning.

3.6.2. Scalable Online Kernel Learning via Budget Maintenance

The key advantage of online kernel learning is the ability of solving linearly non-separable tasks. Despite enjoying better performance over linear models, online kernel learning falls short in a critical drawback, that is, the growing unbounded number of support vectors with increasing computational and space complexity over time — a challenge termed as "Curse of Kernelization" [86]. To address this challenge, a family of algorithms, termed "budget online kernel learning", have been proposed to bound the number of SV's with

a fixed budget $B = |\mathcal{SV}|$ using diverse budget maintenance strategies whenever the budget overflows. The general framework for budgeting strategies is shown in Algorithm 12. Most existing budget online kernel learning methods maintain the budget by three strategies: (i) SV Removal, (ii) SV Projection, and (iii) SV Merging. Next, we briefly review these three categories.

Algorithm 12: Budget Online Kernel Learning

INIT: $f_0 = 0$
for $t = 1, 2, \ldots, T$ **do**
  Given an incoming instance $\mathbf{x}_t$, predict $\hat{y}_t = \text{sgn}(f_t(\mathbf{x}_t))$;
  Receive the true class label $y_t \in \{+1, -1\}$;
  if update is needed **then**
    update the classifier from f_t to $f_{t+\frac{1}{2}}$ and $\mathcal{SV}_{t+\frac{1}{2}} = \mathcal{SV}_t \cup (\mathbf{x}_t, y_t)$
  end if
  if $|\mathcal{SV}_{t+\frac{1}{2}}| > B$ **then**
    Update Support Vector Set from $\mathcal{SV}_{t+\frac{1}{2}}$ to $\mathcal{SV}_{t+1}$ such that $|\mathcal{SV}_{t+1}| = B$
    Update the classifier from $f_{t+\frac{1}{2}}$ to f_{t+1}
  end if
end for

SV Removal. This strategy maintains the budget by a simple and efficient way: 1) update the classifier by adding a new SV whenever necessary (depending on the prediction mistake/loss); 2) if the SV size exceeds the budget, discard one existing SV and update the classifier accordingly.

To achieve this, we need to address the following concerns: (i) how to update the classifier and (ii) how to choose one existing SV for removal. The first step depends on which online learning method is used. For example, the update is based on the Perceptron algorithm in RBP [87], Forgetron [88], and Budget Perceptron [89], the OGD algorithm is adopted as the update step for BOGD [90] and BSGD+ removal [86], while PA is used for performing update in BPA-S [91].

The second step of SV removal, is to decide which existing SV $(\mathbf{x}_{del}, y_t)$ for removal in order to reduce the impact of the resulting classifier. One simple way is to randomly discard one existing SV uniformly with probability $\frac{1}{B}$, as adopted by RBP [87] and BOGD [90]. Besides, instead of choosing randomly, another way as used in "Forgetron" [88] is to discard the oldest SV by assuming an older SV is less representative for the distribution of fresh training data streams. Despite enjoying the merits of simplicity and high efficiency, these methods are often too simple to achieve satisfactory learning accuracy results.

To optimize the performance, some approaches have tried to perform exhaustive search in deciding the best SV for removal. For instance, the Budget Perceptron algorithm [89] searches for one SV that is classified with high confidence by the classifier:

$$y_{del}(f_{t+\frac{1}{2}}(\mathbf{x}_{del}) - \alpha_{del}\kappa(\mathbf{x}_{del}, \mathbf{x}_{del})) > \beta$$

where $\beta > 0$ is a fixed tolerance parameter. BPA-S shares the similar idea of exhaustive search. For every $r \in [B]$, a candidate classifier $f^r = f_{t+\frac{1}{2}} - \alpha_r\kappa(\mathbf{x}_r, \cdot)$ is generated by discarding the r-th SV from $f_{t+\frac{1}{2}}$. By comparing the B candidate classifiers, the algorithm selects the one that minimizes the current objective function of PA:

$$f_{t+1} = \underset{r\in[B]}{\text{argmin}} \frac{1}{2}\|f^r - f_t\|^2_{\mathcal{H}} + C\ell_t(f^r)$$

where $C > 0$ is the regularization parameter of PA that balances aggressiveness and passiveness.

Comparing the principles of different SV removal strategies, we observe that a simple rule may not always generate satisfactory accuracy, while an exhaustive search often incurs non-trivial computational overhead, which again may limit the application to large-scale problems. When deploying a solution in practice, one would need to balance the trade-off between effectiveness and efficiency.

SV Projection. SV Projection strategy first appeared in [92] where two new algorithms, Projectron and Projectron++, were proposed, which significantly outperformed the previous SV removal based algorithms such RBP and Forgetron. The SV projection method follows the setting of SV removal and identifies a support vector for removal during the update of the model. It then chooses a subset of $\mathcal{SV}$ as the projection base, which will be denoted by $\mathcal{P}$. Following this, a linear combination of kernels in $\mathcal{P}$ is used to approximate the removed SV. The procedure of finding the optimal linear combination can be formulated as a convex optimization of minimizing the projection error:

$$\beta = \underset{\beta\in\mathbb{R}^{|\mathcal{P}|}}{\text{argmin}} E_{proj} = \underset{\beta\in\mathbb{R}^{|\mathcal{P}|}}{\text{argmin}} \|\alpha_{del}\kappa(\mathbf{x}_{del}, \cdot) - \sum_{i\in\mathcal{P}} \beta_i\kappa(\mathbf{x}_i, \cdot)\|^2_{\mathcal{H}}$$

Finally, the classifier is updated by combining this linear combination with the original classifier:

$$f_{t+1} = f_{t+\frac{1}{2}} - \alpha_{del}\kappa(\mathbf{x}_{del}, \cdot) + \sum_{i\in\mathcal{P}} \beta_i\kappa(\mathbf{x}_i, \cdot)$$

There are several algorithms adopting the projection strategy, for example Projectron, Projectron++, BPA-P, BPA-NN [91] and BSGD+Project [86]. These methods differ in a few aspects. First, the update rules are based on different online learning algorithms. Generally speaking, PA based and OGD based tend to outperform Perceptron based algorithms because of their effective update. Second, the choice of discarded SV is different. Since projection itself is relative slow, exhaustive search based algorithms (BPA-NN, BPA-P) are extremely time consuming. Thus algorithms with simple selecting rules are prefered (Projectron, Projectron++, BSGD+Project). Third, the choice of projection base set $\mathcal{P}$ is different. In Projectron, Projectron++, BPA-P and BSGD+Project, the discarded SV is projected onto the whole SV set, i.e. $\mathcal{P} = \mathcal{SV}$. While in BPA-NN, $\mathcal{P}$ is only a small subset of $\mathcal{SV}$, made up of the nearest neighbors of the discarded SV $(\mathbf{x}_{del}, y_{del})$. In general, a larger projection base set implies a more complicated optimization problem and thus more time costs. The research direction of SV projection based budget learning is to find a proper way of selecting $\mathcal{P}$ so that the algorithm achieves the minimized projection error with a relative small projection base set.

SV Merging. In [86], a SV merging method BSGD+Merge was proposed that attempts to replace the sum of two SV's $\alpha_m\kappa(\mathbf{x}_m, \cdot) + \alpha_n\kappa(\mathbf{x}_n, \cdot)$ by a newly created support vector $\alpha_z\kappa(\mathbf{z}, \cdot)$, where α_m, α_n and α_z are the corresponding coefficients of $\mathbf{x}_m$, $\mathbf{x}_n$ and $\mathbf{z}$. Following the previous discussion, the goal of online budget learning through SV merging strategy is to find the optimal $\alpha_z \in \mathbb{R}$ and $\mathbf{z} \in \mathbb{R}^d$ that minimizes the gap between f_{t+1} and $f_{t+\frac{1}{2}}$.

As it is relatively complicated to optimize the two terms simultaneously, this optimization is divided into two steps. First, assuming the coefficient of $\mathbf{z}$ is $\alpha_m + \alpha_n$, this algorithm tries to create the optimal support vector that minimizes the merging error. The first optimization is

$$\min_{\mathbf{z}} \|(\alpha_m + \alpha_n)\kappa(\mathbf{z}, \cdot) - (\alpha_m\kappa(\mathbf{x}_m, \cdot) + \alpha_n\kappa(\mathbf{x}_n, \cdot))\|$$

The solution is $\mathbf{z} = h\mathbf{x}_m + (1-h)x_n$, where $0 < h < 1$ is a real number that can be found by a line search method. This solution indicates that the optimal created SV lies on the line connecting $\mathbf{x}_m$ and $\mathbf{x}_n$. After obtaining the optimal created SV $\mathbf{z}$, the next step is to find the optimal coefficient α_z, which can be formulated as

$$\min_{\alpha_z} \|(\alpha_z\kappa(\mathbf{z}, \cdot) - (\alpha_m\kappa(\mathbf{x}_m, \cdot) + \alpha_n\kappa(\mathbf{x}_n, \cdot))\|.$$

The solution becomes $\alpha_z = \alpha_m\kappa(\mathbf{x}_m, \mathbf{z}) + \alpha_n\kappa(\mathbf{x}_n, \mathbf{z})$. The remaining problem is how to select the two SV's $\mathbf{x}_m$ and $\mathbf{x}_n$ for merging. The ideal solution is to find the pair with the minimal merging error through an exhaustive search, which however often requires $O(B^2)$ time complexity. How to perform exhaustive search for efficient SV merging remains an open challenge.

Summary. Among the various algorithms of budget online kernel learning using the idea of budget maintenance, the key differences are the updating rules and budget maintenance strategies used by different methods. Table 1 gives a summary of different algorithms and their properties.

In addition to the previous budget kernel learning algorithms there are still many representative works in online kernel learning field. Some [95, 96] introduce the sparse kernel idea to reduce the number of SV's in the online-to-batch-conversion problem, where an online algorithm is used to train a model in the batch setting (See Section 3.7).

3.6.3. Scalable Online Kernel Learning via Functional Approximation

In contrast to the previous budget online kernel learning methods using budget maintenance strategies to guarantee efficiency and scalability, another emerging and promising strategy is to explore functional approximation techniques for achieving scalable online kernel learning [97, 98].

The key idea is to construct a kernel-induced feature representation $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^D$ such that the inner product of instances in the new feature space can effectively approximate the kernel function:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}_j)$$

Using the above approximation, the predictive model with kernels can be rewritten as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{B} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}) \approx \sum_{i=1}^{B} \alpha_i \mathbf{z}(\mathbf{x}_i)^\top \mathbf{z}(\mathbf{x}) = \mathbf{w}^\top \mathbf{z}(\mathbf{x})$$

where $\mathbf{w} = \sum_{i=1}^{B} \alpha_i \mathbf{z}(\mathbf{x}_i)$ denotes the weight vector to be learned in the new feature space.

As a consequence, solving a regular online kernel classification task can be turned into a linear online classification task on the new feature space derived from the kernel approximation. For example, the methods of online kernel learning with kernel approximation in [97, 98] integrate some existing online learning algorithms (e.g., OGD) with kernel approximation techniques [99, 100, 101] to derive scalable online kernel

Table 1: Comparisons of different budget online kernel learning algorithms.

Algorithms	Update Strategy	Budget Strategy	Update Time	Space
Stoptron [92]	Perceptron	Stop	$O(1)$	$O(B)$
Tighter Perceptron [93]	Perceptron	Removal	$O(B^2)$	$O(B)$
Tightest Perceptron [94]	Perceptron	Removal	$O(B^2)$	$O(B)$
Budget Perceptron [89]	Perceptron	Removal	$O(B^2)$	$O(B)$
RBP [87]	Perceptron	Removal	$O(B)$	$O(B)$
Forgetron[88]	Perceptron	Removal	$O(B)$	$O(B)$
BOGD[90]	OGD	Removal	$O(B)$	$O(B)$
BPA-S [91]	PA	Removal	$O(B)$	$O(B)$
BSGD+removal [86]	OGD	Removal	$O(B)$	$O(B)$
Projectron [92]	Perceptron	Projection	$O(B^2)$	$O(B^2)$
Projectron++ [92]	Perceptron	Projection	$O(B^2)$	$O(B^2)$
BPA-P [91]	PA	Projection	$O(B^3)$	$O(B^2)$
BPA-NN [91]	PA	Projection	$O(B)$	$O(B)$
BSGD+projection [86]	OGD	Projection	$O(B^2)$	$O(B^2)$
BSGD+merging [86]	OGD	Merging	$O(B)$	$O(B)$

learning algorithms, including Fourier Online Gradient Descent (FOGD) that explores random Fourier features for kernel functional approximation [102], and Nyström Online Gradient Descent (NOGD) that explores Nyström low-rank matrix approximation methods for approximating large-scale kernel matrix [103]. A recent work, Dual Space Gradient Descent [104, 105] updates the model as the RBP algorithm, but also builds an FOGD model using the discarded SV's. The final prediction is the combination of the two models.

3.6.4. Online Multiple Kernel Learning

Traditional online kernel methods usually assume a predefined good kernel is given prior to the online learning task. Such approaches could be restricted since it is often hard to choose a good kernel prior to the learning task. To overcome the drawback, Online Multiple Kernel Learning (OMKL) aims to combining multiple kernels automatically for online learning tasks without fixing any predefined kernel. In the following, we begin by introducing some basics of batch Multiple Kernel Learning (MKL) [106].

Given a training set $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$ where $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \{-1, +1\}$, and a set of m kernel functions $\mathcal{K} = \{\kappa_i : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, i = 1, \ldots, m\}$. MKL learns a kernel-based prediction function by identifying an optimal combination of the m kernels, denoted by $\theta = (\theta_1, \ldots, \theta_m)$, to minimize the margin-based classification error, which can be cast into the optimization below:

$$\min_{\theta \in \Delta} \min_{f \in \mathcal{H}_{K(\theta)}} \frac{1}{2}|f|^2_{\mathcal{H}_{K(\theta)}} + C \sum_{t=1}^{n} \ell(f(\mathbf{x}_t), y_t) \tag{11}$$

where $\Delta = \{\theta \in \mathbb{R}^m_+ | \theta^\top \mathbf{1}_m = 1\}$, $K(\theta)(\cdot, \cdot) = \sum_{i=1}^m \theta_i \kappa_i(\cdot, \cdot)$, $\ell(f(\mathbf{x}_t), y_t) = \max(0, 1 - y_t f(\mathbf{x}_t))$. In the above formulation, we use notation $\mathbf{1}_T$ to represent a vector of T dimensions with all its elements being 1. It can also be cast into the following mini-max optimization problem:

$$\min_{\theta \in \Delta} \max_{\alpha \in \Xi} \left\{ \alpha^\top \mathbf{1}_T - \frac{1}{2}(\alpha \circ \mathbf{y})^\top \left(\sum_{i=1}^{m} \theta_i K^i \right) (\alpha \circ \mathbf{y}) \right\} \tag{12}$$

where $K^i \in \mathbb{R}^{T \times T}$ with $K^i_{j,l} = \kappa_i(\mathbf{x}_j, \mathbf{x}_l)$, $\Xi = \{\alpha | \alpha \in [0, C]^T\}$, and $\circ$ defines the element-wise product between two vectors. The above batch MKL optimization has been extensively studied [107, 108], but obtaining efficient solutions remains an open challenge.

Some efforts of online MKL studies [109, 110] have attempted to solve batch MKL optimization via online learning. Unlike these approaches that are mainly concerned in optimizing the optimal kernel combination as regular MKL, another framework of *Online Multiple Kernel Learning* (OMKL) in [111, 112, 113] is focused on exploring effective online combination of multiple kernel classifiers via a significantly more efficient and scalable way. Specifically, the OMKL in [111, 112] learns a kernel-based prediction function by selecting a subset of predefined kernel functions in an online learning fashion, which is in general more challenging than typical online learning because both the kernel classifiers and the subset of selected kernels are unknown, and more importantly the solutions to the kernel classifiers and their combination weights are correlated. [112] proposed novel algorithms based on the fusion of two types of online learning algorithms, i.e., the *Perceptron* algo-

rithm that learns a classifier for a given kernel, and the *Hedge* algorithm [67] that combines classifiers by linear weights. Some stochastic selection strategies were also proposed by randomly selecting a subset of kernels for combination and model updating to further improve the efficiency. These methods were later extended for regression [114], learning from data with time-sensitive patterns [115] and imbalanced data streams[116]. In addition, there have been budgeting approaches to make OMKL scalable [117].

3.7. Online to Batch Conversion

As introduced before, an online learning algorithm $\mathcal{A}$ is a sequential paradigm in which at each round, the algorithm predicts a vector $\mathbf{w}_t \in \mathcal{S} \subseteq \mathbb{R}^d$, nature responds with a convex loss function ℓ_t, and the algorithm suffers loss $\ell_t(\mathbf{w}_t)$. In this setting, the goal of the algorithm is to minimize the regret:

$$\mathrm{Reg}_{\mathcal{A}}(T) = \sum_{t=1}^{T} \ell_t(\mathbf{w}_t) - \min_{\mathbf{w}\in\mathcal{S}} \sum_{t=1}^{T} \ell_t(\mathbf{w}).$$

Obviously, the regret of $\mathcal{A}$ is the difference between its cumulative loss and the cumulative loss of the optimal fixed vector.

Usually, the sequence of loss functions will depend on a sequence of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$, for which there are few assumptions. Specifically, the loss $\ell_t(\mathbf{w})$ can be also expressed as $\ell(\mathbf{w}; (\mathbf{x}_t, y_t))$, so that we can rewrite the previous regret bound as

$$\mathrm{Reg}_{\mathcal{A}}(T) = \sum_{t=1}^{T} \ell(\mathbf{w}_t; (\mathbf{x}_t, y_t)) - \min_{\mathbf{w}\in\mathcal{S}} \sum_{t=1}^{T} \ell(\mathbf{w}; (\mathbf{x}_t, y_t))$$

However, for batch setting, we are more interested in finding a parameter $\hat{\mathbf{w}}$ with good generalization ability, i.e., we would like

$$R(\hat{\mathbf{w}}) - \min_{\mathbf{w}\in\mathcal{S}} R(\mathbf{w})$$

to be small, where the generalization risk is $R(\mathbf{w}) = \mathrm{E}_{(\mathbf{x},y)}[\ell(w; (x, y))]$, and $(\mathbf{x}, y)$ satisfies a fixed unknown distribution.

So, we would like to study the generalization performance of online algorithms via Online to Batch Conversion [118], which is the conversion relate the regret of the online algorithm to its generalization performance.

3.7.1. A General Conversion Theory

In this subsection, we will consider the generalization ability of online learning under the situation that the loss function $\ell(\mathbf{w}; (\mathbf{x}, y))$ is strongly convex. This assumption is reasonable, since some loss functions are really strongly convex, such as, squared loss, and even if some loss function is not strongly convex, like hinge loss, we can add a regularization term, such as $\frac{1}{2}\|\cdot\|$, during the learning tasks, to achieve strong convexity. In addition, we denote the dual norm of $\|\cdot\|$ as $\|\cdot\|_*$, where $\|\mathbf{v}\|_* = \sup_{\|\mathbf{w}\|\le 1} \mathbf{v}^\top \mathbf{w}$. Let $Z = (\mathbf{x}, y)$ be a random variable taking values in some space $\mathcal{Z}$. Our goal is to minimize $R(\mathbf{w}) = \mathrm{E}_Z[\ell(\mathbf{w}; Z)]$ over $\mathbf{w} \in \mathcal{S}$. More specifically, we assume that $\ell : \mathcal{S} \times \mathcal{Z} \to [0, B]$ is a function satisfying the following assumption:

Assumption LIST. [119] (LIpschitz and STrongly convex assumption) For all $z \in \mathcal{Z}$, the function $\ell_z(\mathbf{w}) = \ell(\mathbf{w}; z)$ is convex in $\mathbf{w}$ and satisfies:

1. ℓ_z has Lipschitz constant L with respect to (w.r.t.) the norm $\|\cdot\|$, i.e., $|\ell_z(\mathbf{w}) - \ell_z(\mathbf{w}')| \le L\|\mathbf{w} - \mathbf{w}'\|$.
2. ℓ_z is λ-strongly convex w.r.t. $\|\cdot\|$, i.e., $\forall\theta \in [0, 1]$, $\forall \mathbf{w}, \mathbf{w}' \in \mathcal{S}$,

$$\begin{aligned}&\ell_z(\theta\mathbf{w} + (1-\theta)\mathbf{w}')\\ \le&\theta\ell_z(\mathbf{w}) + (1-\theta)\ell_z(\mathbf{w}') - \frac{\lambda}{2}\theta(1-\theta)\|\mathbf{w} - \mathbf{w}'\|^2.\end{aligned}$$

For this kind of loss function, we consider online learning setting where $Z_1, \ldots, Z_T$ are sequentially provided with an additional assumption that they are independently identically distributed (i.i.d.). As a result, we have

$$\mathrm{E}[\ell(\mathbf{w}; Z_t)] = \mathrm{E}[\ell(\mathbf{w}, (\mathbf{x}_t, y_t))] := R(\mathbf{w}),\ \forall t,\ \mathbf{w} \in \mathcal{S}.$$

Now consider an online learning algorithm $\mathcal{A}$. This algorithm is initialized as $\mathbf{w}_1$, whenever Z_t is provided, the model $\mathbf{w}_t$ is updated to $\mathbf{w}_{t+1}$. Let $\mathrm{E}_t[\cdot]$ denote conditional expectation w.r.t. $Z_1, \ldots, Z_t$, then we have $\mathrm{E}_t[\ell(\mathbf{w}_t; Z_t)] = R(\mathbf{w}_t)$.

Under the above assumptions, we have the following theorem for the generalization ability of online learning using Freedman's inequality:

Theorem 1. *Under the assumption LIST, we have the following inequality, with probability at least* $1-4\delta\ln T$*,*

$$\begin{aligned}&\frac{1}{T}\sum_{t=1}^{T} R(\mathbf{w}_t) - R(\mathbf{w}_*) \le \frac{\mathrm{Reg}_{\mathcal{A}}(T)}{T}\\ &+4\sqrt{\frac{L^2\ln\frac{1}{\delta}}{\lambda}}\frac{\sqrt{\mathrm{Reg}_{\mathcal{A}}(T)}}{T} + \max\left(\frac{16L^2}{\lambda}, 6B\right)\frac{\ln\frac{1}{\delta}}{T},\end{aligned}$$

where $\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathcal{S}} R(\mathbf{w})$*. Further,* $\frac{1}{T}\sum_t R(\mathbf{w}_t)$ *can replaced with* $R(\bar{\mathbf{w}}_T)$ *where* $\bar{\mathbf{w}}_T = \frac{1}{T}\sum_t \mathbf{w}_t$*, using Jensen's inequality.*

If the assumption LIST is satisfied by $\ell_z(\mathbf{w})$, then the Online Gradient Descent (OGD) algorithm there generates $\mathbf{w}_1, \ldots, \mathbf{w}_T$, such that

$$\mathrm{Reg}_{\mathcal{A}}(T) \leq \frac{L^2}{2\lambda}(1 + \ln T).$$

Plugging the above inequality into the above theorem, and using

$$(1 + \ln T)/(2T) \leq \ln T/T, \ \forall T \geq 3$$

gives the following Corollary.

Corollary 1. *Suppose assumption LIST is satisfied for $\ell_z(\mathbf{w})$. Then the Online Gradient Descent algorithm that generates $\mathbf{w}_1, \ldots, \mathbf{w}_T$ and in the end output $\bar{\mathbf{w}}_T = \frac{1}{T}\sum_t \mathbf{w}_t$, satisfies the following inequality for its generalization ability, with probability at least $1 - 4\delta \ln T$,*

$$R(\bar{\mathbf{w}}_T) - R(\mathbf{w}_*) \leq \frac{L^2 \ln T}{\lambda T} + \sqrt{\ln \frac{1}{\delta}} \frac{4L^2 \sqrt{\ln T}}{\lambda T} + \max\left(\frac{16L^2}{\lambda}, 6B\right) \frac{\ln \frac{1}{\delta}}{T},$$

for any $T \geq 3$, where $\mathbf{w}_* = \arg\min_{\mathbf{w}\in\mathcal{S}} R(\mathbf{w})$.

