Online Learning in Machine Learning

Abhilasha Mohania

***Abstract*— This works gives an overview of employ of On-line Learning in Machine Learning. On-line Algorithms are crucial in the field of Machine Learning specially at-least in two scenarios, which makes the Machine Learning application faster and memory efficient than batch Learning Algorithms(Off-line Learning). The primary goal of this seminar is to learn "what On-line Learning is", why this technique is crucial in the field of Machine Learning and what is the computational aspects of this algorithms.**

I. INTRODUCTION

Learning is the ability to improve performance by observing data. Consider, an agent in an environment who has a task to do. The agent may be any forecaster or algorithm, the environment may be adversarial in nature and the task is learning the setup, if environment is unknown or optimize the performance, if it is known.
Learning can be broadly described in two ways:

- **Batch Learning:** In Batch Learning, Algorithm has a access to all the dataset together to train a model and correspondingly finds the hypothesis function for the dataset. This Learning is also known as off-line learning.
- **On-line Learning:** On-line machine learning is a method of machine learning in which data becomes available in a sequential order and is used to update our best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once.

Online learning is a common technique used in areas of machine learning where it is computationally infeasible to train over the entire dataset, requiring the need of out-of-core algorithms. It is also used in situations where it is necessary for the algorithm to dynamically adapt to new patterns in the data, or when the data itself is generated as a function of time, e.g. stock price prediction.

II. ONLINE LEARNING FRAMEWORK

A. Basic Framework

Online learning is performed in a sequence of consecutive rounds, where at round t the learner is given a question, x_t, taken from an instance domain X , and is required to provide an answer to this question, which we denote by pt. After predicting an answer, the correct answer, y_t, taken from a target domain Y, is revealed and the learner suffers a loss, $l(p_t, y_t)$, which measures the discrepancy between his answer and the correct one. While in many cases p_t is in Y, it is sometimes convenient to allow the learner to pick a prediction from a larger set, which we denote by D.

*This work was supported by Seminar Computational Aspects of Machine Learning

[1]Kilian Roehner is the supervisor , Technical University of Munich.

B. Pseudo-Algorithm

Online Learning

for $t = 1, 2, \ldots$
  receive question $\mathbf{x}_t \in \mathcal{X}$
  predict $p_t \in D$
  receive true answer $y_t \in \mathcal{Y}$
  suffer loss $l(p_t, y_t)$

Fig. 1. On-line Learning Framework

III. GOAL OF ONLINE ALGORITHMS

The learners ultimate goal is to minimize the cumulative loss suffered along its run, which translates to making few prediction mistakes in the classification case. The learner tries to deduce information from previous rounds so as to improve its predictions on present and future