3.7.2. Other Conversion Theories

Online to batch conversion has been studied by several other researchers [120, 119, 121, 122]. For general convex loss functions, Cesa-Bianchi et al., proved that the following generalization ability of online learning algorithm holds with probability at least $1 - \delta$, using the Hoeffding-Azuma methods

$$\begin{aligned} R(\bar{\mathbf{w}}_T) &\leq \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{w}_t; z_t) + \sqrt{\frac{2}{T} \ln \frac{1}{\delta}} \\ &= \frac{\mathrm{Reg}_{\mathcal{A}}(T)}{T} + \min_{\mathbf{w}\in\mathcal{S}} \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{w}; z_t) + \sqrt{\frac{2}{T} \ln \frac{1}{\delta}}, \end{aligned}$$

where the loss ℓ is assumed bounded by 1 [119]. The work of Zhang [121] is the closest the one in the previous subsection, which explicitly goes via the exponential moment method to drive sharper concentration results. In addition, Cesa-Bianchi and Gentile [122] improved their initial generalization bounds using Bernstein's inequality under the assumption $\ell(\cdot) \leq 1$, and proves the following inequality with probability at least $1 - \delta$,

$$R(\hat{\mathbf{w}}) \leq \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{w}_t; z_t) + O\left(\frac{\ln(T^2/\delta)}{T} + \sqrt{\frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{w}_t; z_t) \frac{\ln(T^2/\delta)}{T}}\right).$$

where $\hat{\mathbf{w}}$ is selected from $\mathbf{w}_1, \ldots, \mathbf{w}_T$, which can minimize a specifically designed penalized empirical risk. In particular, the generalization risk converges to $\frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{w}_t; z_t)$ at rate $O(\sqrt{\ln T^2/T})$ and vanishes at rate $O(\ln T^2/T)$ whenever the online loss $\sum_{t=1}^{T} \ell(\mathbf{w}_t; z_t)$ is $O(1)$.

4. Applied Online Learning for Supervised Learning

4.1. Overview

In this section, we survey the most representative algorithms for a group of non-traditional online learning tasks, wherein the supervised online algorithms cannot be used directly. These algorithms are motivated by new problem settings and applications which follow the traditional online setting, where the data arrives in a sequential manner. However, there was a need to develop new algorithms which were suited to these scenarios. Our review includes cost-sensitive online learning, online multi-task learning, online multi-view learning, online transfer learning, online metric learning, online collaborative filtering, online learning structured prediction, distributed online learning, online learning with neural networks, and online portfolio selection.

4.2. Cost-Sensitive Online Learning

In a supervised classification task, traditional online learning methods are often designed to optimize mistake rate or equivalently classification accuracy. However, it is well-known that classification accuracy becomes a misleading metric when dealing with class-imbalanced data which is common for many real-world applications, such as anomaly detection, fraud detection, intrusion detection, etc. To address this issue, cost-sensitive online learning [123] represents a family of online learning algorithms that are designed to take care of different misclassification costs of different classes in a class-imbalanced classification task. Next, we briefly survey these algorithms.

Perceptron Algorithms with Uneven Margin (PAUM). PAUM [124] is a cost-sensitive extension of Perceptron [37] and the Perceptron with Margins (PAM) algorithms [125]. Perceptron makes an update only when there is a mistake, while PAM tends to make more aggressive updates by checking the margin instead of mistake. PAM makes an update whenever $y_t \mathbf{w}_t^\top \mathbf{x}_t \leq \tau$, where $\tau \in \mathbb{R}^+$ is a fixed parameter controlling the aggressiveness. To deal with class imbalance, PAUM extends PAM via an

uneven margin setting, i.e., employing different margin parameters for the two classes: τ_+ and τ_-. Consequently, the update becomes $y_t \mathbf{w}_t^\top \mathbf{x}_t \leq \tau_{y_t}$. By properly adjusting the two parameters, PAUM achieves cost-sensitive updating effects for different classes. One of major limitations with PAUM is that it does not directly optimize a predefined cost-sensitive measure, thus, it does not fully resolve the cost-sensitive challenge.

Cost-sensitive Passive Aggressive (CPA). CPA [41] was proposed as a cost-sensitive variant of the PA algorithms. It was originally designed for multi-class classification by the following prediction rule: $\widehat{y_t} = \arg\max_y(\mathbf{w}_t \Phi(\mathbf{x}_t, y))$, where Φ is a feature mapping function that maps $\mathbf{x}_t$ to a new feature according to the class y. For simplicity, we restrict the discussion on the binary classification setting. Using $\Phi(\mathbf{x}, y) = \frac{1}{2} y\mathbf{x}$, we will map the formulas to our setting. The prediction rule is: $\widehat{y_t} = \text{sgn}(\mathbf{w}_t^\top \mathbf{x}_t)$. We define the cost-sensitive loss as

$$\ell(\mathbf{w}, \mathbf{x}, y) = \mathbf{w} \cdot \Phi(\mathbf{x}, \widehat{y}) - \mathbf{w} \cdot \Phi(\mathbf{x}, y) + \sqrt{\rho(y, \widehat{y})},$$

where $\rho(y_1, y_2)$ is the function define to distinguish the different cost of different kind misclassifications and we have assumed $\rho(y, y) = 0$. When being converted to binary setting, the loss becomes

$$\ell(\mathbf{w}, \mathbf{x}, y) = \begin{cases} 0 & y_t = \widehat{y} \\ |\mathbf{w}^\top \mathbf{x}| + \sqrt{\rho(y, \widehat{y})} & y_t \neq \widehat{y} \end{cases}$$

The mistake depends on the prediction confidence and the loss type. We omit the detailed update steps since it follows the similar optimization as PA learning as discussed before. Similar to PAUM, this algorithm also is limited in that it does not optimize a cost-sensitive measure directly.

Cost-Sensitive Online Gradient Descent (CSOGD). Unlike traditional OGD algorithms that often optimize accuracy, CSOGD [126, 127, 128] applies OGD to directly optimize two cost-sensitive measures:

(1) maximizing the weighted sum of *sensitivity* and *specificity*, i.e, $sum = \eta_p \times sensitivity + \eta_n \times specificity$, where the two weights satisfy $0 \leq \eta_p, \eta_n \leq 1$ and $\eta_p + \eta_n = 1$.

(2) minimizing the weighted *misclassification cost*, i.e., $cost = c_p \times M_p + c_n \times M_n$, where M_p and M_n are the number of false negatives and false positives respectively, $0 \leq c_p, c_n \leq 1$ are the cost parameters for positive and negative classes, respectively, and we assume $c_p + c_n = 1$.

The objectives can be equivalently reformulated into the following objective:

$$\sum_{y_t=+1} \rho \mathbf{I}_{(y_t \mathbf{w} \cdot \mathbf{x}_t < 0)} + \sum_{y_t=-1} \mathbf{I}_{(y_t \mathbf{w} \cdot \mathbf{x}_t < 0)}$$

where we set $\rho = \frac{\eta_p T_n}{\eta_n T_p}$ when maximizing the weighted sum, T_p and T_n are the number of positive and negative instances respectively; when minimizing the weighted misclassification cost, we instead set $\rho = \frac{c_p}{c_n}$. The objective is however non-convex, making it hard to optimize directly.

Instead of directly optimizing the non-convex objective, we attempt to optimize a convex surrogate. Specifically, we replace the indicator function $\mathbf{I}_{(\cdot)}$ by a convex surrogate, and attempt to optimize either one of the following modified hinge-loss functions at each online learning iteration:

$$\ell^I(\mathbf{w}; (\mathbf{x}, y)) = \max(0, \rho * \mathbf{I}_{(y=1)} + \mathbf{I}_{(y=-1)} - y(\mathbf{w} \cdot \mathbf{x}))$$

$$\ell^{II}(\mathbf{w}; (\mathbf{x}, y)) = (\rho * \mathbf{I}_{(y=1)} + \mathbf{I}_{(y=-1)}) * \max(0, 1 - y(\mathbf{w} \cdot \mathbf{x}))$$

One can then derive cost-sensitive ODG (CSOGD) algorithms by applying OGD to optimize either one of the above loss functions. The detailed algorithms can be found in [126]. Two recent works extend the problem setting to cost-sensitive classification of multi-class problem [129, 130]. Further there are efforts to do cost-sensitive online learning with kernels [131, 132].

Online AUC Maximization. Instead of optimizing accuracy, some online learning studies have attempted to directly optimize the Area Under the ROC curve (AUC), i.e.,

$$\text{AUC}(\mathbf{w}) = \frac{\sum_{i=1}^{T_+} \sum_{j=1}^{T_-} \mathbb{I}_{\mathbf{w} \cdot \mathbf{x}_i^+ > \mathbf{w} \cdot \mathbf{x}_j^-}}{T_+ T_-} = 1 - \frac{\sum_{i=1}^{T_+} \sum_{j=1}^{T_-} \mathbb{I}_{\mathbf{w} \cdot \mathbf{x}_i^+ \leq \mathbf{w} \cdot \mathbf{x}_j^-}}{T_+ T_-}$$

where $\mathbf{x}^+$ is a positive instance, $\mathbf{x}^-$ is a negative instance, T_+ is the total number of positive instances and T_- is the total number of negative instances. AUC measures the probability for a randomly drawn positive instance to have a higher decision value than a randomly sampled negative instance, and it is widely used in many applications. Optimizing AUC online [133] is however very challenging.

First of all, in the objective, the term $\sum_{i=1}^{T_+} \sum_{j=1}^{T_-} \mathbb{I}_{\mathbf{w} \cdot \mathbf{x}_i^+ \leq \mathbf{w} \cdot \mathbf{x}_j^-}$ is non-convex. To resolve this, a common way is to replace the indicator function by a convex surrogate, e.g., the hinge loss function

$$\ell(\mathbf{w}, \mathbf{x}_i^+ - \mathbf{x}_j^-) = \max\{0, 1 - \mathbf{w}(\mathbf{x}_i^+ - \mathbf{x}_j^-)\}$$

Consequently, the goal of AUC maximization in an online setting is equivalent to minimizing the accumulated

loss $\mathcal{L}_t(\mathbf{w})$ over all previous iterations, where the loss at the t-th iteration is defined:

$$\mathcal{L}_t(\mathbf{w}) = \mathbb{I}_{y_t=1} \sum_{\tau=1}^{t-1} \mathbb{I}_{y_\tau=-1} \ell(\mathbf{w}, \mathbf{x}_t - \mathbf{x}_\tau) + \mathbb{I}_{y_t=-1} \sum_{\tau=1}^{t-1} \mathbb{I}_{y_\tau=1} \ell(\mathbf{w}, \mathbf{x}_\tau - \mathbf{x}_t)$$

The above takes the sum of the pairwise hinge loss between the current instance $(\mathbf{x}_t, y_t)$ and all the received instances with the opposite class $-y_t$. Despite being convex, it is however impractical to directly optimize the above objective in online setting since one would need to store all the received instances and thus lead to the growing computation and memory cost in the online learning process.

The Online AUC Maximization method in [134] proposed a novel idea of exploring *reservoir sampling* techniques to maintain two buffers, B_+ and B_- of size N_+ and N_-, which aim to store a sketch of historical instances. Specifically, when receiving instance $(\mathbf{x}_t, y_t)$, it will be added to buffer B_{y_t} whenever it is not full, i.e. $|B_{y_t}| < N_{y_t}$. Otherwise, $\mathbf{x}_t$ randomly replaces one instance in the buffer with probability $\frac{N_{y_t}}{N_{y_t}^{t+1}}$, where $N_{y_t}^{t+1}$ is the total number of instances with class y_t received so far. The idea of reservoir sampling is to guarantee the instances in the buffers simulate a uniform sampling from the original full dataset. As a result, the loss $\mathcal{L}_t(\mathbf{w})$ can be approximated by only considering the instances in the buffers, and the classifier $\mathbf{w}$ can be updated by either a regular OGD or PA approach.

Others. To improve the study in [134], a number of following studies have attempted to make improvements from different aspects. For example, the study in [135] generalized online AUC maximization as online learning with general pairwise loss functions, and offered new generalization bounds for online AUC maximization algorithms similar to [134]. The bounds were further improved by [136] which employs a generic decoupling technique to provide Rademacher complexity-based generalization bounds. In addition, the work in [137] overcomes the buffering storage cost by developing a regression-based algorithm which only needs to maintain the first and second-order statistics of training data in memory, making the resulting storage requirement independent from the training size. The very recent work in [138] presented a new second-order AUC maximization method by improving the convergence using the adaptive gradient algorithm. The stochastic online AUC maximization (SOLAM) algorithm [139] formulates the online AUC maximization as a stochastic saddle point problem and greatly reduces the memory cost.

4.3. Online Multi-task Learning

Multi-task Learning [140] is an approach that learns a group of related machine learning tasks together. By considering the relationship between different tasks, multi-task learning algorithms are expected to achieve better performance than algorithms that learn each task individually. Batch multi-task learning problems are usually solved by transfer learning methods [141] which transfer the knowledge learnt from one task to another similar tasks. In Online Multi-task Learning (OML) [142, 143, 144], however, the tasks have to be solved in parallel with instances arriving sequentially, which makes the problem more challenging.

During time t, each of the task $i \in \{1, ...K\}$ receives an instance $\mathbf{x}_{i,t} \in \mathbb{R}^{d_i}$, where d_i is the feature dimension of task i. The algorithm then makes a prediction for each task i based on the current model $\mathbf{w}_{i,t}$ as $\hat{y}_{i,t} = \text{sign}(\mathbf{w}_{i,t}^\top \mathbf{x}_{i,t})$. After making the prediction, the true labels $y_{i,t}$ are revealed and we get a loss function vector $\boldsymbol{\ell}_{i,t} \in \mathbb{R}_+^K$. Finally, the models are updated by considering the loss vector and task relationship.

A straightforward baseline algorithm is to parallel update all the classifiers $\mathbf{w}_i, i \in \{1, ..., K\}$. OML algorithm should utilize the relationships between tasks to achieve higher accuracy compared with the baseline. The multi-task Perceptron algorithm [145] is a pioneering work of OML that considers the inter-task relationship. Assuming that a matrix $A \in \mathbb{R}^{K*K}$ is known and fixed, we can update the modePrasanthi Nairl i when an instance $\mathbf{x}_{j,t}$ for task j is received as follows:

$$\mathbf{w}_{i,t+1} = \mathbf{w}_{i,t} + y_{j,t} A_{j,i}^{-1} \mathbf{x}_{j,t}$$

Some later approaches [146, 147] try to learn the relationship matrix in an optimization problem, which also offers the flexibility of using a time-varying relationship.

Apart from learning the relationship explicitly, another widely used approach in OML field is to add some structure regularization terms to the original objective function [148, 149, 150]. For example, we may assume that each model is made up of two parts, a shared part across all tasks $\mathbf{w}_0$ and an individual part $\mathbf{v}_i$, i.e.

$$\mathbf{w}_i = \mathbf{w}_0 + \mathbf{v}_i$$

where the common part helps to take advantage of the task similarity. Now the regularized loss becomes

$$\sum_{i=1}^{K} (\ell_{i,t} + \|\mathbf{v}_i\|_2^2) + \lambda \|\mathbf{w}_0\|_2^2$$

Many later works improve this by considering more complex inter-task relationships [151].

4.4. Online Multi-view Learning

Multi-view learning deals with problems where data are collected from diverse domains or obtained from various feature extractors. By exploring features from different views, multi-view learning algorithms are usually more effective than single-view learning. In literature, there many surveys that offer comprehensive summary of state-of-the-art methods in multi-view learning in batch setting [152, 153, 154], while few works tried to address this problem in online setting.

Two-view PA. We first introduce a seminal work, the two-view online passive aggressive learning (Two-view PA) algorithm [155], which is motivated by the famous single-view PA algorithm [41] and the two-view SVM algorithm [156] in batch setting.

During each iteration t, the algorithm receives an instance $(\mathbf{x}_t^A, \mathbf{x}_t^B, y_t)$, where $\mathbf{x}_t^A \in \mathbb{R}^n$ is the feature vector in the first view, $\mathbf{x}_t^B \in \mathbb{R}^m$ is for the second view and $y_t \in \{1, -1\}$ is the label. The goal is to learn two classifiers $\mathbf{w}^A \in \mathbb{R}^n$ and $\mathbf{w}^B \in \mathbb{R}^m$, each for one view, and make accuracy prediction with their combination

$$\hat{y}_t = \text{sign}(\mathbf{w}^A \cdot \mathbf{x}^A + \mathbf{w}^B \cdot \mathbf{x}^B).$$

Thus the hinge loss at iteration t is redefined as

$$\ell_t(\mathbf{w}_t^A, \mathbf{w}_t^B) = \max(0, 1 - \frac{1}{2} y_t(\mathbf{w}^A \cdot \mathbf{x}_t^A + \mathbf{w}^B \cdot \mathbf{x}_t^B))$$

In the single-view PA algorithm, the objective function in each iteration is a balance between two desires: minimizing the loss function at the current instance and minimizing the change made to the classifier. While to utilize the special information in the multi-view data, an additional term that measures the agreement between two terms is added. Thus, the optimization is as follows,

$$\begin{aligned}(\mathbf{w}_{t+1}^A, \mathbf{w}_{t+1}^B) = \arg\min_{\mathbf{w}^A, \mathbf{w}^B} & \tfrac{1}{2}\|\mathbf{w}^A - \mathbf{w}_t^A\|_2^2 + \tfrac{1}{2}\|\mathbf{w}^B - \mathbf{w}_t^B\|_2^2 \\ & + C\ell_t(\mathbf{w}_t^A, \mathbf{w}_t^B) + \gamma|y_t \mathbf{w}^A \cdot \mathbf{x}_t^A - y_t \mathbf{w}^B \cdot \mathbf{x}_t^B|\end{aligned}$$

where γ and C are weight parameters. Fortunately, this optimization problem has a closed form solution.

Other related works:. Other than solving classification tasks, online multi-view learning has been explored for solving similarity learning or distance metric learning, such as Online multimodal deep similarity learning [157] and online multi-modal distance metric learning [158].

4.5. Online Transfer Learning

Transfer learning aims to address the machine learning tasks of building models in a new target domain by taking advantage of information from another existing source domain through knowledge transfer. Transfer learning is important for many applications where training data in a new domain may be limited or too expensive to collect. There are two different problem settings, *homogeneous* setting where the target domain shares the same feature space as the old/source one, and *heterogeneous* setting where the feature space of the target domain is different from that of the source domain. Although several surveys on transfer learning are available [159, 141], most of the referred algorithms are in batch setting.

Online Transfer Learning (OTL) algorithms aim to learn a classifier $f : \mathbb{R}^d \to \mathbb{R}$ from a well-trained classifier $h : \mathbb{R}^{d'} \to \mathbb{R}$ in the source domain and a group of sequentially arriving instances $\mathbf{x}_t \in R^d, t = 1, ..., T$ in the target domain. For conciseness, we will use the previous notations for the online classification task. We first introduce a pioneer work of OTL [160, 161].

Homogeneous Setting. One key challenge of this task is to address the concept drifting issue that often occurs in this scenario. The algorithm in homogeneous setting ($d = d'$) is based on the ensemble learning approach. At time t, an instance $\mathbf{x}_t$ is received. The algorithm makes a prediction based on the weighted average of the classifier in the source domain $h(\mathbf{x}_t)$ and the current classifier in the target domain $f_t(\mathbf{x}_t)$,

$$\hat{y}_t = \text{sgn}(w_{1,t}\Pi(h(\mathbf{x}_t)) + w_{2,t}\Pi(f_t(\mathbf{x}_t)) - \frac{1}{2})$$

where $w_{1,t} > 0, w_{2,t} > 0$ are the weights for the two functions and need to be updated during each iteration. Π is a normalization function, i.e. $\Pi(a) = \max(0, \min(1, \frac{a+1}{2}))$.

In addition to updating the function f_t by using some online learning algorithms, the weights $w_{1,t}$ and $w_{2,t}$ should also be updated. One suggested scheme is

$$\begin{aligned} w_{1,t+1} &= C_t w_{1,t} \exp(-\eta \ell^*(h)) \\ w_{2,t+1} &= C_t w_{2,t} \exp(-\eta \ell^*(f_t)) \end{aligned}$$

where C_t is a normalization parameter to keep $w_{1,t+1} + w_{2,t+1} = 1$ and $\ell^*(g) = (\Pi(g(\mathbf{x}_t)) - \Pi(y_t))^2$.

Heterogeneous Setting. Since heterogeneous OTL is generally very challenging, we consider one simpler case where the feature space of the source domain is a subset of that of the target domain. Without loss of generality, we assume the first d' dimensions of $\mathbf{x}_t$ represent

the old features, denoted as $\mathbf{x}_t^{(1)} \in \mathbb{R}^{d'}$. The other dimensions form a feature vector $\mathbf{x}_t^{(2)} \in \mathbb{R}^{d-d'}$. The key idea is to adopt a co-regularization principle of online learning two classifiers $f_t^{(1)}$ and $f_t^{(2)}$ simultaneously from the two views, and predict an unseen example on the target domain by

$$\hat{y}_t = \text{sgn}\left(\frac{1}{2} f_t^{(1)}(\mathbf{x}_t^{(1)}) + \frac{1}{2} f_t^{(2)}(\mathbf{x}_t^{(2)})\right)$$

The function from source domain $h(\mathbf{x}^{(1)})$ is used to initialize $f_t^{(1)}$. The update strategy at time t is

$$(f_{t+1}^{(1)}, f_{t+1}^{(2)}) = \arg\min_{f^{(1)}, f^{(2)}} \frac{\gamma_1}{2}\|f^{(1)} - f_t^{(1)}\|_{\mathcal{H}}^2 + \frac{\gamma_1}{2}\|f^{(2)} - f_t^{(2)}\|_{\mathcal{H}}^2 + C\ell_t$$

where γ_1, γ_2 and C are positive parameters and ℓ_t is the hinge loss.

Other Related Work. Multi-source Online Transfer Learning (MSOTL) [162, 163] solves a more challenging problem where k classifiers $h_1, ...h_k$ are provided by k sources. The goal is to learn the optimal combination of the k classifiers and the online updated classifier f_t. A naive solution is to construct a new $d + k$ dimensional feature representation $\mathbf{x}_t' = [\mathbf{x}_t, h_1(\mathbf{x}_t), ..., h_k(\mathbf{x}_t)]$ and the online classifier in this new feature space. An extension of MSOTL [164] aims to deal with transfer learning problem under two disadvantageous assumptions, negative transfer where instead of improving performance, transfer learning from highly irrelevant sources degrades the performance on the target domain, and imbalanced distributions where examples in one class dominate. The Co-transfer Learning algorithm [165, 166] considers the transfer learning problem not only in multi-source setting but also in the scenario where a large group of instances are unlabeled.

4.6. Online Metric Learning

Distance/Similarity metric learning [167] is a fundamental problem in Machine Learning field and is critical to many real-world applications, such as image retrieval, classification and clustering. The goal of distance metric learning is to seek a distance matrix $A \in \mathbb{R}^{d\times d}$, so that for any pair of instances $\mathbf{x}_i \in R^d$ and $\mathbf{x}_j \in \mathbb{R}^d$, the Mahalanobis distance

$$d_A(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^\top A(\mathbf{x}_i - \mathbf{x}_j)$$

reflects the distance or similarity between the two instances accurately. Assuming that $A \succeq 0$ is a symmetric positive semi-definite matrix, there exist a matrix $W \in \mathbb{R}^{d\times d}$ such that $A = W^\top W$. The Mahalanobis distance can be rewritten as

$$d_A(\mathbf{x}_i, \mathbf{x}_j) = \|W\mathbf{x}_i - W\mathbf{x}_j\|_2^2$$

Thus, the distance d_A is the Euclidean distance between two instances in a linearly transformed space.

Unfortunately, it is difficult to collect the data with real numbers as the exact value of distances. Therefore, we usually have two types of problem settings. 1) Pairwise data. During time t, we receive a pair of instances $(\mathbf{x}_t^1, \mathbf{x}_t^2)$ and a label y_t which equals to $+1$ if the pair is similar and -1 otherwise. 2) Triple data. We are given a triple $(\mathbf{x}_t, \mathbf{x}_t^+, \mathbf{x}_t^-)$ at time t, with the knowledge that $d_A(\mathbf{x}_t, \mathbf{x}_t^+) > d_A(\mathbf{x}_t, \mathbf{x}_t^-)$. The goal of online learning is to minimize the accumulated loss during the whole learning process $\sum_{t=1}^T \ell_t(A)$, where ℓ_t is the loss suffered from imperfect prediction at time t. When evaluating the output model for online-to-batch-conversion, we may use the metric for information retrieval to evaluate the performance in the test dataset, such as mean average precision (mAP) or precision-at-top-k.

Below, we briefly introduce a few representative work for distance metric learning in online setting.

Pseudo-metric Online Learning (POLA). The POLA algorithm [168] learns the distance matrix A from a stream of pairwise data. The loss at time t is an adaptation of the hinge loss

$$\ell_t(A, b) = \max\{0, 1 - y_t(b - d_A(\mathbf{x}_t^1, \mathbf{x}_t^2))\}$$

where b is the adaptive threshold value for similarity and will be updated incrementally along with matrix A. We denote $(A, b) \in \mathbb{R}^{d^2+1}$ as the new variable to learn. The update strategy mainly follows the PA approach

$$(A_{t+\frac{1}{2}}, b_{t+\frac{1}{2}}) = \arg\min_{(A,b)} \|(A, b) - (A_t, b_t)\|_2^2$$
$$s.t. \quad \ell_t(A, b) = 0$$

The solution $(A_{t+\frac{1}{2}}, b_{t+\frac{1}{2}})$ makes a correct prediction to the current pair and memorizes as much information from the previous model as possible. Then, the algorithm projects this solution to the feasible space $\{(A, b) : A \succeq 0, b \geq 1\}$ to get the updated model (A_{t+1}, b_{t+1}). Like the PA algorithms, it's easy to generalize the POLA algorithm to a soft margin variant which is robust to noise.

There is another similar work named Online Regularized Metric Learning [169], which is simpler due to the adoption of fixed threshold. The loss function is defined as

$$\ell_t(A) = \max(0, b - y_t(1 - d_A(\mathbf{x}_t^1, \mathbf{x}_t^2)))$$

whose gradient is

$$\nabla\ell_t(A) = y_t(\mathbf{x}_t^1 - \mathbf{x}_t^2)(\mathbf{x}_t^1 - \mathbf{x}_t^2)^\top$$

At time t, if the prediction is incorrect, the algorithm updates the matrix A by projecting the OGD updated matrix into the positive definite space.

Information Theoretic Metric Learning (ITML). In the above introduced algorithms, the distance between two matrices A_t and A is usually defined using the Frobenius norm, i.e. $\|A_t - A\|_F^2$, while a different definition is adopted in the information theoretic metric learning algorithms [170, 171]. Given a Mahalanobis distance parameterized by A, we express its corresponding multivariate Gaussian distribution as $p(\mathbf{x}, A) = \frac{1}{Z}\exp(-\frac{1}{2}d_A(\mathbf{x}, \boldsymbol{\mu}))$. The difference between matrices is defined as the KL divergence between the two distributions. Assuming all distributions have the same mean, the KL divergence can be calculated as,

$$\mathrm{KL}(p(\mathbf{x}; A), p(\mathbf{x}, A_t)) = \mathrm{tr}(AA_t^{-1}) - \log\det(AA_t^{-1}) - d$$

Similar to the PA update strategy, during time t the matrix is updated by solving the optimization problem

$$A_{t+1} = \arg\min_{A\geq 0} \mathrm{KL}(p(\mathbf{x}; A), p(\mathbf{x}, A_t)) + \eta\ell_t(A)$$

where $\eta > 0$ is a regularization parameter. This optimization problem enjoys a closed form solution.

Online Algorithm for Scalable Image Similarity Learning (OASIS). The OASIS algorithm learns a similarity matrix $W \in \mathbb{R}^d$ from a stream of triplet data, where the similarity score between two instances is defined as

$$S_W(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top W \mathbf{x}_j$$

During time t, one triplet $(\mathbf{x}_t, \mathbf{x}_t^+, \mathbf{x}_t^-)$ is received. Ideally, we expect the $\mathbf{x}_t$ is more similar to $\mathbf{x}_t^+$ than to $\mathbf{x}_t^-$, i.e. $S_W(\mathbf{x}_t, \mathbf{x}_t^+) > S_W(\mathbf{x}_t, \mathbf{x}_t^-)$. Similar to the PA algorithm, for a large margin, the loss function is defined as the hinge loss

$$\ell_t(W) = \max\{0, 1 - S_W(\mathbf{x}_t, \mathbf{x}_t^+) + S_W(\mathbf{x}_t, \mathbf{x}_t^-)\}$$

The optimization problem to solve for updating W_t is

$$W_{t+1} = \arg\min_W \frac{1}{2}\|W - W_t\|_F^2 + C\xi$$
$$s.t. \quad \ell_t(W) \leq \xi \ \ and \ \ \xi \geq 0$$

where C is the parameter controlling the trade-off.

The OASIS algorithm is different from the previous work in several aspects. First, it never requires the similarity matrix W to be positive semi-definite and thus saves the computational cost for the projection step. Second, the definition of similarity score is much simpler than the Mahalanobis distance. Third, the triplet data is easy to collect. These advantages led to the state-of-the-art performance of the OASIS algorithm. Most of these methods assume a linear proximity function, and to address this limitation, [172] developed a kernelized approach to do metric learning. Another approach performs sparse online metric learning in order to be suitable for very high dimensional data [173, 174].

There is one notable work that solves the online similarity learning in an active learning setting [175], which significantly reduces the cost of collecting labeled data. SimApp [176, 177] applies technique of the online similarity learning to mobile application recommendation and tagging. The online multi-modal distance metric learning algorithms [178, 158] learn distance metric in multiple modals, which enables it to be applied to image retrieval application.

4.7. Online Collaborative Filtering

Collaborative Filtering (CF) [179] is one of the most successful learning techniques in building recommendation systems. Different from content based filtering techniques, CF algorithms usually require little or no knowledge about the features of items or users apart from the previous preferences. The fundamental assumption of CF algorithms is that if two users rate many items similarly, they will probably share common preference on the other items. Several survey paper provides detailed introduction to the regular CF techniques [180, 181]. However, most of the surveyed algorithms are in batch setting. We now review several popular algorithms for online CF tasks.

An online CF algorithm works on a sequence of observed ratings given by n users to m items. At time $t \in \{1, 2, ..., T\}$, the algorithm receives the index of a user $u^{(t)} \in \{1, 2, ...n\}$ and the index of an item $i^{(t)} \in \{1, 2, ...m\}$ and makes a prediction of the rating $\hat{r}_{u,i}^{(t)} \in \mathbb{R}$ based on the knowledge of the previous ratings. Then the real rating $r_{u,i}^{(t)} \in \mathbb{R}$ is revealed and the algorithm updates the model based on the loss suffered from the imperfect prediction, denoted as $\ell(\hat{r}_{u,i}^{(t)}, r_{u,i}^{(t)})$. The goal of online CF is to minimize the Root Mean Square Error (RMSE) or Mean Absolute Error(MAE) along the whole learning process, defined as follows:

$$\mathrm{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(r_{u,i}^{(t)} - \hat{r}_{u,i}^{(t)})^2}, \quad \mathrm{MAE} = \frac{1}{T}\sum_{t=1}^{T}|r_{u,i}^{(t)} - \hat{r}_{u,i}^{(t)}|$$

Collaborative filtering techniques are generally categorized into two types: memory-based methods and model-based methods.

In the following we will briefly introduce some of the most representative algorithms in each category.

Memory-Based CF Methods. Memory-based CF algorithms usually work in the following steps:

1. Calculate the similarity score between any pairs of items. For example, the cosine similarity between item i and item j is defined as,

$$S_{i,j} = \frac{\sum_{u \in \mathcal{U}_i \cap \mathcal{U}_j} r_{ui} \cdot r_{u,j}}{\sqrt{\sum_{u \in \mathcal{U}_i} r_{ui}^2 \sum_{u \in \mathcal{U}_j} r_{u,j}^2}}$$

   where $\mathcal{U}_i$ denotes the set of users that have rated item i.

2. For each item i, find its k nearest neighbor set $\mathcal{N}_i$ based on the similarity score.

3. Predict the rating $r_{u,i}$ as the weighted average of ratings from user u to the neighbors of item j, where the weight is proportional to the similarity.

We name the above described algorithm as item-based CF, while similarly, the predictions may also be calculated as the weighted average of ratings from similar users, which is called user-based CF method.

Memory-based CF methods were widely used in some early generation recommendation systems. However, they suffer from sensitivity to the data sparsity. Obviously, the similarity score $S_{i,j}$ is only available when there is at least one common user that rates the two items i and j, which might be unrealistic during the beginning rounds of online learning. Another challenge is the large time consumption when updating the large number of similarity scores incrementally with the arrival of new ratings. The Online Evolutionary Collaborative Filtering [182] algorithm provides an efficient similarity score updating method to address this problem. In addition, the decrease of rating influence over time is also considered.

Model-Based CF Methods. As introduced above, there is not much existing work in online memory-based CF methods because of its two limitations, i.e. sensitivity to data sparsity and inefficient similarity score update. To address this issue, lots of model-based CF algorithms have been proposed and have achieved encouraging results. One of the most successful approaches is the matrix factorization methodology [183]. It assumes that the rating by a user to an item is determined by k potential features, $k \ll n, m$. Thus each user u can be represented by a vector $\mathbf{u}_u \in \mathbb{R}^k$, and each item i can be represented by a vector $\mathbf{v}_i \in \mathbb{R}^k$. The rating $r_{u,i}$ can then be approximated by the dot product of the corresponding user vector and item vector, i.e., $\hat{r}_{u,i} = \mathbf{u}_u^\top \mathbf{v}_i$. The CF problem can then be represented by the following optimization problem:

$$\operatorname*{arg\,min}_{U \in \mathbb{R}^{k \times n}, V \in \mathbb{R}^{k \times m}} \sum_{t=1}^{T} \ell(\mathbf{u}_u^{(t)} \cdot \mathbf{v}_i^{(t)}, r_{u,i}^{(t)})$$

where the loss function is defined to optimize certain evaluation metric:

$$\ell_{rmse}(\hat{r}_{u,i}, r_{u,i}) = (r_{u,i} - \hat{r}_{u,i})^2 \quad \ell_{mae}(\hat{r}_{u,i}, r_{u,i}) = |r_{u,i} - \hat{r}_{u,i}|$$

The regularized loss at time t is

$$\mathcal{L}_t = \lambda \|\mathbf{u}_u^{(t)}\|_2^2 + \lambda \|\mathbf{v}_i^{(t)}\|_2^2 + \ell(\mathbf{u}_u^{(t)} \cdot \mathbf{v}_i^{(t)}, r_{u,i}^{(t)})$$

where $\lambda > 0$ is the regularization parameter.

One straightforward CF approach is to adopt the OGD algorithm on the regularized loss function [184],

$$\mathbf{u}_u^{(t+1)} = (1 - 2\eta\lambda)\mathbf{u}_u^{(t)} - \eta \frac{\partial \ell(\mathbf{u}_u^{(t)} \cdot \mathbf{v}_i^{(t)}, r_{u,i}^{(t)})}{\partial \mathbf{u}_u^{(t)}}$$

$$\mathbf{v}_i^{(t+1)} = (1 - 2\eta\lambda)\mathbf{v}_i^{(t)} - \eta \frac{\partial \ell(\mathbf{u}_u^{(t)} \cdot \mathbf{v}_i^{(t)}, r_{u,i}^{(t)})}{\partial \mathbf{v}_i^{(t)}}$$

where $\eta > 0$ is the learning rate.

Later, several speeding up algorithms are proposed, such as the Online Multi-Task Collaborative Filtering algorithm [147], the Dual-Averaging Online Probabilistic Matrix Factorization algorithm [185], the Adaptive Gradient Online Probabilistic Matrix Factorization algorithm [185] and the second order Online Collaborative Filtering algorithm [50, 186]. These algorithms adopt more effective update strategies beyond OGD and thus can achieve faster convergence and catch the rapid user preference changes in real world recommendation tasks.

Besides the algorithms introduced, there are many CF methods that apply the above basic problem setting to more challenging tasks. First, in many applications, the features of users (e.g. age) and items (e.g. description) are available and thus need to be considered for better prediction. This generalized CF problem can be solved by using tensor product kernel functions [187]. The Online Low-rank with Features algorithm [184] addresses this problem in online setting. However, it only adopts the linear kernel for efficiency. Perhaps, better performance might be achieved if online budget learning algorithms are adopted. Second, most CF algorithms are based on a regression model, which is mainly concerned

with the accuracy of rating prediction, while there are some applications where ranking prediction might be much more important. Two algorithms based on OGD and Dual Averaging approaches are proposed to address this problem by replacing the regression-based loss with the ranking-based loss [188]. Third, for extremely large scale applications, when the model has to be learnt using parallel computing, conventional OGD update is not suitable because of the possible conflict in updating the user / item vectors. The Streaming Distributed Stochastic Gradient Descent algorithm [189] provides an operable approach to addresses this problem. Finally, the CF methods for Google News recommendation [190] is a combination of memory-based and model-based algorithms. The techniques such as MinHash clustering, Probabilistic Latent Semantic Indexing, and covisitation counts are very different from the algorithms based on matrix factorization approaches. Fourth, to address the sparsity problem and imbalance of rating data, [191] incorporate content information via latent dirichlet allocation into CF.

4.8. Online Learning to Rank

Learning to rank is an important family of machine learning techniques for information retrieval and recommender systems [192, 193, 194, 195, 196]. Different from classification problems where instances are classified into classes such as "relevant" or "not relevant", learning to rank aims to produce a permutation of a group of unseen instances which is similar to the knowledge acquired from the previously seen rankings. To evaluate the performance of ranking algorithms, metrics for information retrieval such as Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG) and Precision-At-Top-k are most popular.

Unlike traditional learning to rank methods which are often based on batch learning [194], in this survey, we mainly focus on the review of learning to rank methods in the online setting [197, 198], where instances are observed sequentially. Learning to rank techniques are generally categorized into two approaches: pointwise and pairwise. We will introduce some of the most representative algorithms in each category.

Pointwise Approach:. We first introduce a simple Perceptron-based algorithm, the Prank [199, 200], which provides a straightforward view of the commonly used problem setting for pointwise learning to rank approaches.

To define the online learning to rank problem setting formally, we have a finite set of ranks $\mathcal{Y} = \{1, ..., k\}$ from which a rank $y \in \mathcal{Y}$ is assigned to an instance $\mathbf{x} \in \mathbb{R}^d$. During time t, an instance $\mathbf{x}_t$ is received and the algorithm makes a prediction $\hat{y}_t$ based on the current model $H_t : \mathbb{R}^d \to \mathcal{Y}$. Then the true rank y_t is revealed and the model is updated based on the loss $\ell(\hat{y}_t, y_t)$. The loss, for instance, can be defined as $\ell(\hat{y}_t, y_t) = |\hat{y}_t, y_t|$. The goal of the online learning to rank task is to minimize the accumulated loss along the whole learning process $\sum_{t=1}^{T} \ell(\hat{y}_t, y_t)$.

The ranking rule of Prank algorithm consists of the combination of Perceptron weight $\mathbf{w} \in \mathbb{R}^d$ and a threshold vector $\mathbf{c} \in \{\mathbb{R}, \infty\}^d$, whose elements are in nondecreasing order i.e., $c^1 \leq c^2 \leq, ..., \leq c^k = \infty$. Like the Perceptron algorithm, the rank prediction is determined by the value of the inner product $\mathbf{w}_t^\top \mathbf{x}_t$,

$$\hat{y}_t = \min_{r \in \{1,...,k\}} \{r : \mathbf{w}_t^\top \mathbf{x}_t < c_t^r\}$$

We can expand the target rank y_t to a vector $\mathbf{y}_t = \{+1, ..., +1, -1, ..., -1\} \in \mathbb{R}^k$. For $r = 1, ...k$, $y_t^r = -1$ if $y_t < r$, and $y_t^r = 1$ otherwise. Thus, for a correct prediction, $y_t^r(\mathbf{w}_t^\top \mathbf{x}_t - c_t^r) > 0$ holds for all $r \in \mathcal{Y}$. When a mistake appears $\hat{y}_t \neq y_t$, there is subset $\mathcal{M}$ of $\mathcal{Y}$ where $y_t^r(\mathbf{w}_t^\top \mathbf{x}_t - c_t^r) > 0$ does not hold. The update rule is to move the corresponding thresholds for ranks in $\mathcal{M}$ and the weight vector toward each other:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + (\sum_{r \in \mathcal{M}} y_t^r)\mathbf{x}_t, \quad \textit{and} \;\; c_{t+1}^r = c_t^r - y_t^r, \forall r \in \mathcal{M}$$

It is proven theoretically that the elements in threshold vector $\mathbf{c}$ are always in nondecreasing order and the total number of mistakes made during the learning process is bounded.

Online Aggregate Prank-Bayes Point Machine (OAP-BPM) [201] is an extension of the Prank algorithm by approximating the Bayes point. Specifically, the OAP-BPM algorithm generates N diverse solutions of $\mathbf{w}$ and $\mathbf{c}$ during each iteration and combines them for a better final solution. We denote $H_{j,t}$ as the j-th solution at time t. The algorithm samples N Bernoulli variables $b_{j,t} \in \{0, 1\}$, $j = \{1, ..., N\}$ independently. If $b_{i,t} = 1$, The j-th solution is updated using the Prank algorithm according to the current instance, $H_{j,t+1} = Prank(H_{j,t}, (\mathbf{x}_t, y_t))$. Otherwise, no update is conducted to the j-th solution. The solution $\mathbf{w}_{t+1}$ and $\mathbf{c}_{t+1}$ is the average over N solutions. This work shows better generalization performance than the basic Prank algorithm.

Pairwise Approach:. One simple method is to address the ranking problem by transforming it to a classification problem [202]. In a more challenging problem setting, where no accurate rank y is available when collecting the data, only pairwise instances are provided.

At time t, a pair of instances $(\mathbf{x}_t^1, \mathbf{x}_t^2)$ are received with the knowledge that $\mathbf{x}_t^1$ is ranked before $\mathbf{x}_t^2$ or the inverse case, and the aim is to find a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that fits the instance pairs, i.e., $f(\mathbf{x}^1) > f(\mathbf{x}^2)$ when it is known $\mathbf{x}_t^1$ is ranked before $\mathbf{x}_t^2$ or otherwise $f(\mathbf{x}^1) < f(\mathbf{x}^2)$. When the function is linear, the problem can be rewritten as $\mathbf{w}^\top(\mathbf{x}^1 - \mathbf{x}^2) > 0$ when $\mathbf{x}^1$ is in front and otherwise $\mathbf{w}^\top(\mathbf{x}^1 - \mathbf{x}^2) < 0$, where $\mathbf{w} \in \mathbb{R}^d$ is the weight vector. This problem can easily be solved by using a variety of online classification algorithms (Online Gradient Descent [203] for example).

4.9. Distributed Online Learning

Distributed online learning [204] has become increasing popular due to the explosion in size and complexity of datasets. Similar to the mini-batch online learning, during each iteration, K instances are received and processed simultaneously. Usually, each node processes one of the instances and updates its local model. These nodes communicate with each other to make their local model consistent. When designing a distributed algorithm, besides computational time cost and accuracy, another important issue to consider is the communication load between nodes. This is because in real world systems with limited network capacity and large communication burden result in long latency.

Based on the network structure, distributed online learning algorithms can be classified into two groups, centralized and decentralized algorithms. A centralized network is made up of 1 master node and $K - 1$ worker nodes, where the workers can only communicate with the master node. By gathering and distributing information across the network, it is not difficult for distributed algorithms to reach a global consensus [34]. In decentralized networks, however, there is no master and each node can only communicate with its neighbors [205, 206]. Although the algorithms are more complex, decentralized learning is more popular because of the robustness of network structure.

We can also group the distributed learning algorithms by synchronized and asynchronized working mode. Synchronized algorithms are easy to design and enjoy better theoretical bounds but the speed of the whole network is limited by the slowest node. Asynchronized learning algorithms, on the other hand, are complex and usually have worse theoretical bounds. The advantage is its faster processing speed [207]

4.10. Online Learning with Neural Networks

In addition to kernel-based online learning approaches, another rich family of nonlinear online learning algorithms follows the general idea of neural network based learning approaches [208, 209, 210, 211, 212, 213]. For example, the Perceptron algorithm could be viewed as the simplest form of online learning with neural networks (but it is not nonlinear due to its trivial network). Despite many extensive studies for online learning (or incremental learning) with neural networks, many of existing studies in this field fall short in either of some critical drawbacks, including the lack of theoretical analysis for performance guarantee, heuristic algorithms without solid justification, and computationally too expensive to achieve efficient and scalable online learning purposes. Due to the large body of related work, it is impossible to examine every piece of work in this area. In the following, we review several of the most popularly cited related papers and discuss their key ideas for online learning with neural networks.

A series of related work has explored online convex optimization methods for training classical neural network models [214], such as the Multi-layer Perceptron (MLP). For example, online/stochastic gradient descent has been extensively studied for training neural networks in sequential/online learning settings, such as the efficient back-propagation algorithm using SGD [211]. These works are mainly motivated to accelerate the training of batch learning tasks instead of solving online learning tasks directly and seldom give theoretical analysis.

In addition to the above, we also briefly review other highly cited works that address online/incremental learning with neural networks. For example, the study in [208] presented a novel learning algorithm for training fully recurrent neural networks for temporal supervised learning which can continually run over time. However, the work is limited in lacking theoretical analysis and performance guarantee, and the solution could be quite computationally expensive. The work in [209] presented a Resource-Allocating Network (RAN) that learns a two-layer network by a strategy for allocating new units whenever an unusual pattern occurs and a learning rule for refining the network using gradient descent. Although the algorithm was claimed to run in online learning settings, it may suffer poor scalability as the model complexity would grow over time. The study in [210] proposed a new neural network architecture called "ARTMAP" that autonomously learns to classify arbitrarily many, arbitrarily ordered vectors into recognition categories based on predictive success. This supervised learning system was built from a pair of ART modules that are capable of self organizing stable recognition categories in response to arbitrary sequences of input patterns. Although an online learning

simulation has been done with ART (adaptive resonance theory), the solution is not optimized directly for online learning tasks and there is also no theoretical analysis. The work in [212] proposed an online sequential extreme learning machine (OS-ELM) which explores an online/sequential learning algorithm for training single hidden layer feedforward networks (SLFNs) with additive or radial basis function (RBF) hidden nodes in a unified framework. The limitation also falls short in some heuristic approaches and lacking theoretical analysis. Last but not least, there are also quite many studies in the field which claim that they design neural network solutions to work online, but essentially they are not truly online learning. They just adapt some batch learning algorithms to work efficiently for seqential learning environments, such as the series of Learning++ algorithms and their variants [213, 215]. Recently, Hedge Backpropagation [216] was proposed to learn deep neural networks in the online setting with the aim to address slow convergence of deep networks through dynamic depth adaptation.

4.11. Online Portfolio Selection

Online Portfolio Selection is the application of Online Learning to sequentially select a portfolio of stocks with the aim to optimize a certain metric (e.g. cumulative wealth, risk adjusted returns, etc.) [4, 217, 218, 219]. Consider a financial market with m assets, in which we have to allocate our wealth. At every time period (or iteration), the price of the m stocks changes by a factor of $\mathbf{x}_t \in \mathbb{R}_+^m$. This vector is also called the price relative vector. $x_{t,i}$ denotes the ratio of the closing price of asset i at time t to the last closing price at time $t-1$. Thus, an investment in asset i changes by a factor of $x_{t,i}$ in period t. At the beginning of time period t the investment is specified by a portfolio vector $\mathbf{b}_t \in \delta_m$ where $\delta_m = \mathbf{b} : \mathbf{b} \geq 0, \mathbf{b}^\top \mathbf{1} = 1$. The portfolio is updated in every time-period based on a specific strategy, and produces a sequence of mappings:

$$\mathbf{b}_1 = \frac{1}{m}, \quad \mathbf{b}_t : \mathbb{R}_+^{m(t-1)} \rightarrow \delta_m, \quad t = 2, 3, \ldots, T$$

where T is the maximum length of the investment horizon. To make a decision for constructing a portfolio at time t, the entire historical information from $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$ is available. The theoretical framework starts with a wealth of $S_0 = 1$, and at the end of every time period, the wealth changes as $S_t = S_{t-1} \times (\mathbf{b}_t^\top \mathbf{x}_t)$.

Most efforts in Online Portfolio Selection make a few (possibly unrealistic) assumptions, including: no transaction costs, perfectly liquid market, and no impact cost (the portfolio selection strategy does not affect the market). Besides the traditional benchmarking approaches, the approaches for online portoflio selection can be categorized into *Follow-the-winner, Follow-the-loser, Pattern Matching*, and *Meta-Learning* approaches [217].

The benchmark approaches, as the name suggests, are simple baseline methods whose performance can be used to benchmark the performance of proposed algorithms. Common baselines are Buy and Hold (BAH) strategy, Best Stock and Constant Rebalanced Portfolio (CRP). The idea of BAH is to start with a portfolio with equal investment in each asset, and never rebalance it. Best Stock is the performance of the asset with the highest returns at the end of the investment horizon. CRP [220] is a fixed portfolio allocation to which the portfolio is rebalanced to at the end of every period, and Best-CRP is the CRP which obtains the highest returns at the end of the investment horizon. It should be noted that Best Stock and Best-CRP strategies can only be executed in hindsight.

Follow-the-winner approaches adhere to the principle of increasing the relative portfolio allocation weight of the stocks that have performed well in the past. Many of the approaches are directly inspired by Convex Optimization theory, including Universal Portfolios [221], Exponential Gradient [222], Follow the Leader [223] and Follow the Regularized Leader[224]. In contrast to follow-the-winner, there is a set of approaches that aim to follow-the-loser, with the belief that asset prices have a tendency to revert back to a mean, i.e., if the asset price falls, it is likely to rise up in the next time-period. These are also called mean-reversion strategies. The early efforts in this category included Anti Correlation [225] which designed a strategy by betting making statistical bets on positive-lagged correlation and negative auto-correlation; and Passive-Aggressive Mean Reversion (PAMR) [226], which extended the Online Passive Aggressive Algorithms [41] to update the portfolio to an optimal "loser" portfolio - by selecting a portfolio that would have made an optimal loss in the last observed time-period. A similar idea was used to extend confidence-weighted online learning to develop Confidence-Weighted Mean Reversion [227, 228]. The idea of PAMR was extended to consider multi-period asset returns, which led to the development of Online Moving Average Reversion (OLMAR) [229, 230] and Robust Median Reversion (RMR) [231] strategies.

Another popular set of approaches is the Pattern-Matching approaches, which aim to find patterns (they may be able to exploit both follow-the-winner and follow-the-loser) for optimal sequential decision making. Most of these approaches are non-parametric. Ex-

emplar approaches include [232, 233, 234]. Finally, meta-learning algorithms for portfolio selection aim to rebalance the portfolio on the basis of the expert advice. There are a set of experts that output a portfolio vector, and the meta-learner uses this information to obtain the optimal portfolio. In general, the meta-learner adheres to the follow-the-winner principle to identify the best performing experts. Popular approaches in this category include Aggregating Algorithms [235], Fast Universalization Algorithm [236] and Follow the Leading History [237]. Besides these approaches, there are also efforts in portfolio selection with aims to optimize the returns accounting for transaction costs. The idea is to incorporate the given transaction cost into the optimization objective [238, 239, 240].

A closely related area to Online Portfolio Selection is Online Learning for Time Series Prediction. Time series analysis and prediction [241, 242, 243, 244] is a classical problem in machine learning, statistics, and data mining. The typical problem setting of time series prediction is as follows: a learner receives a temporal sequence of observations, $x_1, \ldots, x_t$, and the goal of the learner is to predict the future observations (e.g., x_{t+1} or onwards) as accurately as possible. In general, machine learning methods for time series prediction may also be divided into linear and non-linear, univariate and multivariate, and batch and online. Some time series prediction tasks may be resolved by adapting an existing batch learning algorithm using sliding window strategies. Recently there have been some emerging studies for exploring online learning algorithms for time series prediction [245, 246].

5. Bandit Online Learning

5.1. Overview

Bandit online learning, a.k.a. the multi-armed bandit problem [247, 248, 249, 250, 251, 252, 253] in statistics, is considered a branch of online learning methods where an online learner only receives partial feedback from the environment at every step.

A multi-armed bandit problem is an online learning task, where the player needs to choose one out of multiple actions to obtain some observable payoff. Its goal is to maximize the total payoff obtained in the online learning process. To make this problem more easily understandable, we will explain a bit more. The name bandit comes from American slang, one-armed bandit for a slot machine. So, when a player comes in a casino with many slot machines, he must repeatedly choose which machine to play, which is an intuitive example for multi-armed bandit problem. For multi-armed bandit problems, the player has to address the fundamental tradeoff between exploitation of actions that did well in the past and the exploration of actions that might give higher payoffs in the future. Multi-Armed Bandit (MAB) problems mainly consist of two formalization: *stochastic MAB* [254] and *adversarial MAB* [255].

Now, we will formally provide the procedure for stochatic Multi-Armed Bandits (MAB) problem. Stochastic MAB will takes place in a sequence of rounds with length $T \in \mathcal{N}$, which may not be disclosed at the begining. At the t-th round, the player will choose one action $I_t \in [k] = \{1, \ldots, k\}$, based on some learning strategy. Then the enviroment will draw the reward $X_{I_t,t}$ and reveal it to the forecaster. The regret after T plays $I_1, \ldots, I_T$ is defined by

$$R_T = \max_{i\in[k]} \sum_{t=1}^{T} X_{i,t} - \sum_{t=1}^{T} X_{I_t,t}$$

which is the difference between the cumulative payoff from the best arm and the one from the algorithm. Since, the rewards $X_{i,t}$ and the player's choices I_t might be stochastic. Two more generally used regrets are: the expected regret

$$\mathbb{E}R_T = \mathbb{E}\left[\max_{i\in[k]} \sum_{t=1}^{T} X_{i,t} - \sum_{t=1}^{T} X_{I_t,t}\right]$$

and the pseudo-regret

$$\bar{R}_T = \max_{i\in[k]} \mathbb{E}\left[\sum_{t=1}^{T} X_{i,t} - \sum_{t=1}^{T} X_{I_t,t}\right]$$

where, the expecation is taken with respect to the random draw of both rewards and forecster's actions. It is easy to note $\bar{R}_T \leq \mathbb{E}R_T$.

5.2. Stochastic Bandit

For stochastic Multi-Armed Bandits (MAB) problem, each arm $i \in [k]$ coreesponds to an unknown probability distributon P_i on $[0, 1]$, and the rewards $X_{i,t}$ are independent draws from the distribution P_i corresponding to the selected arm. Denote by μ_i the mean of the distribution P_i, and define

$$\mu_* = \max_{i\in[k]} \mu_i \quad \text{and} \quad \mathrm{i}_* \in \arg\max_{\mathrm{i}\in[\mathrm{k}]} \mu_{\mathrm{i}}$$

The goal of stochastic MAB is to minimize the pseudo-regret, i.e.,

$$\bar{R}_T = n\mu_* - \mathbb{E}\sum_{t=1}^{T} \mu_{I_t}$$

since it is more natural to compete against the optimal action in expectation.

5.2.1. Stochastic Multi-armed Bandit

To solve this stochastic MAB problem, the most well-know algorithm is the Upper Confidence Bound (UCB) algorithm [256]. To introduce this algorithm, we also use a different formula for the pseudo-regret. Specifically, let $N_i(s) = \sum_{t=1}^{s} \mathbb{I}(I_t = i)$ denote the number of times the player selected arm i on the first s iterations, and $\Delta_i = \mu_* - \mu_i$ be the suboptimality parameter of arm i. Then the pseudo-regret can be also re-written as:

$$\bar{R}_T = (\sum_{i=1}^{k} \mathbb{E}N_i(T))\mu_* - \mathbb{E}\sum_{i=1}^{k} N_i(T)\mu_i = \sum_{i=1}^{k} \Delta_i \mathbb{E}N_i(T).$$

Given this new pseudo-regret, we shall introduce UCB method. UCB is a strategy to simultaneously perform exploration and exploitation, which is based on a heuristic principle, optimism in face of uncertainty. In brief, this principle will prescribe that the arm which has the largest UCB should be selected. Formally, suppose the rewards X_i satisfy the following moment condition: there exists a convex function ϕ on the reals such that, for all $\lambda \geq 0$, and $i \in [k]$

$$\ln \mathbb{E}e^{\lambda(X_i - \mathbb{E}[X_i])} \leq \phi(\lambda), \text{ and } \ln \mathbb{E}\mathrm{e}^{\lambda(\mathbb{E}[\mathrm{X_i}]-\mathrm{X_i})} \leq \phi(\lambda).$$

For example, when $X_i \in [0, 1]$ one can take $\phi(\lambda) = \frac{\lambda^2}{8}$, which is known as Hoeffding' lemma. Under this assumption, we will estimate an upper confidence bound of the mean of each arm, and then choose the largest one. Specifically, let $\hat{\mu}_{i,s}$ be the sample mean of rewards obtained by pulling arm i for s times, it is easy to prove the following bound using Markov's inequality and the moment assumption:

$$\mathcal{P}(\mu_i - \hat{\mu}_{i,s} \geq \epsilon) \leq e^{-s\phi^*(\epsilon)},$$

where $\phi^*(\epsilon) = \sup_\lambda(\lambda\epsilon - \phi(\lambda)$ is the conjugate function of ϕ. This inequality implies, with probability at least $1 - \delta$, the following inequality holds

$$\hat{\mu}_{i,s} + (\phi^*)^{-1}\left(\frac{1}{s}\ln\frac{1}{\delta}\right) > \mu_i.$$

Thus, UCB method, considering the following strategy: at time t, selects

$$I_t \in \arg\max_{i\in[k]} \left[\hat{\mu}_{i,N_i(t-1)} + (\phi^*)^{-1}\left(\frac{\alpha \ln t}{N_i(t-1)}\right)\right]$$

where $\alpha > 0$ is a parameter. This algorithm is termed as (α, ϕ)- Upper Confidence Bound ((α, ϕ)-UCB) algorithm, which is summarized in Algorithm 13:

For this algorithm, we have the followign theorem

Algorithm 13: UCB

INPUT: number of arms k, number of iterations $T \geq k$, function ϕ
INIT:$\alpha > 0$, $\hat{\mu}_{i,N_i(0)} = 0$, $N_i(0) = 0$, $\forall i \in [k]$
for $t = 1, 2, \ldots, T$ **do**
  The player chooses
  $I_t \in \arg\max_{i\in[k]} \left[\hat{\mu}_{i,N_i(t-1)} + (\phi^*)^{-1}\left(\frac{\alpha \ln t}{N_i(t-1)}\right)\right]$
  The environment draws the reward $X_{I_t,t} \sim P_{I_t}$ independently from the past and reveals it to the player.
  The player update
  $\hat{\mu}_{i,N_i(t)} = \frac{N_i(t-1)}{N_i(t-1)+1}\hat{\mu}_{i,N_i(t-1)} + \frac{1}{N_i(t-1)+1}X_{I_t,t}$,
  $N_i(t) = N_i(t-1) + 1$ for $i = I_t$.
  The player update $\hat{\mu}_{i,N_i(t)} = \hat{\mu}_{i,N_i(t-1)}$,
  $N_i(t) = N_i(t-1)$ for $i \neq I_t$.
end for

Theorem 2. *Assume that the above moment assumption holds, i.e., there exists a convex function ϕ on the reals such that, for all $\lambda \geq 0$, and $i \in [k]$, $\ln \mathbb{E}e^{\lambda(X_i - \mathbb{E}[X_i])} \leq \phi(\lambda)$, and $\ln \mathbb{E}\mathrm{e}^{\lambda(\mathbb{E}[\mathrm{X_i}]-\mathrm{X_i})} \leq \phi(\lambda)$. Then this (α, ϕ)-UCB algorithm with $\alpha > 2$ satisfys the following regret bound*

$$\bar{R}_T \leq \sum_{i\in\{j|\Delta_j>0\}} \left(\frac{\alpha\Delta_i}{\phi^*(\Delta_i/2)}\ln T + \frac{\alpha}{\alpha - 2}\right).$$

When $\phi(\lambda) = \frac{\lambda^2}{8}$, and $X_{i,t} \in [0, 1]$, this algorithm is usually termed as α-UCB for short.

5.2.2. Stochastic Combinatorial Bandit

Combinatorial bandit was introduced in [257].

We first introduce the problem setting of the *linear bandit optimization* problem. During each iteration, the player makes its decision by choosing a vector from a finite set $\mathcal{S} \subseteq \mathbb{R}^d$ of elements $\mathbf{v}(i)$ for $i = 1, ..., k$. The chosen action at iteration t is indexed as I_t. The environment chooses a loss vector $\boldsymbol{\ell}_t \in \mathbb{R}^d$ and returns the linear loss as $c_t(I_t) = \boldsymbol{\ell}_t^\top \mathbf{v}(I_t)$. Note that the player has no access to the full knowledge of loss vector and the only information revealed to the player is the loss of its own decision $c_t(I_t)$. Obviously, when setting $d = k$ and $\mathbf{v}(i)$ is the standard basis vector, this problem is identical to that in the previous section.

The *combinatorial bandit* problem is a special case of linear bandit optimization where $\mathcal{S}$ is a subset of the binary hypercube $\{0, 1\}^d$. The loss vector $\boldsymbol{\ell}_t$ may be generated from an unknown but fixed distribution, which is termed as stochastic combinatorial bandit, or chosen from some adversarial environment, which is termed as adversarial combinatorial bandit. In this section, we

will focus on the first one. The goal of Stochastic Combinatorial Bandit is to minimize the expected regret,

$$\bar{R}_T = \mathbb{E}[\sum_{t=1}^{T} c_t(I_t)] - \min_{i\in[k]} L_T(i)$$

where $L_T(i) = \mathbb{E}\sum_{t=1}^{T} c_t(i)$ is the expected sum of loss for choosing action i in all T iterations, not a random variable in stochastic setting.

COMBAND. The COMBAND algorithm [257] works as follows. First, a sampling probability vector $\mathbf{p}_t \in \mathbb{R}^k$ is defined for sampling $\mathbf{v}(I_t)$ from $\mathcal{S}$,

$$\mathbf{p}_t = (1-r)\mathbf{q}_t + \gamma\boldsymbol{\mu}$$

where $\mathbf{q}_t \in \mathbb{R}^k$ is the exploitation probability vector that is updated during all iterations to follow the best action and $\boldsymbol{\mu} \in \mathbb{R}^d$ is a fixed exploration probability. $\gamma \in [0,1]$ is the weight that controls the exploitation and exploration trade-off. The algorithm draws the action I_t based on distribution $\mathbf{p}_t$ and get the loss $c_t(I_t)$ from the environment.

Second, an estimation of the loss vector $\boldsymbol{\ell}_t$ is build with the new information,

$$\widetilde{\boldsymbol{\ell}}_t = c_t(I_t)P_t^{+}\mathbf{v}(I_t)$$

where P_t^{+} is the pseudo-inverse of the expected correlation matrix $\mathbb{E}_{\mathbf{p}_t}[\mathbf{v}\mathbf{v}^\top]$.

Finally, the exploitation weights are scaled based on the estimated loss vector,

$$\mathbf{q}_{t+1}(i) \propto \mathbf{q}_t(i)\exp(-\eta\widetilde{\boldsymbol{\ell}}_t^\top\mathbf{v}(i))$$

where $\eta > 0$ is a learning rate parameter and $\propto$ indicates that this scaling step is followed by a normalization step so that $\sum_{i=1}^{k}\mathbf{q}(i) = 1$.

The COMBAND algorithm achieves a regret bound better than $O(\sqrt{Td\ln|\mathcal{S}|})$ for a variety of concrete choices of $\mathcal{S}$.

Other Related Works. Recently, many related works also address the combinatorial bandit in different manners. The ESCB algorithm [258] efficiently exploits the structure of the problem and gets a better regret bound of $O(\ln(T))$. The CUCB algorithm [259] contributes to the problem where the loss function may be nonlinear.

5.2.3. Stochastic Contextual Bandit

A widely used extension of multi-armed bandits problem is to associate contextual information with each arm [260]. For example, in personalized recommendation problem, the task is to select products that are most likely to be purchased by a user. In this case, each product corresponds to an arm and the features of each product are easy to acquire [261]. There are many comprehensive surveys on contextual bandit algorithms, both in stochastic and adversarial setting [262, 255]. In this section, we will focus on stochastic setting.

In a contextual bandits problem, there is a set of policies $\mathcal{F}$, which may be finite or infinite. Each $f \in \mathcal{F}$ maps a context $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ to an arm $i \in [k]$. Different from the previous setting where the regret is defined by competing with the arm with the highest expected reward, the regret here is defined by comparing the decision I_t with the best policy $f^* = \arg\inf_{f\in\mathcal{F}}\ell_D(f)$, where D is the data distribution.

$$R_T(f) = \sum_{t=1}^{T}[\ell_{I_t,t} - \ell_t(f^*)]$$

LinUCB [263] is an extension of the UCB algorithm to contextual bandit problem. The algorithm assumes that there is a feature vector $\mathbf{x}_{t,i} \in \mathbb{R}^d$ at time t for each arm i. Similar to the UCB algorithm, a model is learnt to estimate the upper confidence bound of each arm $i \in [k]$ given the input of $\mathbf{x}_{t,i}$. The algorithm simply chooses the arm with the highest UCB. The LinREL algorithm [264] is similar to LinUCB in that it adopts the same problem setting and same maximizing UCB strategy. While, a different regularization term is used which leads to a different calculation of the UCB.

Contextual Bandit problem can also be regarded as a special case of the online multi-class classification problem. The goal is to learn a mapping from the context space $\mathbb{R}^d$ to the label space $\{1,...,k\}$ from a sequence of instances $\mathbf{x}_t \in \mathbb{R}^d$. Different from regular setting of online multi-class classification problems where a class label $y_t \in \{1,...,k\}$ is revealed at the end of each iteration, in bandit setting, the learner can only get a partial feedback on whether the prediction $\hat{y}_t$ equals to y_t. In the following, we briefly review several representative works of bandit multi-class classification algorithms.

Banditron. Banditron is the first bandit algorithm for online multiclass prediction [265], which is a variant of the Perceptron. To efficiently make prediction and update the model, the Banditron algorithm keep a linear model W^t, which is initialized as $W^1 = 0 \in \mathbb{R}^{k\times d}$. At the t-th iteration, after receiving the instance $\mathbf{x}_t \in \mathbb{R}^d$, it will first set

$$\hat{y}_t = \arg\max_{r\in[k]}(W^t\mathbf{x}_t)_r$$

where $(\mathbf{z})_r$ denotes the r-th element of $\mathbf{z}$. Then the algorithm will define a distribution as

$$\Pr(r) = (1-\gamma)\mathbb{I}(r = \hat{y}_t) + \gamma/k,\ \forall r \in [k]$$

which roughly implies that the algorithm exploits with probability $1-\gamma$ and explores with the remaining probability by uniformly predicting a random label from $[k]$. The parameter γ controls the exploration-exploitation tradeoff. The algorithm then randomly sample $\tilde{y}_t$ according to the probability Pr and predicts it as the label of $\mathbf{x}_t$. After the prediction, the algorithm then receives the bandit feedback $\mathbb{I}(\tilde{y}_t = y_t)$. Then the algorithm uses this feedback to construct a matrix

$$\tilde{U}^t_{r,j} = x_{t,j}\left(\mathbb{I}(\hat{y}_t = r) - \frac{\mathbb{I}(\tilde{y}_t = y_t)\mathbb{I}(\tilde{y}_t = r)}{\Pr(r)}\right)$$

since its expectation satisfies $\mathbb{E}\tilde{U}^t_{r,j} = U^t_{r,j} = x_{t,j}\left(\mathbb{I}(\hat{y}_t = r) - \mathbb{I}(y_t = r)\right)$, where U^t is actually a (sub)-gradient of the following hinge loss

$$\ell(W;(\mathbf{x}_t, y_t)) = \max_{r\in[k]/\ \{y_t\}} [1 - (W\mathbf{x}_t)_{y_t} + (W\mathbf{x}_t)_r]_+$$

where $[z]_+ = \max(0, z)$. Then the algorithm will update the model using

$$W^{t+1} = W^t - \tilde{U}^t$$

We summarize the Banditron algorithm as follows:

Algorithm 14: Banditron

INIT: $\mathbf{w}_{1,1} = 0, .., \mathbf{w}_{k,1} = 0$
for $t = 1, 2, \ldots, T$ **do**
  Receive an incoming instance $\mathbf{x}_t$
  $P(r) = (1-\gamma)\mathbf{1}[r = \arg\max_i \mathbf{w}_{i,t}^\top \mathbf{x}_t] + \frac{\gamma}{k}$.
  Sample $\hat{y}_t$ according to $P(r), r \in \{1, ..., k\}$
  $\mathbf{u}_r = \mathbf{x}_t(\frac{\mathbf{1}[y_t=\hat{y}_t=r]}{P(r)} - \mathbf{1}[r = \arg\max_i \mathbf{w}_{i,t}^\top \mathbf{x}_t])$;
  $\mathbf{w}_{r,t+1} = \mathbf{w}_{r,t} + \mathbf{u}_r$
end for

This algorithm achieves $O(\sqrt{T})$ in linear separable case and $O(T^{\frac{2}{3}})$ in inseparable case.

Bandit Passive Aggressive.. Different from the Banditron and bandit EG, bandit Passive Aggressive (Bandit PA) adopts the framework of one vs all others to make prediction and update the model [266]. Specifically, the algorithm keeps an matrix M, whose diagonal elements are 1 and off-diagonal elements are -1, and a matrix $W^t = (\mathbf{w}_1^t, \ldots, \mathbf{w}_k^t)$, which are initialized as zero matrix. At the t-th iteration, the bandit PA predicts the label of $\mathbf{x}_t$ as

$$\hat{y}_t = \arg\min_r \sum_{s=1}^{k} [1 - M(r,s)\mathbf{x}_t^\top \mathbf{w}_s^t]_+$$

which encourage larger $\mathbf{x}_t^\top \mathbf{w}_{y_t}^t$ and smaller $\mathbf{x}_t^\top \mathbf{w}_s^t$, $s \neq y_t$. After prediction, the algorithm will receive the bandit feedback $\mathbb{I}(\hat{y}_t = y_t)$. If $\hat{y}_{tt} = y_t$, then this feedback is actually a full one, so the algorithm can update the model using the standard PA algorithm,

$$\mathbf{w}_s^{t+1} = \mathbf{w}_s^t + \tau_t M(y_t, s)\mathbf{x}_t$$

where $\tau_t = \ell_t/\|\mathbf{x}_t\|^2$ (basic PA), $\tau_t = \min(C, \ell_t/\|\mathbf{x}_t\|^2)$ (PA-I), or $\tau_t = \ell_t/[\|\mathbf{x}_t\|^2 + \frac{1}{2C}$ (PA-II), and $\ell_t = [1 - M(\hat{y}_t, s)\mathbf{x}_t^\top \mathbf{w}_s^t]_+$. Otherwise, only the $\mathbf{w}_{\hat{y}_t}^t$ will be updated by

$$\mathbf{w}_{\hat{y}_t}^{t+1} = \mathbf{w}_{\hat{y}_t}^t M(y_t, \hat{y}_t)\tau_t \mathbf{x}_t.$$

since we only know $M(y_t, s) = -1$ for $s = \hat{y}_t$.

Following the Banditron, many algorithms have been proposed to address the online multi-class classification in bandit setting. Some updates in first order gradient descent [267], others in second order learning [268, 269, 254, 270]. Most of these algorithms explore the k classes uniformly with probability γ, while [268] sample the classes based on the Upper Confidence Bound.

5.3. Adversarial Bandit

In the previous sections of stochastic setting, we assumed that the rewards are drawn from an unknown but fixed distribution. Here, we will not have the stochastic assumption on the reward. Instead, the reward distribution can be affected by the previous actions of the player, which is termed as the Adversarial Bandit problem.

5.3.1. Adversarial Multi-armed Bandit

We first define the regret of an adversarial multi-armed bandit problem for T iterations,

$$R_T = \sum_{t=1}^{T} \ell_{I_t,t} - \min_{i=1,\ldots k} \sum_{t=1}^{T} \ell_{i,t}$$

in which we are comparing the player's action with the best fixed arm. The goal is to achieve a sublinear bound with regards to T uniformly over all possible adversarial assignments of gains to arms. Since $\ell_{i,t}$ depends on the previous actions $I_{i,\tau}, \tau \in \{1, ...t-1\}$ and might be adversarial, this goal is impossible for any fixed strategy.

A effective idea is to surprise the adversary by adding randomization to I_t. The goal becomes minimizing the pseudo-regret,

$$\bar{R}_T = \mathbb{E}\sum_{t=1}^{T} \ell_{I_t,t} - \min_{i=1,\ldots k} \mathbb{E}\sum_{t=1}^{T} \ell_{i,t}$$

Exp3. The Exponential weights for Exploration and Exploitation algorithm (Exp3) [271] is a landmark in the Adversarial Multi-armed Bandit field.

We first define a probability vector $\mathbf{p}_t \in \mathbb{R}^k$ in which the $i-$th element $p_{i,t}$ indicates the probability of drawing arm i at time t. This vector is initialized uniformly and updated in each iteration. After drawing $I_t \sim \mathbf{p}_t$, we can get an unbiased estimator of the k loss function,

$$\widetilde{\ell_{i,t}} = \frac{\ell_{i,t}}{p_{i,t}} \mathbb{I}_{I_t=i}$$

Finally, the probability vector is updated according to the accumulated loss function of each arm

$$p_{i,t+1} = \frac{\exp(-\eta_t \widetilde{L}_{i,t})}{\sum_{j=1}^k \exp(-\eta_t \widetilde{L}_{j,t})}$$

where the $\widetilde{L}_{i,t} = \sum_{\tau=1}^t \widetilde{\ell_{i,t}}$ is the estimate of accumulated loss function and $\eta_t > 0$ is a parameter that controls the exploitation and exploration trade-off.

The Exp3 algorithm achieves $O(\sqrt{Tk \ln k})$ pseudo-regret in adversarial setting.

Other Related Works. As a more challenging problem, this area is not extensively studied compared with stochastic setting. However, some algorithms are still available in literature [255]. The Exp3.P algorithm [271] improves the loss estimation and probability update strategies to get a high probability bound. The Exp3.M algorithm [272] explores the new problem setting of multiple plays.

5.3.2. Adversarial Combinatorial Bandit

As introduced in the Stochastic Combinatorial Bandit section, Combinatorial Bandit section is a special case of linear bandit optimization where elements in all k decision vectors $\mathbf{v}(i), i \in [k]$ are binary value. Different from the stochastic setting in previous section, where the loss vector $\boldsymbol{\ell}_t$ is assumed to be generated from a fixed distribution, here we assume that the loss vector $\boldsymbol{\ell}_t$ is generated by the adversarial environment.

Adversarial Combinatorial Bandit has been extensively studied in early days [273]. Recently, [258] provided a useful literature survey for closely related works [274, 257] and proposed a novel algorithm with promising bounds.

5.3.3. Adversarial Contextual Bandit

In Adversarial Contextual Bandit problems, each arm is associated with some side information and the reward of each arm does not follow a fixed distribution. Moreover, the reward can be set by an adversary against the player. In the following, we will briefly introduce the most representative work in this field, Exponential-weight Algorithm for Exploration and Exploitation using Expert advice (Exp4) [271].

The Exp4 algorithm assumes that there are N experts who will give advice on the distribution over arms during all iterations. $\xi_{i,t}^n$ indicates the probability of picking arm $i \in [k]$ recommended by expert $n \in [N]$ during time $t \in [T]$. Obviously, $\sum_{i=1}^k \xi_{i,t}^n = 1$. During time t, the true reward vector is denoted by $\mathbf{r}_t \in [0,1]^k$. Thus the expected reward of expert n is $\boldsymbol{\xi}_t^n \cdot \mathbf{r}_t$. The regret is defined by comparing with the expert with the highest expected cumulative reward.

$$R_t = \max_{n\in[N]} \sum_{t=1}^T \boldsymbol{\xi}_t^n \cdot \mathbf{r}_t - \mathbb{E} \sum_{t=1}^T r_{t,I_t}$$

The Exp4 algorithm first defines a weight vector $\mathbf{w}_t \in \mathbb{R}^N$ that indicates the weights for the N experts. We set the weight as $\mathbf{w}_0 = \mathbf{1}$ and update it during each iteration.

During iteration t, we calculate the probability of picking arm i as the weighted sum of advices from all N experts,

$$p_{i,t} = (1-\gamma) \frac{\sum_{n=1}^N w_{n,t} \xi_{i,t}^n}{\sum_{n=1}^N w_{n,t}} + \frac{\gamma}{K}$$

where $\gamma \in [0,1]$ is the weight parameter that controls the exploitation and exploration trade-off. We then draw the arm I_t according to the probability $p_{i,t}$ and calculate an unbiased estimator of

$$\hat{r}_{i,t} = \frac{r_{i,t}}{p_{i,t}} \mathbb{I}_{i=I_t}$$

which will be used to calculate the expected reward. Finally the weight $\mathbf{w}_t$ is updated according to the expected reward of each arm.

The Exp4 algorithm achieves the regret bound of $O(\sqrt{Tk \ln N})$

Other Related Works. There are many related algorithms in the field of Adversarial Contextual Bandit, which can be in [262]. An important extension to Exp3 algorithm is the Exp4.P algorithm [275], which adopts a small modification to the weight update strategy and achieves the same regret with high probability.

6. Online Active Learning

6.1. Overview

In this section, we will introduce online active learning. The basic process of active online learning works in iterations. At each iteration, one unlabelled instance

is presented to the learner, and the learner needs to decide whether to query its label. If the label is queried, then the learner can use the labelled instance to update the model, otherwise the model is kept unchanged.

Specifically, there are two kinds of settings for active online learning. One is selective sampling setting [276], and the other is label efficient learning setting. There are several key differences between these two settings. Firstly, in the selective sampling setting the instances are drawn randomly from a fixed distribution, while in the label efficient setting the instances can be generated adversarially. Secondly, the label efficient model must make predictions on those instances where the label is not requested, while the selective sampling models are concerned with the generalization error rather than the performance of the algorithm on the sequence of instances.

6.2. Selective Sampling Algorithms

Margin-based Selective Sampling Algorithm. It is assumed that, the instances $\mathbf{x}_t$, $t \in [T]$ are drawn independently from a fixed and unknown distribution on the surface of the unit Euclidean sphere in $\mathbb{R}^d$, so that $\|\mathbf{x}_t\| = 1$. The label y_t of $\mathbf{x}_t$ is drawn from $\{-1, +1\}$ with $\Pr(y_t = 1) = (1 + \mathbf{w}^\top \mathbf{x}_t)/2$, where $\mathbf{w} \in \mathbb{R}^d$ is fixed and unknown with $\|\mathbf{w}\| = 1$. Under these assumptions $\text{sign}(\mathbf{w}^\top \mathbf{x}_t)$ is the Bayes optimal classifier for this noise model.

This algorithm [277] has two stages: the first one is all the steps before N-th step, where N is parameter which will be explained later; the second one is all the steps after N-th steps. At the t-th step, the algorithm will use the following rule to predict the label of $\mathbf{x}_t$ as

$$\hat{y}_t = \text{sign}(p_t), \quad \text{where,} \quad p_t = \mathbf{w}_t^\top \mathbf{x}_t$$

where $\mathbf{w}_t = A_t^{-1}\mathbf{u}_t$, $\mathbf{u}_t = \sum_{i=1}^{t-1} Z_i y_i \mathbf{x}_i^\top$ and $A_t = (I + \sum_{i=1}^{t-1} Z_i \mathbf{x}_i \mathbf{x}_i^\top)$ with I be the identity matrix. After prediction, if it is at the first stage, the algorithm will set $Z_t = 1$ which means the label y_t is queried for updating the model; if it is at the second stage, the algorithm will set:

$$\Pr(Z_{t+1} = 1) = \mathbb{I}(p_t \le \frac{4 \ln t}{\sum_{i=1}^{t-1} Z_i})$$

Let λ be the minimal eigenvalue of the process covariance matrix $\{\mathbb{E}[x_i, x_j]\}_{i,j=1}^d$ and $N = \lceil \max(96d, 912 \ln T)/\lambda^2 \rceil$. Then the cumulative regret of this algorithm is bounded as

$$\sum_{t=1}^{T} [\Pr(y_t \mathbf{w}_t^\top \mathbf{x}_t \le 0) - \Pr(y_t \mathbf{w}^\top \mathbf{x}_t \le 0)]$$
$$\le N + \mathbb{E}L + 4 \ln T = \mathbb{E}L + O(\frac{d + \ln T}{\lambda^2})$$

where L is the number of queried labels during the second stage. L satisfies $\mathbb{E}L \le \mathbb{E}[\frac{16 \ln T}{\lambda(\min_t \mathbf{w}_t^\top \mathbf{x}_t)^2}] + 4$

BBQ: Bound on Bias Query.. In this algorithm [278], it is assumed that $\|\mathbf{x}_t\| = 1$ for all $t \ge 1$ and the corresponding labels $y_t \in \{-1, +1\}$ are realizations of random variables Y_t such that $\mathbb{E}Y_t = \mathbf{w}^\top \mathbf{x}_t$ for all $t \ge 1$, where $\mathbf{w} \in \mathbb{R}^d$ is a fixed and unknown vector such that $\|\mathbf{w}\| = 1$. Under these assumptions $\text{sign}(\mathbf{w}^\top \mathbf{x}_t)$ is the Bayes optimal classifier for this noise model.

This algorithm keeps a vector $\mathbf{u}_t = \sum_{i=1}^{t-1} Z_i y_i \mathbf{x}_i$, and a matrix $A_t = I + \sum_{i=1}^{t-1} Z_i \mathbf{x}_i \mathbf{x}_i^\top$ which is the sum of identity matrix I and the correlation matrix over the queried instances, then this algorithm predicts the label of the current instance $\mathbf{x}_t$ as

$$\hat{y}_t = \text{sign}(p_t), \text{where} \quad p_t = \mathbf{w}_t^\top \mathbf{x}_t$$

where $\mathbf{w}_t = (A_t + \mathbf{x}_t \mathbf{x}_t^\top)^{-1} \mathbf{u}_t$.

After prediction, the algorithm will query the label of y_t using

$$\Pr(Z_t = 1) = \mathbb{I}\left(\mathbf{x}_t^\top (A_t + \mathbf{x}_t \mathbf{x}_t^\top)^{-1} \mathbf{x}_t > t^{-k}\right)$$

If $Z_t = 1$, the label y_t will be requested, and the model will be updated by

$$\mathbf{u}_{t+1} = \mathbf{u}_t + Z_t y_t \mathbf{x}_t, \quad A_{t+1} = A_t + Z_t \mathbf{x}_t \mathbf{x}_t^\top$$

Let $T_\epsilon = |\{1 \le t \le T | \mathbf{w}^\top \mathbf{x}_t| \le \epsilon\}|$ be the number of examples with margin less than ϵ. Then if BBQ is run with input $k \in [0, 1]$, its cumulative regret can be bounded as follows:

$$\sum_{t=1}^{T} \left[\Pr(y_t \mathbf{w}_t^\top \mathbf{x}_t \le 0) - \Pr(y_t \mathbf{w}^\top \mathbf{x}_t \le 0)\right]$$
$$\le \min_{\epsilon \in [0,1]} \left(\epsilon T_\epsilon + (2 + e)\lceil 1/k \rceil (\frac{8}{\epsilon^2})^{1/k} \right.$$
$$\left. + (1 + \frac{2}{e}) \frac{8d}{\epsilon^2} \ln(1 + \frac{\sum_{t=1}^{T} Z_t}{d}) \right)$$

The number of queried labels is $O(dT^k \ln T)$

Parametric BBQ. : In BBQ, the value ϵ is the unknown optimal one. However, if we set it as parameter, we can get a different query strategy. Specifically, we can provide two parameters $\epsilon, \delta \in (0, 1)$ to the parametric BBQ algorithm [278]. Then the query strategy is designed as

$$\Pr(Z_t = 1)$$
$$= \mathbb{I}\left(\max(0, \epsilon - r_t - s_t) < \|\mathbf{q}_t\| \sqrt{2 \ln \frac{t(t+1)}{2\delta}}\right)$$

where $r_t = \mathbf{x}_t^\top (A_t + \mathbf{x}_t \mathbf{x}_t^\top)^{-1} \mathbf{x}_t$, $s_t = \|A_t^{-1} \mathbf{x}_t\|$, and $\mathbf{q}_t = S_{t-1}^\top (A_t + \mathbf{x}_t \mathbf{x}_t^\top)^{-1} \mathbf{x}_t$, with $S_{t-1} = [\mathbf{x}'_1, \ldots, \mathbf{x}'_{N_{t-1}}]$ which is

the matrix of the queried instances up to time $t-1$. For this parametric version of BBQ,

$$\Pr(|\mathbf{w}_t^\top \mathbf{x}_t - \mathbf{w}^\top \mathbf{x}_t| \le \epsilon) \ge 1-\delta$$

holds on all time steps t when no query is issued. The number of queries issued can be bounded by $O(\frac{d}{\epsilon^2}(\ln\frac{T}{\delta})\ln\frac{\ln(T/\delta)}{\epsilon})$

DGS: Dekel Gentile Sridharan.. In this algorithm [279], it is assumed that $\|\mathbf{x}_t\| \le 1$ for all $t \ge 1$ and the corresponding labels $y_t \in \{-1,+1\}$ are realizations of random variables Y_t such that $\mathbb{E}Y_t = \mathbf{w}^\top \mathbf{x}_t$ for all $t \ge 1$, where $\mathbf{w} \in \mathbb{R}^d$ is a fixed and unknown vector such that $\|\mathbf{w}\| \le 1$. Under these assumptions $\text{sign}(\mathbf{w}^\top \mathbf{x}_t)$ is the Bayes optimal classifier for this noise model.

Similar with BBQ, DGS maintains a weight vector $\mathbf{w}_t$ (initialized as 0) and a data correlation matrix A_t (initialized as I). After receiving $\mathbf{x}_t$ and predicting

$$\hat{y}_t = \text{sign}(p_t), \quad \text{where} \quad \hat{p}_t = \mathbf{w}_t^\top \mathbf{x}_t$$

the algorithm computes an adaptive data-dependent threshold θ_t, defined as

$$\theta_t^2 = \mathbf{x}_t^\top A_t^{-1}\mathbf{x}_t(1+4\sum_{i=1}^{t-1} Z_i r_i + 36\log\frac{t}{\delta})$$

where $r_i = \mathbf{x}_i^\top A_{i+1}^{-1}\mathbf{x}_i$. The definition of θ_t can be interpreted as the algorithm's uncertainty in its own predictions. The algorithm then queries the label of $\mathbf{x}_t$ by using

$$\Pr(Z_t = 1) = \mathbb{I}\left(|\mathbf{w}_t^\top \mathbf{x}_t| \le \theta_t\right)$$

If $Z_t = 1$, i.e., y_t is queried, then the algorithm will firstly update the model by

$$\mathbf{w}_{t+\frac{1}{2}} = \mathbf{w}_t - \mathbb{I}(|\hat{p}_t| > 1)\text{sign}(\hat{p}_t)(\frac{|\hat{p}_t - 1|}{\mathbf{x}_t^\top A_t^{-1}\mathbf{x}_t}))A_t^{-1}\mathbf{x}_t$$

and

$$\mathbf{w}_{t+1} = A_{t+1}^{-1}(A_t\mathbf{w}_{t+\frac{1}{2}} + y_t\mathbf{x}_t), \quad \text{where} \quad A_{t+1} = A_t + \mathbf{x}_t\mathbf{x}_t^\top$$

Theoretically, if we assume that DGS is run with confidence parameter $\delta \in (0,1]$, then with probability $\ge 1-\delta$ it holds that for all $T > 0$ that

$$\Pr(y_t\mathbf{w}_t^\top \mathbf{x}_t \le 0) - \Pr(y_t\mathbf{w}^\top \mathbf{x}_t \le 0)$$
$$\le \inf_{\epsilon>0}[\epsilon T_\epsilon + O(\frac{d\ln T + \ln(T/\delta)}{\epsilon})]$$

and the number of queried labels is bounded by $\inf_{\epsilon>0}[T_\epsilon + O(\frac{d^2\ln^2(T/\delta)}{\epsilon^2})]$

DGS-Mod: a modified DGS algorithm.. Different with DGS, a parameter $\alpha > 0$ is introduced in the query rule to trade off regret against queries in a smooth way [280]. Specifically the value θ_t is defined in a different way

$$\theta_t^2 = 2\alpha(\mathbf{x}_t^\top A_t^{-1}\mathbf{x}_t)\ln t(4\sum_{s=1}^{t-1} Z_s r_s + 36\ln(t/\delta)$$

where $r_s = \mathbf{x}_s^\top A_{s+1}^{-1}\mathbf{x}_s$. Then the algorithm will query the label of $\mathbf{x}_t$ by using

$$\Pr(Z_t = 1) = \mathbb{I}\left(|\mathbf{w}_t^\top \mathbf{x}_t| \le \theta_t\right)$$

The update strategy for the model is the same with DGS algorithm. For this modified version, after any number of steps T , with probability at least $1-\delta$, the cumulative regret satisfies

$$\Pr(y_t\mathbf{w}_t^\top \mathbf{x}_t \le 0) - \Pr(y_t\mathbf{w}^\top \mathbf{x}_t \le 0)$$
$$\le \min_{\epsilon\in(0,1)}[1+\epsilon T_\epsilon + \frac{2}{3}\exp[\frac{1}{\alpha}(\frac{\|\mathbf{w}\|^2}{24}+1)]$$
$$+\frac{1}{\epsilon}(2\|\mathbf{w}\|^2 + 8\ln|A_{T+1}| + 144\ln\frac{T}{\delta})]$$

If $X \ge \max_t \|\mathbf{x}_t\|$ the number of queried label satisfies is bounded by

$$1 + T_\epsilon + \frac{4(1+X^2)}{\epsilon^2}\ln|A_{T+1}|[\|\mathbf{w}\|^2$$
$$+(1+2\alpha\ln T)(4\ln|A_{T+1}| + 36\ln\frac{T}{\delta})]$$

6.3. Label Efficient Algorithms

Label Efficient Perceptron. At the t-th iteration, an unlabelled instance $\mathbf{x}_t$ is presented. The algorithm first makes a prediction as $\hat{y}_t = \text{sign}(\hat{p}_t)$, where $\hat{p}_t = \mathbf{w}_t^\top \mathbf{x}_t$. Then the algorithm [281] decides whether to ask for the label y_t through a simple randomized rule: draw a Bernoulli random variable $Z_t \in \{0,1\}$ with

$$\Pr(Z_t = 1) = \frac{\delta}{\delta + |\hat{p}_t|}$$

where $\delta > 0$ is a smooth parameter. The parameter δ can be used to control the number of labels queried during the online learning process. If δ increases, the number of queried labels will also increase. If $Z_t = 1$, then the label y_t of $\mathbf{x}_t$ will be queried. If the the algorithm makes a mistake, i.e., $\hat{y}_t \ne y_t$, then the model is updated by using the Perceptron additive rule, i.e.,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_t\mathbf{x}_t.$$

On the other hand, if either $Z_t = 0$ or $\hat{y}_t = y_t$, no update will take place.

Denote $M_t = \mathbb{I}(\hat{y}_t \neq y_t)$, where $\mathbb{I}$ is the indicator function. Then given a sequence of examples $\{(\mathbf{x}_t, y_t)|t \in [T]\}$, the summation $\sum_{t=1}^{T} M_t$ is the (random) number of mistakes of the proposed algorithm. If assume $\|\mathbf{x}_t\| \leq R$, then for any $\mathbf{w} \in \mathbb{R}^d$, the expected number of mistakes of the algorithm can be bounded as follows:

$$\mathbb{E}[\sum_{t=1}^{T} M_t] \leq (1 + \frac{R^2}{2\delta})\frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{\|\mathbf{w}\|^2(2\delta + R^2)^2}{8\delta\gamma^2}.$$

where $\bar{L}_{\gamma,T}(\mathbf{w}) = \mathbb{E}[\sum_{t=1}^{T} Z_t M_t \ell_{\gamma,t}(\mathbf{w})]$, and $\ell_{\gamma,t}(\mathbf{w}) = \max(0, \gamma - y_t \mathbf{w}^\top \mathbf{x}_t)$.

Furthermore, the expected number of labels queried by the algorithm equals $\sum_{t=1}^{T} \mathbb{E}[\frac{\delta}{\delta+|\hat{p}_t|}]$. This bound depends on the value of the parameter δ. The optimal value of δ is

$$\delta = \frac{R^2}{2}\sqrt{1 + \frac{4\gamma^2}{\|\mathbf{w}\|^2 R^2}\frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma}}$$

and the bound on the expected number of mistakes becomes

$$\frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{\|\mathbf{w}\|^2 R^2}{2\gamma^2} + \frac{\|\mathbf{w}\|R}{\gamma}\sqrt{\frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{\|\mathbf{w}\|^2 R^2}{4\gamma^2}}$$

This is an expectation version of the mistake bound for the standard Perceptron Algorithm. Especially, in the special case when the data is linearly separable, the optimal value of δ is $R^2/2$ and this bound becomes the familiar Perceptron bound $(\|\mathbf{w}\|R)^2/\gamma^2$.

Adaptive Label Efficient Perceptron.. This algorithm [282] is to learn the best trade-off parameter δ in an online fashion without relying on prior knowledge on the sequence of examples, including the value $R > \|\mathbf{x}_t\|$. The algorithm follows the "self-confident" approach. Specifically, the algorithm predict $\hat{y}_t = \text{sign}(\hat{p}_t)$ where $\hat{p}_t = \mathbf{w}_t^\top \mathbf{x}_t$. Then, it will draw a Bernoulli random variable $Z_t \in \{0, 1\}$ with

$$\Pr(Z_t = 1) = \frac{\delta_t}{\delta_t + |\hat{p}_t|}, \quad \text{s.t.} \quad \delta_t = \beta(R')^2\sqrt{1 + \sum_{i=1}^{t-1} Z_i M_i},$$

where $\beta > 0$ is a predefined parameter, $R' = \max R_{t-1}, \|\mathbf{x}_t\|$, with $R_{t-1} = \max\{\|\mathbf{x}_i\| | Z_i M_i = 1\}$. The algorithm still has a parameter $\beta > 0$ but, it will be observed that β has far less influence on the final bound than the δ parameter in the label efficient Perceptron. The query strategy is similar with label efficient Perceptron, although it will depend on another two numbers R_i and $\sum_{i=1}^{t-1} Z_i M_i$. R_i is maximal norm of all the previous instances which are used for updating the model. $\sum_{i=1}^{t-1} Z_i M_i$ is the number of updates made by the algorithm. $\sum_{i=1}^{t-1} Z_i M_i$ increasing implies the problem is difficult, so more labels should be queried. However, this does not mean the label rate $\frac{\delta_t}{\delta_t+|\hat{p}_t|}$ converges to 1 as $t \to \infty$, since δ_t does not scale with time t. After the label is requested, the update method is the same as the one in the label efficient Perceptron:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$$

For this algorithm, the expected number of mistakes can be bounded as

$$\begin{aligned}&\mathbb{E}[\sum_{t=1}^{T} M_t] \\ \leq &\frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{R}{2\beta} + \frac{B^2}{2} + B\sqrt{\frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{R}{2\beta} + \frac{B^2}{4}}\end{aligned}$$

where $\bar{L}_{\gamma,T}(\mathbf{w}) = \mathbb{E}[\sum_{t=1}^{T} Z_t M_t \ell_{\gamma,t}(\mathbf{w})]$ with $\ell_{\gamma,t}(\mathbf{w}) = \max(0, \gamma - y_t \mathbf{w}^\top \mathbf{x}_t)$, $B = R + \frac{1+3R/2}{\beta}$ and $R = \frac{\|\mathbf{w}\|(\max_t \|\mathbf{x}_t\|)}{\gamma}$. Moreover, the expected number of labels queried by the algorithm equals $\sum_{t=1}^{T} \mathbb{E}[\frac{\delta_t}{\delta_t+|\hat{p}_t|}]$

Label Efficient Second-Order Perceptron.. The second-order Perceptron algorithm [282] may be seen as running the standard (first-order) Perceptron algorithm as a subroutine. Let $\mathbf{u}_t$ denote the weight vector computed by the standard Perceptron, and $A_t = I + \sum_{i \leq t-1, Z_i M_i = 1} \mathbf{x}_i \mathbf{x}_i^\top$ denote the sum of identity matrix I and the correlation matrix over the mistaken trials, then the second-order Perceptron algorithm predict the label of the current instance $\mathbf{x}_t$ as

$$\begin{aligned}\hat{y}_t =&\text{sign}(\hat{p}_t), \hat{p}_t \\ =&[(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-\frac{1}{2}}\mathbf{u}_t]^\top[(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-\frac{1}{2}}\mathbf{x}_t] \\ =&\mathbf{u}_t^\top(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-1}\mathbf{x}_t\end{aligned}$$

Hence the second-order algorithm differs from the standard Perceptron in that, before each prediction, a linear transformation $(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-1/2}$ is applied to both the current Perceptron weight $\mathbf{u}_t$ and the current instance $\mathbf{x}_t$. After prediction, the query strategy of this algorithm is the same with the label efficient Perceptron: draw a Bernoulli random variable $Z_t \in \{0, 1\}$ with

$$\Pr(Z_t = 1) = \frac{\delta}{\delta + |\hat{p}_t|},$$

After the label y_t is disclosed, we will get $M_t = \mathbb{I}(\hat{y}_t \neq y_t)$. If $M_t = 1$, then the algorithm will update the model using the the following rules:

$$\mathbf{u}_{t+1} = \mathbf{u}_t + y_t \mathbf{x}_t, \quad A_{t+1} = A_t + \mathbf{x}_t\mathbf{x}_t^\top$$

Theoretically, if the algorithm runs on a sequence of example $\{(\mathbf{x}_t, y_t)|t \in [t]\}$ then for any $\mathbf{w}$, the expected number of mistakes made by the algorithm is bounded as follows:

$$\mathbb{E}[\sum_{t=1}^{T} M_t] \leq \frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{\delta}{2\gamma^2}\mathbf{w}^\top \mathbb{E}[A_T]\mathbf{w} + \frac{1}{2\delta}\sum_{i=1}^{d}\mathbb{E}\ln(1+\lambda_i)$$

where $\bar{L}_{\gamma,T}(\mathbf{w}) = \mathbb{E}[\sum_{t=1}^{T} Z_t M_t \ell_{\gamma,t}(\mathbf{w})]$ with $\ell_{\gamma,t}(\mathbf{w}) = \max(0, \gamma - y_t \mathbf{w}^\top \mathbf{x}_t)$, $\lambda_1, \ldots, \lambda_d$ are the eigenvalues of the random correlation matrix $\sum_{t=1}^{T} Z_t M_t \mathbf{x}_t \mathbf{x}_t^\top$ and $A_T = I + \sum_{t=1}^{T} M_t Z_t \mathbf{x}_t \mathbf{x}_t^\top$. Moreover, the expected number of labels queried by the algorithm equals $\sum_{t=1}^{T} \mathbb{E}[\frac{\delta}{\delta+|\hat{p}_t|}]$. Furthermore, setting $\delta = \gamma\sqrt{\frac{\sum_{i=1}^{d}\mathbb{E}\ln(1+\lambda_i)}{\mathbf{w}^\top \mathbb{E}[A_T]\mathbf{w}}}$ results in the optimal bound

$$\mathbb{E}[\sum_{t=1}^{T} M_t] \leq \frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{1}{\gamma}\sqrt{(\mathbf{w}^\top \mathbb{E}[A_T]\mathbf{w})\sum_{i=1}^{d}\mathbb{E}\ln(1+\lambda_i)}$$

Adaptive Label Efficient Second-Order Perceptron.. This algorithm [281] predicts the label of the current instance $\mathbf{x}_t$ as $\hat{y}_t = \text{sign}(\hat{p}_t)$, where $\hat{p}_t$ is computed by the following:

$$\begin{aligned}
\hat{p}_t &= [(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-\frac{1}{2}}\mathbf{u}_t]^\top[(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-\frac{1}{2}}\mathbf{x}_t] \\
&= \mathbf{u}_t^\top(A_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-1}\mathbf{x}_t \\
A_t &= I + \sum_{i\leq t-1, Z_i M_i = 1} \mathbf{x}_i\mathbf{x}_i^\top \\
\mathbf{u}_t &= \sum_{i\leq t-1, Z_i M_i = 1} y_i\mathbf{x}_i
\end{aligned}$$

$\mathbf{x}_t^\top A_t^{-1}\mathbf{x}_t$ can be considered as a measure of the variance of the prediction $\hat{p}_t$, but it is not used to measure the uncertainty of the prediction in the label efficient second-order Perceptron. To solve this issue, the adaptive label efficient second-order perceptron algorithm is proposed, where the query strategy is to draw a Bernoulli random variable $Z_t \in \{0, 1\}$ with

$$\Pr(Z_t = 1) = \frac{\delta}{\delta + |\hat{p}_t| + \frac{1}{2}\hat{p}_t^2(1 + \mathbf{x}_t^\top A_t^{-1}\mathbf{x}_t)},$$

After the label y_t is disclosed, we will get $M_t = \mathbb{I}(\hat{y}_t \neq y_t)$. If $M_t = 1$, then the algorithm will update the model using the the following rules:

$$\mathbf{u}_{t+1} = \mathbf{u}_t + y_t\mathbf{x}_t, \quad A_{t+1} = A_t + \mathbf{x}_t\mathbf{x}_t^\top$$

Theoretically, its expected number of mistake is the same with label efficient second-order Perceptron, i.e.,

$$\mathbb{E}[\sum_{t=1}^{T} M_t] \leq \frac{\bar{L}_{\gamma,T}(\mathbf{w})}{\gamma} + \frac{\delta}{2\gamma^2}\mathbf{w}^\top \mathbb{E}[A_T]\mathbf{w} + \frac{1}{2\delta}\sum_{i=1}^{d}\mathbb{E}\ln(1+\lambda_i)$$

where $\bar{L}_{\gamma,T}(\mathbf{w}) = \mathbb{E}[\sum_{t=1}^{T} Z_t M_t \ell_{\gamma,t}(\mathbf{w})]$ with $\ell_{\gamma,t}(\mathbf{w}) = \max(0, \gamma - y_t \mathbf{w}^\top \mathbf{x}_t)$, $\lambda_1, \ldots, \lambda_d$ are the eigenvalues of the random correlation matrix $\sum_{t=1}^{T} Z_t M_t \mathbf{x}_t \mathbf{x}_t^\top$ and $A_T = I + \sum_{t=1}^{T} M_t Z_t \mathbf{x}_t \mathbf{x}_t^\top$.

Passive-Aggressive Active Learning.. Similar with the previous Perceptron-based label efficient algorithms, the Passive-Aggressive Active learning algorithms [283, 284] keep a linear model $\mathbf{w} \in \mathbb{R}^d$ and predict the label of $\mathbf{x}_t \in \mathbb{R}^d$ as

$$\hat{y}_t = \text{sign}(\hat{p}_t), \quad \text{where} \hat{p}_t = \mathbf{w}_t^\top \mathbf{x}_t$$

and then draw a Bernoulli random variable $Z_t \in \{0, 1\}$ using

$$\Pr(Z_t = 1) = \frac{\delta}{\delta + |\hat{p}_t|}$$

where δ is used to control the number of disclosed labels. If $Z_t = 1$, the true label y_t will be disclosed, then the model will be updated. Unlike the previous Perceptron-based label efficient algorithms that employ only the misclassified instances for updating the model, the Passive-Aggressive Active learning algorithms not only use the misclassified instances to update the classifier, but also exploit correctly classified examples with low prediction confidence. In addition, unlike the previous Perceptron-based approaches that set the learning rate of each example as 1, the Passive-Aggressive Active learning algorithms update the models using a learning rate depending on the loss on the current example. Specifically, the update rules for Passive-Aggressive Active (PAA) learning algorithms are

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \tau_t y_t \mathbf{x}_t$$

where the stepsize τ_t is computed respectively as follows:

$$\tau_t = \begin{cases} \ell_t(\mathbf{w}_t; (\mathbf{x}_t, y_t))/\|\mathbf{x}_t\|^2, & \text{(PAA)} \\ \min(C, \ell_t(\mathbf{w}_t; (\mathbf{x}_t, y_t))/\|\mathbf{x}_t\|^2), & \text{(PAA-I)} \\ \ell_t(\mathbf{w}_t; (\mathbf{x}_t, y_t))/(\|\mathbf{x}_t\|^2 + 1/(2C)). & \text{(PAA-II)} \end{cases}$$

where $C > 0$ is a trade off between regularization and empirical loss.

Theoretically, when the dataset is linearly separable, PAA algorithm achieves an expected mistake bound as

$$\mathbb{E}[\sum_{t=1}^{T} M_t] \leq \mathbb{E}[\sum_{t=1}^{T} M_t \ell_t(\mathbf{w}_t))] \leq \frac{R^2}{4}(\delta + \frac{1}{\delta} + 2)\|\mathbf{w}\|^2.$$

where $\mathbf{w} \in \mathbb{R}^d$. While for any dataset and any $\mathbf{w} \in \mathbb{R}^d$, PAA-I algorithm can bound the expected number of mistakes as

$$\begin{aligned}&\mathbb{E}[\sum_{t=1}^{T} M_t] \\ \leq &\beta\left\{(\frac{\delta+1}{2})^2\|\mathbf{w}\|^2 + (\delta+1)C\mathbb{E}[\sum_{t=1}^{T} Z_t \ell_t(\mathbf{w})]\right\},\end{aligned}$$

where $\beta = \frac{1}{\rho}\max\{\frac{1}{C}, R^2\}$ and $\ell_t(\mathbf{w}) = \max(0, 1 - y_t \mathbf{w}^\top \mathbf{x}_t)$. PAA-II can bound the expected number of mistakes as

$$\begin{aligned}&\mathbb{E}[\sum_{t=1}^{T} M_t] \\ \leq &\gamma\frac{1}{\delta}\{(\frac{\delta+1}{2})^2\|\mathbf{w}\|^2 + 2C(\frac{\delta+1}{2})^2\mathbb{E}[\sum_{t=1}^{T} Z_t \ell_t(\mathbf{w})^2]\},\end{aligned}$$

where $\gamma = \{R^2 + \frac{1}{2C}\}$ and C is the aggressiveness parameter for PAA-II.

There are also extensions to second order [285, 286] and cost-sensitive [287] approaches for online active learning.

6.4. Active Learning with Expert Advice

We also discuss a third category, online classification with expert advice. Consider an unknown sequence of instances $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^d$, a "forecaster" aims to predict the class labels of every incoming instance $\mathbf{x}_t$. The forecaster sequentially computes its predictions based on the predictions from a set of N "experts". Specifically, at the t-th round, after receiving an instance $\mathbf{x}_t$, the forecaster first accesses the predictions of the experts $\{f_{i,t} : \mathbb{R}^d \to [0,1] | i = 1, \ldots, N\}$, and then computes its own prediction $p_t \in [0,1]$ based on the predictions of the N experts. After p_t is computed, the true outcome $y_t \in \{0,1\}$ is disclosed.

To solve this problem, the "Exponentially Weighted Average Forecaster" (EWAF) makes the following prediction:

$$p_t = \frac{\sum_{i=1}^{N} \exp(-\eta L_{i,t-1}) f_i(\mathbf{x}_t)}{\sum_{i=1}^{N} \exp(-\eta L_{i,t-1})}, \tag{13}$$

where η is a learning rate, $L_{i,t} = \sum_{j=1}^{t} \ell(f_i(\mathbf{x}_j), y_j)$, $L_t = \sum_{j=1}^{t} \ell\left(p_j, y_j\right)$ with $\ell(p_t, y_t) = |p_t - y_t|$.

Unlike the above regular learning, in an active learning with expert advice task [288], the outcome of an incoming instance is *only* revealed whenever the learner requests the label from the environment/oracle. To solve this problem, binary variables $z_s \in \{0,1\}, s = 1, \ldots, t$ are introduced to indicate if an active forecaster has requested the label of an instance at s-th trial. $\widehat{L}_{i,t}$ is used to denote the loss function experienced by the active learner w.r.t. the i^{th} expert, i.e., $\widehat{L}_{i,t} = \sum_{s=1}^{t} \ell(f_i(\mathbf{x}_s), y_s) z_s$. For this problem setting, a general framework of active forecasters for online active learning with expert advice, is proposed as shown in 15 [288].

Algorithm 15: Online Active Learning with Expert Advice

INPUT: a pool of experts $f_i,\ i = 1, \ldots, N$.
INIT: tolerance threshold δ and $\widehat{L}_{i,t} = 0,\ i \in [N]$.
for $t = 1, 2, \ldots, T$ **do**
  Receive $\mathbf{x}_t$ and compute $f_i(\mathbf{x}_t), i \in [N]$;
  Compute $\hat{p}_t = \frac{\sum_{i=1}^{N} \exp(-\eta \widehat{L}_{i,t-1}) f_i(\mathbf{x}_t))}{\sum_{i=1}^{N} \exp(-\eta \widehat{L}_{i,t-1})}$;
  If a *confidence condition* is not satisfied request label y_t and update $\widehat{L}_{i,t} = \widehat{L}_{i,t-1} + \ell(f_i(\mathbf{x}_t), y_t),\ i \in [N]$;
end for

At each round, after receiving an input instance $\mathbf{x}_t$, we compute the prediction by each expert in the pool, i.e., $f_i(\mathbf{x}_t)$. Then, we examine if a confidence condition is satisfied. If so, we will skip the label request for this instance; otherwise, the learner will request the class label for this instance from the environment.

To decide when to request the class label or not, the key idea is to seek a confidence condition by estimating the difference between p_t and $\hat{p}_t$. Intuitively, the smaller the difference, the more confident we have for the prediction made by the forecaster. A confidence condition is presented in the following theorem, which guarantees a small difference between p_t and $\widehat{p}_t$.

Theorem 2. *For a small constant* $\delta > 0$, $\max_{1 \leq i,j \leq N} |f_i(\mathbf{x}_t) - f_j(\mathbf{x}_t)| \leq \delta$ *implies* $|p_t - \widehat{p}_t| \leq \delta$.

This theorem roughly implies that, if any two experts do not disagree with each other too much on one instance, then we can skip requiring its label.

There are also active learning strategies for other algorithms for online learning with expert advices, for example the active greedy forecaster [288].

7. Online Semi-supervised Learning

7.1. Overview

In real world applications, usually it is easy to acquire large scale of unlabeled data while obtaining labels for the entire data is often expensive or even impossible. Besides labeling a few instances and using them to train a supervised model, there is still an urgent need to make use of the unlabeled data which are cheap and even unlimited in scale to improve the poor model learnt on insufficient labeled data.

Semi-supervised learning has been extensively studied to address this challenge. In addition to utilizing the labeled instances, the algorithm also digs into the structure of the unlabeled data using some unsupervised learning algorithms. There are some surveys that introduce works in this field [289, 290]. A simple example of semi-supervised learning is manifold regularization [291]. In addition to minimizing the prediction error of the labeled data, the algorithm also minimizes the prediction difference between similar instances, which can be calculated without labels. The idea behind this algorithm is that a single label can benefit the classifier by suggesting the possible labels for its neighbors. Compared to pure supervised models trained only on limited number of labeled instances, semi-supervised algorithms enjoys better performance since more training instances are available.

Despite the clear advantage of semi-supervised learning, many challenges are to be addressed when semi-supervised learning meets online learning. Different from batch semi-supervised learning algorithm, which can easily learn and remember the similarity between instances, online semi-supervised learning algorithms receive large scale unlabeled data in a stream. When receiving an instance, due to the lack of knowledge on future data, the current similarity information may be inappropriate. And after processing, the majority of the earlier data stream may be discarded due to space limitation, which makes it hard to adopt unsupervised learning algorithms.

Note that in this section, online semi-supervised learning refers to a group of algorithms that learn a supervised model using both labeled and unlabeled instances that arrive sequentially. This is very different from supervised learning algorithms with limited feedback, such as active learning, where labels are given on request.

7.2. Online Manifold Regularization

As discussed, manifold regularization is a power in semi-supervised learning model. Given instances $(\mathbf{x}_t, y_t)$, $t \in \{1, ...T\}$, we try to minimize the function,

$$J(f) = \frac{1}{l}\sum_{t=1}^{T}\delta(y_t)\ell(f(\mathbf{x}_t), y_t)$$
$$+\frac{\lambda_1}{2}\|f\|^2 + \frac{\lambda_2}{2T}\sum_{s,t=1}^{T}(f(\mathbf{x}_s) - f(\mathbf{x}_t))w_{st}$$

The first term is the total loss of all labeled instances, where $\delta(y_t) = 1$ if and only if y_t exists and l is the number of labeled instances. The second term is a regularization term, which is also commonly used in contentional supervised learning. While the third term is totally unsupervised learning. We would like to minimize the prediction difference between similar instances, where w_{st} is the similarity between the two instances and could be calculated without labels.

When under online setting [292], it is easy to separate the above objective function to each instance:

$$J_t(f) = \frac{T}{l}\delta(y_t)\ell(f(\mathbf{x}_t), y_t)$$
$$+\frac{\lambda_1}{2}\|f\|^2 + \lambda_2\sum_{i=1}^{t-1}(f(\mathbf{x}_i) - f(\mathbf{x}_t))w_{it}$$

This problem can be solved using Online Gradient Descent in $O(T^2)$ time.

Unfortunately, the straightforward solution is expensive in both time and space. To calculate the last term, we have to store all instances and measure the similarity w_{it} between the incoming instances and all existing ones. To address this problem, the authors offer two sparse approximations of the objective function.

The first solution is not to keep all instances but to keep only the newest τ ones, where τ is the buffer size. This strategy is simple but not very efficient since the discarded old instances may contain important information.

The second solution adopts a random projection tree to find s cluster centers during online learning. Finally, instead of calculating the similarity between $\mathbf{x}_t$ and all existing instances, the algorithm only consider the s cluster centers as the most representative instances.

7.3. Online Transductive Learning

The Transductive SVMs (S3VMs) [293] is a successful transductive learning algorithm for semi-supervised learning. The basic goal is to find a label for the unlabeled data, so that the boundary has the maximum margin on both the original labeled data and the unlabeled data, i.e.

$$\min_f \sum_{i=1}^{l}(1 - y_i f(\mathbf{x}_i))_+ + \lambda_1\|f\|^2 + \lambda_2\sum_{i=l+1}^{n}(1 - |f(\mathbf{x}_i)|)_+$$

where the first two terms are regularized loss on labeled data and are commonly used in supervised learning. The last term is for unsupervised learning. We would like to assign any label to the unlabeled data so that the margin is maximized. Similar to the supervised SVM based algorithms, it is easy to modify to for online learning [294].

8. Related Areas and Other Terminologies

8.1. Overview

In this section, we discuss the relationship of online learning with other related areas and terminologies which sometimes may be confused. We note that the following remarks may be somewhat subjective, and their meanings may vary in diverse contexts whereas some terms may be used interchangeably.

8.2. Incremental Learning

Incremental learning, or decremental learning, represents a family of machine learning techniques [295, 2, 296], which are particularly suitable for learning from data streams. There exist a variety of different definitions of incremental learning/decremental learning. The basic idea of incremental learning is to learn some models from a stream of training instances with limited space and computational costs, often attempting to approximate a traditional offline machine learning counterpart as much as possible.

For example, incremental SVM [2] aims to train an SVM classifier the same as a batch SVM in an incremental manner where one training instance is added for updating the model each time (and similarly a training instance can be removed by updating the model decrementally). Incremental learning can work either in online learning or batch learning manners [296]. For the incremental online learning [2], only one example is presented for updating the model at one time, while for the incremental batch learning [297], a batch of multiple training examples are used for updating the model each time.

Incremental learning (or decremental learning) methods are often natural extensions of existing supervised learning or unsupervised learning techniques for addressing efficiency and scalability when dealing with real-world data particularly arriving in stream-based settings. Generally speaking, incremental learning can be viewed as a branch of online learning and extensions for adapting traditional offline learning counterparts in data-stream settings.

8.3. Sequential Learning

Sequential learning is mainly concerned with learning from sequential training data [298], formulated as follows: a learner trains a model from a collection of N training data pairs $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}), i = 1, \ldots, N\}$ where $\mathbf{x}^{(i)} = (x_1^i, x_2^i, \ldots, x_{N_i}^i)$ is an N_i-dimensional instance vector and $\mathbf{y}^{(i)} = (y_1^i, y_2^i, \ldots, y_{N_i}^i)$ is an N_i-dimensional label vector. It can be viewed as a special type of supervised learning, known as structured prediction or structured (output) learning [299], where the goal is to predict structured objects (e.g., sequence or graphs), rather than simple scalar discrete ("classification") or real values ("regression"). Unlike traditional supervised learning that often assume data is independently and identically distributed, sequential learning attempts to exploit significant sequential correlation of sequential data when training the predictive models. Some classical methods of sequential learning include sliding window methods, recurrent sliding windows, hidden Markov models, conditional random fields, and graph transformer networks, etc. There are also many recent studies for structured prediction with application to sequential learning [299, 300]. In general, sequential learning can be solved by either batch or online learning algorithms. Finally, it is worth mentioning another closely related learning, i.e., "sequence classification", whose goal is to predict a single class output for a whole input "sequence" instance. Sequence classification is a special case of sequential learning with the target class vector reduced to a single variable. It is generally simpler than regular sequential learning, and can be solved by either batch or online learning algorithms.

8.4. Stochastic Learning

Stochastic learning refers to a family of machine learning algorithms by following the theory and principles of stochastic optimization [301, 302, 303], which have achieved great successes for solving large-scale machine learning tasks in practice [304]. Stochastic learning is closely related to online learning. Typically, stochastic learning algorithms are motivated to accelerate the training speed of some existing batch machine learning methods for large-scale machine learning tasks, which may be often solved by batch gradient descent algorithms. Stochastic learning algorithms, e.g., Stochastic Gradient Descent (SGD) or a.k.a Online Gradient Descent (OGD) in online learning terminology, often operate sequentially by processing one training instance (randomly chosen) each time in an online learning manner, which thus are computationally more efficient and scalable than the batch GD algorithms for

large-scale applications. Rather than processing a single training instance each time, a more commonly used stochastic learning technique in practice is the mini-batch SGD algorithm [304, 3], which processes a small batch of training instances each time. Thus, stochastic learning can be viewed as a special family of online learning algorithms and extensions, while online learning may explore more other topics and challenges beyond stochastic learning/optimizations.

8.5. Interactive Learning

Traditional machine learning mostly works in a fully automated process where training data are collected and prepared typically with the aid of domain experts. By contrast, interactive (machine) learning aims to make the machine learning procedure interactive by engaging human (users or domain experts) in the loop [305, 306]. The advantages of interactive learning include the natural integration of domain knowledge in the learning process, effective communication and continuous improvements for learning efficacy through the interaction between learning systems and users/experts. Online learning often plays an important role in an interactive learning system, in which active (online) learning can be used in finding the most informative instances to save labeling costs, incremental (online) learning algorithms could be applied for updating the models sequentially, and/or bandit learning algorithms may be explored for decision-making via the tradeoff of exploration and exploitation in some scenarios.

8.6. Adaptive Learning

This term is occasionally used in the machine learning and neural networks fields. There is no a very formal definition about what exactly is adaptive learning in literature. In literature, there are quite a lot of different studies more or less concerned with adaptive learning [210, 307], which attempt to adapt a learning model/system (e.g., neural networks) for dynamically changing environments over time. In general, these existing works are similar to online learning in that the environment is often changing and evolving dynamically. But they are different in that they are not necessarily purely based on online learning theory and algorithms. Some of these works are based on heuristic adaptation/modification of existing batch learning algorithms for updating the models with respect to the environment changes. Last but not least, most of these existing works are motivated by different kinds of heuristics, generally lack solid theoretical analysis and thus can seldom give performance guarantee in theory.

8.7. Reinforcement Learning

Reinforcement Learning (RL) [308, 309] is a branch of machine learning inspired by behaviorist psychology, which is often concerned with how software agents should take actions in an environment for the goal of maximizing some cumulative rewards. Specifically, the RL problem can be formulated as follows: the environment is modelled as a stochastic Finite State Machine (FSM) with inputs (actions sent from the agent) and outputs (observations and rewards sent to the agent), consisting of three key components: (i) state transition function, (ii) observation (output) function, and (iii) reward function. The agent is also modelled as stochastic FSM with inputs (observations/rewards sent from the environment) and outputs (actions sent to the environment), i.e., involving two key components: (i) state transition function, and (ii) Policy/output function. The goal of an agent of RL is to find a good policy and state-update function by attempting to maximize the the expected sum of discounted rewards.

Reinforcement learning is different from supervised learning [310] in that the goal of supervised learning is to reconstruct an unknown function f that can assign the desired output values y to input data x; while the goal of RL is to find the input (policy/action) x that gives the maximum reward $R(x)$. In general, RL can work either batch or online learning manner. In practice, RL methods are commonly applied to problems involving sequential dynamics and optimization of some objectives, typically with online exploration of the effects of actions. RL is similar to online learning in that some RL tasks also have an important focus on online performance by balancing the tradeoff between exploration (of uncertainty) and exploitation (of known knowledge), in which some solutions also follow the same idea of multi-armed bandit learning, a scenario of online learning with limited feedback.

9. Conclusions

9.1. Concluding Remarks

This paper gave a comprehensive survey of existing online learning works and reviewed ongoing trends of online learning research. In theory, online learning methodologies are founded primarily based on learning theory, optimization theory, and game theory. According to the type of feedback to the learner, the existing online learning methods can be roughly grouped into the following three major categories:

- **Supervised online learning** is concerned with the online learning tasks where full feedback information is always revealed to the learner, which can be further divided into three groups: (i) "'linear online learning" that aims to learn a linear predictive model, (ii) "nonlinear online learning" that aims to learn a nonlinear predictive model, and (iii) non-traditional online learning that addresses a variety of supervised online learning tasks which are different from traditional supervised prediction models for classification and regression.

- **Online learning with limited feedback** is concerned with the online learning tasks where the online learner receives partial feedback information from the environment during the learning process. The learner often has to make online predictions or decisions by achieving a tradeoff between the exploitation of disclosed knowledge and the exploration of unknown information.

- **Unsupervised online learning** is concerned with the online learning tasks where the online learner only receives the sequence of data instances without any additional feedback (e.g., true class label) during the online learning tasks. Examples of unsupervised online learning include online clustering, online representation learning, and online anomaly detection tasks, etc.

In literature, the first category received more research attentions than the other two categories, mainly because supervised online learning is a natural extension of traditional supervised batch learning, and thus can be directly applied to a wide range of real applications where conventional batch learning techniques may suffer from critical limitations. However, online learning with limited feedback or unsupervised online learning without any feedback are usually much more challenging, and should receive more research attentions in the future.

9.2. Future Directions

Despite the extensive studies in literature, when applying online learning for big data analytics, there are still a number of open issues which have not been fully solved by the existing works and need to be addressed in the future work.

First of all, one critical challenge with online learning is "concept drifting" where the target concepts to be predicted may change over time in unforeseeable ways. Although many online learning studies have attempted to address concept drifting by a variety of approaches, they are fairly limited in that they often make some restricted assumptions for addressing certain types of concept drifting patterns. In general, there is still no a formal theoretical framework or a principled way for resolving all types of concept drifting issues, particularly for non-stationary settings where target concepts to be learned may drift over time in arbitrary ways.

Second, an important growing trend of online learning research is to explore large-scale online learning for real-time big data analytics. Although online learning has huge advantages over batch learning in efficiency and scalability, it remains a non-trivial task when dealing with real-world big data analytics with extremely high volume and high velocity. To tackle these challenges, more research efforts should address parallel online learning and distributed online learning by leveraging the powers of cloud computing and high-performance computing infrastructures in near future.

Third, another challenge of online learning is to address the "variety" in online data analytics tasks. Most existing online learning studies are often focused on handling single-source structured data typically by vector space representations. In many real-world data analytis applications, data may come from multiple diverse sources and could contain different types of data (such as structured, semi-structured, and unstructured data). Some existing studies, such as the series of online multiple kernel learning works, have attempted to address some of these issues, but certainly have not yet fully resolved all the challenges of variety. In the future, more research efforts should address the "variety" challenges, such as multi-source online learning, multi-modal online learning, etc.

Last but not least, existing online learning works seldom address the data "veracity" issue, that is, the quality of data, which can considerably affect the efficacy of online learning. Conventional online learning studies often implicitly assume data and feedback are given in perfect quality, which is not always true for many real-world applications, particularly for real-time data analytics tasks where data arriving on-the-fly may be contaminated with noise or may have missing values or incomplete data without applying advanced preprocessing. More future research efforts should address the data veracity issue by improving the robustness of online learning algorithms particularly when dealing with real data of poor quality.

References

[1] J. Platt, et al., Fast training of support vector machines using sequential minimal optimization, Advances in kernel method-

ssupport vector learning 3.
[2] G. C. Poggio, Incremental and decremental support vector machine learning, in: Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference, Vol. 13, MIT Press, 2001, p. 409.
[3] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: Primal estimated sub-gradient solver for svm, Mathematical programming 127 (1) (2011) 3–30.
[4] N. Cesa-Bianchi, G. Lugosi, Prediction, Learning, and Games, Cambridge University Press, New York, NY, USA, 2006.
[5] S. Shalev-Shwartz, Online learning and online convex optimization, Foundations and Trends in Machine Learning 4 (2) (2011) 107–194.
[6] R. D. Kleinberg, Online decision problems with large strategy sets, Ph.D. thesis, Massachusetts Institute of Technology (2005).
[7] S. Shalev-Shwartz, Online learning: Theory, algorithms, and applications, Ph.D. thesis, The Hebrew University of Jerusalem (2007).
[8] P. Zhao, Kernel based online learning, Ph.D. thesis, Nanyang Technological University (2013).
[9] B. Li, Online portfolio selection, Ph.D. thesis, Nanyang Technological University (2013).
[10] A. Fiat, G. Woeginger, Online algorithms: The state of the art, Springer Heidelberg, 1998.
[11] L. Bottou, Online learning and stochastic approximations, Online learning in neural networks 17 (9) (1998) 142.
[12] A. Rakhlin, Lecture notes on online learning, Notes appeared in the Statistical Learning Theory course at UC Berkeley.
[13] A. Blum, On-line algorithms in machine learning, Springer, 1998.
[14] S. Albers, Online algorithms: a survey, Mathematical Programming 97 (1-2) (2003) 3–26.
[15] S. C. Hoi, J. Wang, P. Zhao, Libol: a library for online learning algorithms, The Journal of Machine Learning Research 15 (1) (2014) 495–499.
[16] Y. Wu, S. C. Hoi, C. Liu, J. Lu, D. Sahoo, N. Yu, Sol: A library for scalable online learning algorithms, arXiv preprint arXiv:1610.09083.
[17] J. Langford, L. Li, A. Strehl, Vowpal wabbit online learning project (2007).
[18] V. N. Vapnik, V. Vapnik, Statistical learning theory, Vol. 1, Wiley New York, 1998.
[19] V. N. Vapnik, An overview of statistical learning theory, IEEE transactions on neural networks 10 (5) (1999) 988–999.
[20] E. Hazan, et al., Introduction to online convex optimization, Foundations and Trends® in Optimization 2 (3-4) (2016) 157–325.
[21] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: Proceedings of the Twentieth International Conference on Machine Learning(ICML 2003), 2003, pp. 928–936.
[22] E. Hazan, A. Agarwal, S. Kale, Logarithmic regret algorithms for online convex optimization, Machine Learning 69 (2-3) (2007) 169–192.
[23] J. Abernethy, E. Hazan, A. Rakhlin, Competing in the dark: An efficient algorithm for bandit linear optimization., in: COLT, 2008, pp. 263–274.
[24] S. Shalev-Shwartz, Y. Singer, A primal-dual perspective of online learning algorithms, Machine Learning 69 (2-3) (2007) 115–142.
[25] A. T. Kalai, S. Vempala, Efficient algorithms for online decision problems, J. Comput. Syst. Sci. 71 (3) (2005) 291–307.
[26] J. Hannan, Approximation to bayes risk in repeated play, Contributions to the Theory of Games 3 (97-139) (1957) 2.
[27] P. Tseng, On accelerated proximal gradient methods for Convex-Concave optimization, SIAM Journal on Optimization.
[28] J. C. Duchi, S. Shalev-Shwartz, Y. Singer, A. Tewari, Composite objective mirror descent, in: COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010, 2010, pp. 14–26.
[29] J. Kivinen, M. K. Warmuth, Additive versus exponentiated gradient updates for linear prediction, in: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC'95), 1995, pp. 209–218.
[30] J. C. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (2011) 2121–2159.
[31] R. Jenatton, J. Huang, C. Archambeau, Adaptive algorithms for online convex optimization with long-term constraints, NIPS.
[32] H. Wang, A. Banerjee, Online alternating direction method, in: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012, 2012.
[33] D. Gabay, B. Mercier, A dual algorithm for the solution of nonlinear variational problems via finite element approximation, Computers & Mathematics with Applications 2 (1) (1976) 17–40.
[34] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends® in Machine Learning 3 (1) (2011) 1–122.
[35] A. Cutkosky, K. A. Boahen, Online convex optimization with unconstrained domains and losses, in: Advances In Neural Information Processing Systems, 2016, pp. 748–756.
[36] N. Nisan, T. Roughgarden, E. Tardos, V. V. Vazirani, Algorithmic game theory, Vol. 1, Cambridge University Press Cambridge, 2007.
[37] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychological review 65 (6) (1958) 386.
[38] S. Agmon, The relaxation method for linear inequalities, Canadian Journal of Mathematics 6 (3) (1954) 382–392.
[39] A. B. Novikoff, On convergence proofs on perceptrons (1962) 615–622.
[40] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, Machine learning 2 (4) (1988) 285–318.
[41] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, Y. Singer, Online passive-aggressive algorithms, The Journal of Machine Learning Research 7 (2006) 551–585.
[42] E. Hazan, A. Rakhlin, P. L. Bartlett, Adaptive online gradient descent, in: Advances in Neural Information Processing Systems, 2007, pp. 65–72.
[43] O. Dekel, R. Gilad-Bachrach, O. Shamir, L. Xiao, Optimal distributed online prediction using mini-batches, The Journal of Machine Learning Research 13 (1) (2012) 165–202.
[44] C. Gentile, A new approximate maximal margin classification algorithm, Journal of Machine Learning Research 2 (2001) 213–242.
[45] Y. Li, P. M. Long, The relaxed online maximum margin algorithm, Machine Learning 46 (1-3) (2002) 361–387.
[46] T. van Erven, W. M. Koolen, Metagrad: Multiple learning rates in online learning, in: Advances in Neural Information Processing Systems, 2016, pp. 3666–3674.
[47] N. Cesa-Bianchi, A. Conconi, C. Gentile, A second-order perceptron algorithm, SIAM Journal on Computing 34 (3) (2005) 640–668.
[48] M. Dredze, K. Crammer, F. Pereira, Confidence-weighted lin-

ear classification, in: Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 264–271.
[49] K. Crammer, A. Kulesza, M. Dredze, Adaptive regularization of weight vectors, Machine Learning (2009) 1–33.
[50] J. Lu, S. Hoi, J. Wang, Second order online collaborative filtering, in: Asian Conference on Machine Learning, 2013, pp. 325–340.
[51] Y. Wu, S. C. Hoi, T. Mei, Massive-scale online feature selection for sparse ultra-high dimensional data, arXiv preprint arXiv:1409.7794.
[52] J. Wang, P. Zhao, S. C. Hoi, Exact soft confidence-weighted learning, arXiv preprint arXiv:1206.4612.
[53] J. Wang, P. Zhao, S. C. Hoi, Soft confidence-weighted learning, ACM Transactions on Intelligent Systems and Technology (TIST) 8 (1) (2016) 15.
[54] K. Crammer, M. Dredze, A. Kulesza, Multi-class confidence weighted algorithms, in: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2, Association for Computational Linguistics, 2009, pp. 496–504.
[55] M. Dredze, K. Crammer, Active learning with confidence, in: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers, Association for Computational Linguistics, 2008, pp. 233–236.
[56] A. Mejer, K. Crammer, Confidence in structured-prediction using confidence-weighted models, in: Proceedings of the 2010 conference on empirical methods in natural language processing, Association for Computational Linguistics, 2010, pp. 971–981.
[57] L. Yang, R. Jin, J. Ye, Online learning by ellipsoid method, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 1153–1160.
[58] F. Orabona, K. Crammer, New adaptive algorithms for online classification, in: Advances in neural information processing systems, 2010, pp. 1840–1848.
[59] K. Crammer, D. D. Lee, Learning via gaussian herding, in: Advances in neural information processing systems, 2010, pp. 451–459.
[60] H. Luo, A. Agarwal, N. Cesa-Bianchi, J. Langford, Efficient second order online learning by sketching, in: Advances in Neural Information Processing Systems, 2016, pp. 902–910.
[61] T. Roughgarden, O. Schrijvers, Online prediction with selfish experts, in: Advances In Neural Information Processing Systems, 2017.
[62] J. Hannan, Approximation to bayes risk in repeated plays, Contributions to the Theory of Games 3 (1957) 97–139.
[63] N. Cesa-Bianchi, G. Lugosi, Prediction, learning, and games, Cambridge University Press, 2006.
[64] N. Littlestone, M. K. Warmuth, The weighted majority algorithm, Inf. Comput. 108 (2) (1994) 212–261.
[65] N. Littlestone, M. K. Warmuth, The weighted majority algorithm, in: Foundations of Computer Science, 1989., 30th Annual Symposium on, IEEE, 1989, pp. 256–261.
[66] S. Arora, E. Hazan, S. Kale, The multiplicative weights update method: a meta-algorithm and applications., Theory of Computing 8 (1) (2012) 121–164.
[67] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1) (1997) 119–139.
[68] J. Langford, L. Li, T. Zhang, Sparse online learning via truncated gradient, The Journal of Machine Learning Research 10 (2009) 777–801.
[69] J. Duchi, Y. Singer, Efficient online and batch learning using forward backward splitting, The Journal of Machine Learning Research 10 (2009) 2899–2934.
[70] Y. Nesterov, Primal-dual subgradient methods for convex problems, Mathematical programming 120 (1) (2009) 221–259.
[71] L. Xiao, Dual averaging method for regularized stochastic learning and online optimization, in: Advances in Neural Information Processing Systems, 2009, pp. 2116–2124.
[72] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, The Journal of Machine Learning Research 12 (2011) 2121–2159.
[73] S. C. Hoi, J. Wang, P. Zhao, R. Jin, Online feature selection for mining big data, in: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, ACM, 2012, pp. 93–100.
[74] J. Wang, P. Zhao, S. C. Hoi, R. Jin, Online feature selection and its applications, IEEE Transactions on Knowledge and Data Engineering 26 (3) (2014) 698–710.
[75] S. Kale, Z. Karnin, T. Liang, D. Pál, Adaptive feature selection: Computationally efficient online sparse linear regression under rip, in: International Conference on Machine Learning, 2017.
[76] Y. Wu, S. C. Hoi, T. Mei, N. Yu, Large-scale online feature selection for ultra-high dimensional sparse data, ACM Transactions on Knowledge Discovery from Data (TKDD) 11 (4) (2017) 48.
[77] S. Shalev-Shwartz, A. Tewari, Stochastic methods for l 1-regularized loss minimization, The Journal of Machine Learning Research 999999 (2011) 1865–1892.
[78] D. Wang, P. Wu, P. Zhao, Y. Wu, C. Miao, S. C. Hoi, High-dimensional data stream classification via sparse online learning, in: Data Mining (ICDM), 2014 IEEE International Conference on, IEEE, 2014, pp. 1007–1012.
[79] D. Wang, P. Wu, P. Zhao, S. C. Hoi, A framework of sparse online learning and its applications, arXiv preprint arXiv:1507.07146.
[80] B. Schölkopf, R. Herbrich, A. J. Smola, A generalized representer theorem, in: COLT/EuroCOLT, 2001, pp. 416–426.
[81] Y. Freund, R. E. Schapire, Large margin classification using the perceptron algorithm, Mach. Learn. 37 (3) (1999) 277–296.
[82] J. Kivinen, A. J. Smola, R. C. Williamson, Online learning with kernels, Signal Processing, IEEE Transactions on 52 (8) (2004) 2165–2176.
[83] P. Zhao, S. C. Hoi, R. Jin, Duol: A double updating approach for online learning, in: Advances in Neural Information Processing Systems, 2009, pp. 2259–2267.
[84] P. Zhao, S. C. Hoi, R. Jin, Double updating online learning, The Journal of Machine Learning Research 12 (2011) 1587–1615.
[85] P. Zhao, S. C. Hoi, Bduol: double updating online learning on a fixed budget, in: Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, 2012, pp. 810–826.
[86] Z. Wang, K. Crammer, S. Vucetic, Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training, The Journal of Machine Learning Research 13 (1) (2012) 3103–3131.
[87] G. Cavallanti, N. Cesa-Bianchi, C. Gentile, Tracking the best hyperplane with a simple budget perceptron, Machine Learning 69 (2-3) (2007) 143–167.
[88] O. Dekel, S. Shalev-Shwartz, Y. Singer, The forgetron: A kernel-based perceptron on a fixed budget, in: NIPS, 2005.
[89] K. Crammer, J. S. Kandola, Y. Singer, Online classification on a budget, in: NIPS, Vol. 2, 2003, p. 5.
[90] P. Zhao, J. Wang, P. Wu, R. Jin, S. C. H. Hoi, Fast bounded online gradient descent algorithms for scalable kernel-based online learning, in: ICML, 2012.
[91] Z. Wang, S. Vucetic, Online passive-aggressive algorithms on

a budget, Journal of Machine Learning Research - Proceedings Track 9 (2010) 908–915.

[92] F. Orabona, J. Keshet, B. Caputo, Bounded kernel-based online learning, The Journal of Machine Learning Research 10 (2009) 2643–2666.

[93] J. Weston, A. Bordes, L. Bottou, et al., Online (and offline) on an even tighter budget, in: Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics, 2005, pp. 413–420.

[94] Z. Wang, S. Vucetic, Tighter perceptron with improved dual use of cached data for model representation and validation, in: Neural Networks, 2009. IJCNN 2009. International Joint Conference on, IEEE, 2009, pp. 3297–3302.

[95] J. Lu, P. Zhao, S. C. Hoi, Online sparse passive aggressive learning with kernels, in: Proceedings of the 2016 SIAM International Conference on Data Mining, SIAM, 2016, pp. 675–683.

[96] L. Zhang, R. Jin, C. Chen, J. Bu, X. He, Efficient online learning for large-scale sparse kernel logistic regression., in: AAAI, 2012.

[97] J. Wang, S. C. Hoi, P. Zhao, J. Zhuang, Z.-y. Liu, Large scale online kernel classification, in: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, AAAI Press, 2013, pp. 1750–1756.

[98] J. Lu, S. C. Hoi, J. Wang, P. Zhao, Z.-Y. Liu, Large scale online kernel learning, The Journal of Machine Learning Research.

[99] K.-P. Lin, M.-S. Chen, Efficient kernel approximation for large-scale support vector machine classification, in: Proceedings of the Eleventh SIAM International Conference on Data Mining, SIAM, 2011, pp. 211–222.

[100] S. Sonnenburg, V. Franc, Coffin: A computational framework for linear svms.

[101] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, C.-J. Lin, Training and testing low-degree polynomial data mappings via linear svm, The Journal of Machine Learning Research 11 (2010) 1471–1490.

[102] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Advances in neural information processing systems, 2007, pp. 1177–1184.

[103] C. K. I. Williams, M. Seeger, Using the nyström method to speed up kernel machines, in: T. Leen, T. Dietterich, V. Tresp (Eds.), Advances in Neural Information Processing Systems 13, MIT Press, 2001, pp. 682–688.

[104] T. Le, T. Nguyen, V. Nguyen, D. Phung, Dual space gradient descent for online learning, in: Advances In Neural Information Processing Systems, 2016, pp. 4583–4591.

[105] T. D. Nguyen, T. Le, H. Bui, D. Phung, Large-scale online kernel learning with random feature reparameterization, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17), 2017, pp. 2543–2549.

[106] S. Sonnenburg, G. Rätsch, C. Schäfer, B. Schölkopf, Large scale multiple kernel learning, J. Mach. Learn. Res. (JMLR) 7 (2006) 1531–1565.

[107] A. Rakotomamonjy, F. R. Bach, S. Canu, Y. Grandvalet, Simplemkl, J. Mach. Learn. Res. (JMLR) 11 (2008) 2491–2521.

[108] Z. Xu, R. Jin, I. King, M. R. Lyu, An extended level method for efficient multiple kernel learning, in: Advances in Neural Information Processing Systems (22), 2008.

[109] L. Jie, F. Orabona, M. Fornoni, B. Caputo, N. Cesa-bianchi, Om-2: An online multi-class multi-kernel learning algorithm, in: Proc. of the 4th IEEE Online Learning for Computer Vision Workshop, 2010.

[110] A. F. T. Martins, N. A. Smith, E. P. Xing, P. M. Q. Aguiar, M. A. T. Figueiredo, Online learning of structured predictors with multiple kernels, Journal of Machine Learning Research - Proceedings Track 15 (2011) 507–515.

[111] R. Jin, S. C. H. Hoi, T. Yang, Online multiple kernel learning: Algorithms and mistake bounds, in: Algorithmic Learning Theory, 21st International Conference, ALT 2010, Canberra, Australia, October 6-8, 2010. Proceedings, 2010, pp. 390–404.

[112] S. C. H. Hoi, R. Jin, P. Zhao, T. Yang, Online multiple kernel classification, Machine Learning 90 (2) (2013) 289–316.

[113] T. Yang, M. Mahdavi, R. Jin, J. Yi, S. C. Hoi, Online kernel selection: Algorithms and evaluations., in: AAAI, 2012.

[114] D. Sahoo, S. C. Hoi, B. Li, Online multiple kernel regression, in: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2014, pp. 293–302.

[115] D. Sahoo, A. Sharma, S. C. Hoi, P. Zhao, Temporal kernel descriptors for learning with time-sensitive patterns.

[116] D. Sahoo, P. Zhao, S. C. Hoi, Cost-sensitive online multiple kernel classification, in: Proceedings of The 8th Asian Conference on Machine Learning, 2016, pp. 65–80.

[117] J. Lu, D. Sahoo, P. Zhao, S. C. Hoi, Sparse passive-aggressive learning for bounded online kernel methods, ACM Transactions on Intelligent Systems and Technology (TIST) 9 (4) (2018) 45.

[118] K. Y. Levy, Online to offline conversions and adaptive minibatch sizes, in: Advances in Neural Information Processing Systems, 2017.

[119] N. Cesa-Bianchi, A. Conconi, C. Gentile, On the generalization ability of on-line learning algorithms, IEEE Transactions on Information Theory 50 (9) (2004) 2050–2057.

[120] N. Littlestone, From on-line to batch learning, in: Proceedings of the Second Annual Workshop on Computational Learning Theory, COLT 1989, Santa Cruz, CA, USA, July 31 - August 2, 1989., 1989, pp. 269–284.

[121] T. Zhang, Data dependent concentration bounds for sequential prediction algorithms, in: Learning Theory, 18th Annual Conference on Learning Theory, COLT, 2005, pp. 173–187.

[122] N. Cesa-Bianchi, C. Gentile, Improved risk tail bounds for on-line algorithms, IEEE Transactions on Information Theory 54 (1) (2008) 386–390.

[123] Z. Chen, Z. Fang, W. Fan, A. Edwards, K. Zhang, Cstg: An effective framework for cost-sensitive sparse online learning, in: Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, 2017, pp. 759–767.

[124] Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, J. Kandola, The perceptron algorithm with uneven margins, in: ICML, Vol. 2, 2002, pp. 379–386.

[125] W. Krauth, M. Mézard, Learning algorithms with optimal stability in neural networks, Journal of Physics A: Mathematical and General 20 (11) (1987) L745.

[126] J. Wang, P. Zhao, S. C. Hoi, Cost-sensitive online classification, Knowledge and Data Engineering, IEEE Transactions on 26 (10) (2014) 2425–2438.

[127] J. Wang, P. Zhao, S. C. H. Hoi, Cost-sensitive online classification, in: 12th IEEE International Conference on Data Mining (ICDM2012), 2012, pp. 1140–1145.

[128] P. Zhao, F. Zhuang, M. Wu, X.-L. Li, S. C. Hoi, Cost-sensitive online classification with adaptive regularization and its applications, in: Data Mining (ICDM), 2015 IEEE International Conference on, IEEE, 2015, pp. 649–658.

[129] S. Wang, L. L. Minku, X. Yao, Dealing with multiple classes in online class imbalance learning, International Joint Conferences on Artificial Intelligence, 2016.

[130] X. Zhang, T. Yang, P. Srinivasan, Online asymmetric active learning with imbalanced data, KDD.

[131] P. Zhao, S. C. Hoi, Cost-sensitive online active learning with application to malicious url detection, in: Proceedings of the

19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 919–927.
[132] J. Hu, H. Yang, I. King, M. R. Lyu, A. M.-C. So, Kernelized online imbalanced learning with fixed budgets., in: AAAI, 2015, pp. 2666–2672.
[133] P. Z. S. H. Yi Ding, Chenghao Liu, Large scale kernel methods for online auc maximization, in: The IEEE International Conference on Data Mining (ICDM), 2017.
[134] P. Zhao, R. Jin, T. Yang, S. C. Hoi, Online auc maximization, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 233–240.
[135] Y. Wang, R. Khardon, D. Pechyony, R. Jones, Generalization bounds for online learning algorithms with pairwise loss functions., in: COLT, Vol. 23, 2012, pp. 13–1.
[136] P. Kar, B. K. Sriperumbudur, P. Jain, H. C. Karnick, On the generalization ability of online learning algorithms for pairwise loss functions, in: ICML, 2013.
[137] W. Gao, R. Jin, S. Zhu, Z.-H. Zhou, One-pass auc optimization, in: ICML, 2013.
[138] Y. Ding, P. Zhao, S. C. Hoi, Y.-S. Ong, An adaptive gradient method for online auc maximization, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
[139] Y. Ying, L. Wen, S. Lyu, Stochastic online auc maximization, in: Advances in Neural Information Processing Systems, 2016, pp. 451–459.
[140] R. Caruana, Multitask learning, in: Learning to learn, Springer, 1998, pp. 95–133.
[141] S. J. Pan, Q. Yang, A survey on transfer learning, Knowledge and Data Engineering, IEEE Transactions on 22 (10) (2010) 1345–1359.
[142] O. Dekel, P. M. Long, Y. Singer, Online multitask learning, in: International Conference on Computational Learning Theory, Springer, 2006, pp. 453–467.
[143] G. Li, S. C. Hoi, K. Chang, R. Jain, Micro-blogging sentiment detection by collaborative online learning, in: Data Mining (ICDM), 2010 IEEE 10th International Conference on, IEEE, 2010, pp. 893–898.
[144] G. Li, S. C. Hoi, K. Chang, W. Liu, R. Jain, Collaborative online multitask learning, IEEE Transactions on Knowledge and Data Engineering 26 (8) (2014) 1866–1876.
[145] G. Cavallanti, N. Cesa-Bianchi, C. Gentile, Linear algorithms for online multitask classification, Journal of Machine Learning Research 11 (Oct) (2010) 2901–2934.
[146] A. Saha, P. Rai, H. D. I. S. Venkatasubramanian, Online learning of multiple tasks and their relationships, update 1 (1) (2011) 2.
[147] J. Wang, S. C. Hoi, P. Zhao, Z.-Y. Liu, Online multi-task collaborative filtering for on-the-fly recommender systems, in: Proceedings of the 7th ACM conference on Recommender systems, ACM, 2013, pp. 237–244.
[148] T. Evgeniou, M. Pontil, Regularized multi–task learning, in: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2004, pp. 109–117.
[149] H. Yang, I. King, M. R. Lyu, Online learning for multi-task feature selection, in: Proceedings of the 19th ACM international conference on Information and knowledge management, ACM, 2010, pp. 1693–1696.
[150] A. Kumar, H. Daume III, Learning task grouping and overlap in multi-task learning, arXiv preprint arXiv:1206.6417.
[151] K. Murugesan, H. Liu, J. Carbonell, Y. Yang, Adaptive smoothed online multi-task learning, in: Advances in Neural Information Processing Systems, 2016, pp. 4296–4304.
[152] S. Sun, A survey of multi-view machine learning, Neural Computing and Applications 23 (7-8) (2013) 2031–2038.
[153] C. Xu, D. Tao, C. Xu, A survey on multi-view learning, arXiv preprint arXiv:1304.5634.
[154] Y. Li, M. Yang, Z. Zhang, Multi-view representation learning: A survey from shallow methods to deep methods, arXiv preprint arXiv:1610.01206.
[155] T. T. Nguyen, K. Chang, S. C. Hui, Two-view online learning, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2012, pp. 74–85.
[156] J. Farquhar, D. Hardoon, H. Meng, J. S. Shawe-taylor, S. Szedmak, Two view learning: Svm-2k, theory and practice, in: Advances in neural information processing systems, 2006, pp. 355–362.
[157] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, C. Miao, Online multimodal deep similarity learning with application to image retrieval, in: Proceedings of the 21st ACM international conference on Multimedia, ACM, 2013, pp. 153–162.
[158] P. Wu, S. C. Hoi, P. Zhao, C. Miao, Z.-Y. Liu, Online multimodal distance metric learning with application to image retrieval, IEEE Transactions on Knowledge and Data Engineering 28 (2) (2016) 454–467.
[159] R. Sousa, L. M. Silva, L. A. Alexandre, J. Santos, J. M. de Sá, Transfer learning: Current status, trends and challenges.
[160] P. Zhao, S. C. Hoi, J. Wang, B. Li, Online transfer learning, Artificial Intelligence 216 (2014) 76–102.
[161] P. Zhao, S. C. Hoi, Otl: A framework of online transfer learning, in: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 2010, pp. 1231–1238.
[162] L. Ge, J. Gao, A. Zhang, Oms-tl: a framework of online multiple source transfer learning, in: Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, ACM, 2013, pp. 2423–2428.
[163] T. Tommasi12, F. Orabona, M. Kaboli, B. Caputo, C. Martigny, Leveraging over prior knowledge for online learning of visual categories.
[164] L. Ge, J. Gao, H. Ngo, K. Li, A. Zhang, On handling negative transfer and imbalanced distributions in multiple source transfer learning, Statistical Analysis and Data Mining: The ASA Data Science Journal 7 (4) (2014) 254–271.
[165] H. Bhatt, R. Singh, M. Vatsa, N. Ratha, Improving cross-resolution face matching using ensemble based co-transfer learning.
[166] H. S. Bhatt, R. Singh, M. Vatsa, N. Ratha, Matching cross-resolution face images using co-transfer learning, in: Image Processing (ICIP), 2012 19th IEEE International Conference on, IEEE, 2012, pp. 1453–1456.
[167] S. Hao, P. Zhao, Y. Liu, S. C. Hoi, C. Miao, Online multitask relative similarity learning, International Joint Conference on Artificial Intelligence.
[168] S. Shalev-Shwartz, Y. Singer, A. Y. Ng, Online and batch learning of pseudo-metrics, in: Proceedings of the twenty-first international conference on Machine learning, ACM, 2004, p. 94.
[169] R. Jin, S. Wang, Y. Zhou, Regularized distance metric learning: Theory and algorithm, in: Advances in neural information processing systems, 2009, pp. 862–870.
[170] J. V. Davis, B. Kulis, P. Jain, S. Sra, I. S. Dhillon, Information-theoretic metric learning, in: Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 209–216.
[171] P. Jain, B. Kulis, I. S. Dhillon, K. Grauman, Online metric learning and fast similarity search, in: Advances in neural information processing systems, 2009, pp. 761–768.
[172] H. Xia, S. C. Hoi, R. Jin, P. Zhao, Online multiple kernel similarity learning for visual search, IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (3) (2014) 536–549.
[173] X. Gao, S. C. Hoi, Y. Zhang, J. Wan, J. Li, Soml: Sparse on-

line metric learning with application to image retrieval, Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence.

[174] X. Gao, S. C. Hoi, Y. Zhang, J. Zhou, J. Wan, Z. Chen, J. Li, J. Zhu, Sparse online learning of image similarity, ACM Transactions on Intelligent Systems and Technology (TIST) 8 (5) (2017) 64.

[175] S. Hao, P. Zhao, S. C. Hoi, C. Miao, Learning relative similarity from data streams: Active online learning approaches, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 1181–1190.

[176] N. Chen, S. C. Hoi, S. Li, X. Xiao, Simapp: A framework for detecting similar mobile applications by online kernel learning, in: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, ACM, 2015, pp. 305–314.

[177] N. Chen, S. C. Hoi, S. Li, X. Xiao, Mobile app tagging, in: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, ACM, 2016, pp. 63–72.

[178] H. Xia, P. Wu, S. C. Hoi, Online multi-modal distance learning for scalable multimedia retrieval, in: Proceedings of the sixth ACM international conference on Web search and data mining, ACM, 2013, pp. 455–464.

[179] R. Heckel, K. Ramchandran, The sample complexity of online one-class collaborative filtering, in: International Conference on Machine Learning, 2017.

[180] Y. Shi, M. Larson, A. Hanjalic, Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges, ACM Computing Surveys (CSUR) 47 (1) (2014) 3.

[181] X. Su, T. M. Khoshgoftaar, A survey of collaborative filtering techniques, Advances in artificial intelligence 2009 (2009) 4.

[182] N. N. Liu, M. Zhao, E. Xiang, Q. Yang, Online evolutionary collaborative filtering, in: Proceedings of the fourth ACM conference on Recommender systems, ACM, 2010, pp. 95–102.

[183] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer (8) (2009) 30–37.

[184] J. Abernethy, K. Canini, J. Langford, A. Simma, Online collaborative filtering, University of California at Berkeley, Tech. Rep.

[185] L. Yuan-Xiang, L. Zhi-Jie, W. Feng, K. Li, Accelerated online learning for collaborative filtering and recommender systems, in: Data Mining Workshop (ICDMW), 2014 IEEE International Conference on, IEEE, 2014, pp. 879–885.

[186] C. Liu, S. C. Hoi, P. Zhao, J. Sun, E.-P. Lim, Online adaptive passive-aggressive methods for non-negative matrix factorization and its applications, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, 2016, pp. 1161–1170.

[187] J. Abernethy, F. Bach, T. Evgeniou, J.-P. Vert, Low-rank matrix factorization with attributes, arXiv preprint cs/0611124.

[188] G. Ling, H. Yang, I. King, M. R. Lyu, Online learning for collaborative filtering, in: Neural Networks (IJCNN), The 2012 International Joint Conference on, IEEE, 2012, pp. 1–8.

[189] M. Ali, C. C. Johnson, A. K. Tang, Parallel collaborative filtering for streaming data, University of Texas Austin, Tech. Rep.

[190] A. S. Das, M. Datar, A. Garg, S. Rajaram, Google news personalization: scalable online collaborative filtering, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 271–280.

[191] C. Liu, T. Jin, S. C. Hoi, P. Zhao, J. Sun, Collaborative topic regression for online recommender systems: an online and bayesian approach, Machine Learning 106 (5) (2017) 651–670.

[192] A. Trotman, Learning to rank, Information Retrieval 8 (3) (2005) 359–381.

[193] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, H. Li, Learning to rank: from pairwise approach to listwise approach, in: Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 129–136.

[194] L. Hang, A short introduction to learning to rank, IEICE TRANSACTIONS on Information and Systems 94 (10) (2011) 1854–1862.

[195] M. Zoghi, T. Tunys, M. Ghavamzadeh, B. Kveton, C. Szepesvari, Z. Wen, Online learning to rank in stochastic click models, in: International Conference on Machine Learning, 2017, pp. 4199–4208.

[196] P. Shah, A. Soni, T. Chevalier, Online ranking with constraints: A primal-dual algorithm and applications to web traffic-shaping, in: KDD, 2017.

[197] J. Wang, J. Wan, Y. Zhang, S. C. Hoi, Solar: Scalable online learning algorithms for ranking, ACL, 2015.

[198] J. Wan, P. Wu, S. C. Hoi, P. Zhao, X. Gao, D. Wang, Y. Zhang, J. Li, Online learning to rank for content-based image retrieval., in: IJCAI, 2015, pp. 2284–2290.

[199] K. Crammer, Y. Singer, et al., Pranking with ranking., in: Nips, Vol. 1, 2001, pp. 641–647.

[200] K. Crammer, Y. Singer, Online ranking by projecting, Neural Computation 17 (1) (2005) 145–175.

[201] E. F. Harrington, Online ranking/collaborative filtering using the perceptron algorithm, in: ICML, Vol. 20, 2003, pp. 250–257.

[202] R. Herbrich, T. Graepel, K. Obermayer, Support vector learning for ordinal regression.

[203] O. Chapelle, S. S. Keerthi, Efficient algorithms for ranking with svms, Information Retrieval 13 (3) (2010) 201–215.

[204] W. Zhang, P. Zhao, W. Zhu, S. C. Hoi, T. Zhang, Projection-free distributed online learning in networks, in: International Conference on Machine Learning, 2017, pp. 4054–4062.

[205] A. Agarwal, M. J. Wainwright, J. C. Duchi, Distributed dual averaging in networks, in: Advances in Neural Information Processing Systems, 2010, pp. 550–558.

[206] J. F. Mota, J. M. Xavier, P. M. Aguiar, M. Püschel, D-admm: A communication-efficient distributed algorithm for separable optimization, IEEE Transactions on Signal Processing 61 (10) (2013) 2718–2723.

[207] P. Smyth, M. Welling, A. U. Asuncion, Asynchronous distributed learning of topic models, in: Advances in Neural Information Processing Systems, 2009, pp. 81–88.

[208] R. J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural computation 1 (2) (1989) 270–280.

[209] J. Platt, A resource-allocating network for function interpolation, Neural computation 3 (2) (1991) 213–225.

[210] G. A. Carpenter, S. Grossberg, J. H. Reynolds, Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network, Neural networks 4 (5) (1991) 565–588.

[211] Y. A. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, Efficient backprop, in: Neural networks: Tricks of the trade, Springer, 1998, pp. 9–48.

[212] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, Neural Networks, IEEE Transactions on 17 (6) (2006) 1411–1423.

[213] R. Polikar, L. Upda, S. S. Upda, V. Honavar, Learn++: An incremental learning algorithm for supervised neural networks, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 31 (4) (2001) 497–508.

[214] L. Bottou, Online algorithms and stochastic approximations, in: D. Saad (Ed.), Online Learning and Neural Networks, Cambridge University Press, Cambridge, UK, 1998, revised, oct

2012.
[215] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, Neural Networks, IEEE Transactions on 22 (10) (2011) 1517–1531.
[216] D. Sahoo, Q. Pham, J. Lu, S. C. Hoi, Online deep learning: Learning deep neural networks on the fly, arXiv preprint arXiv:1711.03705.
[217] B. Li, S. C. Hoi, Online portfolio selection: A survey, ACM Computing Surveys (CSUR) 46 (3) (2014) 35.
[218] B. Li, S. C. H. Hoi, Online Portfolio Selection: Principles and Algorithms, Crc Press, 2015.
[219] B. Li, D. Sahoo, S. C. Hoi, Olps: a toolbox for on-line portfolio selection, Journal of Machine Learning Research 17 (35) (2016) 1–5.
[220] J. L. Kelly Jr, A new interpretation of information rate, in: The Kelly Capital Growth Investment Criterion: Theory and Practice, World Scientific, 2011, pp. 25–34.
[221] T. M. Cover, Universal portfolios, in: The Kelly Capital Growth Investment Criterion: Theory and Practice, World Scientific, 2011, pp. 181–209.
[222] D. P. Helmbold, R. E. Schapire, Y. Singer, M. K. Warmuth, On-line portfolio selection using multiplicative updates, Mathematical Finance 8 (4) (1998) 325–347.
[223] A. A. Gaivoronski, F. Stella, Stochastic nonstationary optimization for finding universal portfolios, Annals of Operations Research 100 (1) (2000) 165–188.
[224] A. Agarwal, E. Hazan, S. Kale, R. E. Schapire, Algorithms for portfolio management based on the newton method, in: Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 9–16.
[225] A. Borodin, R. El-Yaniv, V. Gogan, Can we learn to beat the best stock, in: Advances in Neural Information Processing Systems, 2004, pp. 345–352.
[226] B. Li, P. Zhao, S. C. Hoi, V. Gopalkrishnan, Pamr: Passive aggressive mean reversion strategy for portfolio selection, Machine learning 87 (2) (2012) 221–258.
[227] B. Li, S. C. Hoi, P. Zhao, V. Gopalkrishnan, Confidence weighted mean reversion strategy for on-line portfolio selection, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 434–442.
[228] B. Li, S. C. Hoi, P. Zhao, V. Gopalkrishnan, Confidence weighted mean reversion strategy for online portfolio selection, ACM Transactions on Knowledge Discovery from Data (TKDD) 7 (1) (2013) 4.
[229] B. Li, S. C. Hoi, On-line portfolio selection with moving average reversion, arXiv preprint arXiv:1206.4626.
[230] B. Li, S. C. Hoi, D. Sahoo, Z.-Y. Liu, Moving average reversion strategy for on-line portfolio selection, Artificial Intelligence 222 (2015) 104–123.
[231] D. Huang, J. Zhou, B. Li, S. C. Hoi, S. Zhou, Robust median reversion strategy for on-line portfolio selection, in: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, AAAI Press, 2013, pp. 2006–2012.
[232] L. Gyorfi, D. Schafer, Nonparametric prediction, NATO SCIENCE SERIES SUB SERIES III COMPUTER AND SYSTEMS SCIENCES 190 (2003) 341–356.
[233] L. Györfi, F. Udina, H. Walk, Nonparametric nearest neighbor based empirical portfolio selection strategies, Statistics & Decisions International mathematical journal for stochastic methods and models 26 (2) (2008) 145–157.
[234] B. Li, S. C. Hoi, V. Gopalkrishnan, Corn: Correlation-driven nonparametric learning approach for portfolio selection, ACM Transactions on Intelligent Systems and Technology (TIST) 2 (3) (2011) 21.
[235] V. Vovk, C. Watkins, Universal portfolio selection, in: Proceedings of the eleventh annual conference on Computational learning theory, ACM, 1998, pp. 12–23.
[236] K. Akcoglu, P. Drineas, M.-Y. Kao, Fast universalization of investment strategies, SIAM Journal on Computing 34 (1) (2004) 1–22.
[237] E. Hazan, C. Seshadhri, Efficient learning algorithms for changing environments, in: Proceedings of the 26th annual international conference on machine learning, ACM, 2009, pp. 393–400.
[238] M. Ormos, A. Urbán, Performance analysis of log-optimal portfolio strategies with transaction costs, Quantitative Finance 13 (10) (2013) 1587–1597.
[239] D. Huang, Y. Zhu, B. Li, S. Zhou, S. C. Hoi, Semi-universal portfolios with transaction costs.
[240] B. Li, J. Wang, D. Huang, S. C. Hoi, Transaction cost optimization for online portfolio selection, Quantitative Finance (2017) 1–14.
[241] B. George, Time Series Analysis: Forecasting & Control, 3/e, Pearson Education India, 1994.
[242] M. Clements, D. Hendry, Forecasting economic time series, Cambridge University Press, 1998.
[243] C. Chatfield, Time-series forecasting, CRC Press, 2000.
[244] N. I. Sapankevych, R. Sankar, Time series prediction using support vector machines: a survey, Computational Intelligence Magazine, IEEE 4 (2) (2009) 24–38.
[245] O. Anava, E. Hazan, S. Mannor, O. Shamir, Online learning for time series prediction, arXiv preprint arXiv:1302.6927.
[246] C. Liu, S. C. Hoi, P. Zhao, J. Sun, Online arima algorithms for time series prediction.
[247] M. N. Katehakis, A. F. Veinott Jr, The multi-armed bandit problem: decomposition and computation, Mathematics of Operations Research 12 (2) (1987) 262–268.
[248] J. Vermorel, M. Mohri, Multi-armed bandit algorithms and empirical evaluation, in: Machine Learning: ECML 2005, Springer, 2005, pp. 437–448.
[249] J. Gittins, K. Glazebrook, R. Weber, Multi-armed bandit allocation indices, John Wiley & Sons, 2011.
[250] S. R. Chowdhury, A. Gopalan, On kernelized multi-armed bandits, in: International Conference on Machine Learning, 2017.
[251] C. Gentile, S. Li, P. Kar, A. Karatzoglou, G. Zappella, E. Etrue, On context-dependent clustering of bandits, in: International Conference on Machine Learning, 2017, pp. 1253–1262.
[252] L. Li, Y. Lu, D. Zhou, Provable optimal algorithms for generalized linear contextual bandits, in: International Conference on Machine Learning, 2017.
[253] K.-S. Jun, A. Bhargava, R. Nowak, R. Willett, Scalable generalized linear bandits: Online computation and hashing, in: Advances in Neural Information Processing Systems, 2017.
[254] L. Zhang, T. Yang, R. Jin, Y. Xiao, Z.-H. Zhou, Online stochastic linear optimization under one-bit feedback, in: International Conference on Machine Learning, 2016, pp. 392–401.
[255] S. Bubeck, N. Cesa-Bianchi, Regret analysis of stochastic and nonstochastic multi-armed bandit problems, arXiv preprint arXiv:1204.5721.
[256] P. Auer, N. Cesa-Bianchi, P. Fischer, Finite-time analysis of the multiarmed bandit problem, Machine learning 47 (2-3) (2002) 235–256.
[257] N. Cesa-Bianchi, G. Lugosi, Combinatorial bandits, Journal of Computer and System Sciences 78 (5) (2012) 1404–1422.
[258] R. Combes, M. S. T. M. Shahi, A. Proutiere, et al., Combinatorial bandits revisited, in: Advances in Neural Information Processing Systems, 2015, pp. 2116–2124.
[259] W. Chen, Y. Wang, Y. Yuan, Q. Wang, Combinatorial multi-armed bandit and its extension to probabilistically triggered

arms, Journal of Machine Learning Research 17 (50) (2016) 1–33.
[260] C. Zeng, Q. Wang, S. Mokhtari, T. Li, Online context-aware recommendation with time varying multi-arm bandit, KDD.
[261] L. Li, W. Chu, J. Langford, R. E. Schapire, A contextual-bandit approach to personalized news article recommendation, in: Proceedings of the 19th international conference on World wide web, ACM, 2010, pp. 661–670.
[262] L. Zhou, A survey on contextual multi-armed bandits, arXiv preprint arXiv:1508.03326.
[263] W. Chu, L. Li, L. Reyzin, R. E. Schapire, Contextual bandits with linear payoff functions., in: AISTATS, Vol. 15, 2011, pp. 208–214.
[264] P. Auer, Using confidence bounds for exploitation-exploration trade-offs, Journal of Machine Learning Research 3 (Nov) (2002) 397–422.
[265] S. M. Kakade, S. Shalev-Shwartz, A. Tewari, Efficient bandit algorithms for online multiclass prediction, in: Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008, 2008, pp. 440–447.
[266] G. Chen, G. Chen, J. Zhang, S. Chen, C. Zhang, Beyond banditron: A conservative and efficient reduction for online multiclass prediction with bandit setting model, in: ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009, 2009, pp. 71–80.
[267] S. Wang, R. Jin, H. Valizadegan, A potential-based framework for online multi-class learning with partial feedback, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 900–907.
[268] K. Crammer, C. Gentile, Multiclass classification with bandit feedback using adaptive regularization, in: Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011, 2011, pp. 273–280.
[269] E. Hazan, S. Kale, Newtron: an efficient bandit algorithm for online multiclass prediction, in: Advances in Neural Information Processing Systems, 2011, pp. 891–899.
[270] A. Beygelzimer, F. Orabona, C. Zhang, Efficient online bandit multiclass learning with $\sqrt{T}$ regret, in: International Conference on Machine Learning, 2017.
[271] P. Auer, N. Cesa-Bianchi, Y. Freund, R. E. Schapire, The nonstochastic multiarmed bandit problem, SIAM Journal on Computing 32 (1) (2002) 48–77.
[272] T. Uchiya, A. Nakamura, M. Kudo, Algorithms for adversarial bandit problems with multiple plays, in: International Conference on Algorithmic Learning Theory, Springer, 2010, pp. 375–389.
[273] J.-Y. Audibert, S. Bubeck, G. Lugosi, Regret in online combinatorial optimization, Mathematics of Operations Research 39 (1) (2013) 31–45.
[274] S. Bubeck, N. Cesa-Bianchi, S. M. Kakade, et al., Towards minimax policies for online linear optimization with bandit feedback., in: COLT, Vol. 23, 2012.
[275] A. Beygelzimer, J. Langford, L. Li, L. Reyzin, R. E. Schapire, Contextual bandit algorithms with supervised learning guarantees., in: AISTATS, 2011, pp. 19–26.
[276] G. Cavallanti, N. Cesa-Bianchi, C. Gentile, Linear classification and selective sampling under low noise conditions, in: Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008, 2008, pp. 249–256.
[277] N. Cesa-Bianchi, A. Conconi, C. Gentile, Learning probabilistic linear-threshold classifiers via selective sampling, in: Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, 2003, pp. 373–387.
[278] N. Cesa-Bianchi, C. Gentile, F. Orabona, Robust bounds for classification via selective sampling, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML2009, 2009, pp. 121–128.
[279] O. Dekel, C. Gentile, K. Sridharan, Robust selective sampling from single and multiple teachers, in: COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010, 2010, pp. 346–358.
[280] F. Orabona, N. Cesa-Bianchi, Better algorithms for selective sampling, in: Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011, 2011, pp. 433–440.
[281] N. Cesa-Bianchi, C. Gentile, L. Zaniboni, Worst-case analysis of selective sampling for linear-threshold algorithms, in: Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada], 2004.
[282] N. Cesa-Bianchi, C. Gentile, L. Zaniboni, Worst-case analysis of selective sampling for linear classification, Journal of Machine Learning Research 7 (2006) 1205–1230.
[283] J. Lu, P. Zhao, S. C. Hoi, Online passive aggressive active learning and its applications, The 6th Asian Conference on Machine Learning (ACML2014).
[284] J. Lu, P. Zhao, S. C. Hoi, Online passive-aggressive active learning, Machine Learning 103 (2) (2016) 141–183.
[285] S. Hao, P. Zhao, J. Lu, S. C. Hoi, C. Miao, C. Zhang, Soal: Second-order online active learning, in: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, 2016, pp. 931–936.
[286] S. Hao, J. Lu, P. Zhao, C. Zhang, S. C. Hoi, C. Miao, Second-order online active learning and its applications, IEEE Transactions on Knowledge and Data Engineering.
[287] P. Zhao, S. C. Hoi, Cost-sensitive online active learning with application to malicious url detection, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 919–927.
[288] P. Zhao, S. C. H. Hoi, J. Zhuang, Active learning with expert advice, in: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, Bellevue, WA, USA, August 11-15, 2013, 2013.
[289] X. Zhu, Semi-supervised learning literature survey.
[290] X. Zhu, A. B. Goldberg, Introduction to semi-supervised learning, Synthesis lectures on artificial intelligence and machine learning 3 (1) (2009) 1–130.
[291] M. Belkin, P. Niyogi, V. Sindhwani, Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, Journal of machine learning research 7 (Nov) (2006) 2399–2434.
[292] A. B. Goldberg, M. Li, X. Zhu, Online manifold regularization: A new learning setting and empirical study, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2008, pp. 393–407.
[293] K. Bennett, A. Demiriz, et al., Semi-supervised support vector machines, Advances in Neural Information processing systems (1999) 368–374.
[294] M.-S. Chen, T.-Y. Ho, D.-Y. Huang, Online transductive support vector machines for classification, in: Information Security and Intelligence Control (ISIC), 2012 International Conference on, IEEE, 2012, pp. 258–261.
[295] R. S. Michalski, I. Mozetic, J. Hong, N. Lavrac, The multipurpose incremental learning system aq15 and its testing application to three medical domains, Proc. AAAI 1986 (1986)

1–041.

[296] J. Read, A. Bifet, B. Pfahringer, G. Holmes, Batch-incremental versus instance-incremental learning in dynamic and evolving data, in: Advances in Intelligent Data Analysis XI, Springer, 2012, pp. 313–323.

[297] H. Wang, W. Fan, P. S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2003, pp. 226–235.

[298] T. G. Dietterich, Machine learning for sequential data: A review, in: Structural, syntactic, and statistical pattern recognition, Springer, 2002, pp. 15–30.

[299] G. BakIr, Predicting structured data, MIT press, 2007.

[300] D. Roth, K. Small, I. Titov, Sequential learning of classifiers for structured prediction problems, in: International Conference on Artificial Intelligence and Statistics, 2009, pp. 440–447.

[301] L. Bottou, Stochastic learning, in: Advanced lectures on machine learning, Springer, 2004, pp. 146–168.

[302] T. Zhang, Solving large scale linear prediction problems using stochastic gradient descent algorithms, in: Proceedings of the twenty-first international conference on Machine learning, ACM, 2004, p. 116.

[303] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[304] O. Bousquet, L. Bottou, The tradeoffs of large scale learning, in: Advances in neural information processing systems, 2008, pp. 161–168.

[305] M. Ware, E. Frank, G. Holmes, M. Hall, I. H. Witten, Interactive machine learning: letting users build classifiers, International Journal of Human-Computer Studies 55 (3) (2001) 281–292.

[306] D. Johnson, S. Levesque, T. Zhang, Interactive machine learning system for automated annotation of information in text, uS Patent App. 10/630,854 (Jul. 31 2003).

[307] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, D. B. Rosen, Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps, Neural Networks, IEEE Transactions on 3 (5) (1992) 698–713.

[308] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, Journal of artificial intelligence research (1996) 237–285.

[309] A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.

[310] A. G. BARTO, T. G. DIETTERICH, Reinforcement learning and its relationship to supervised learning, Handbook of learning and approximate dynamic programming 2 (2004) 47.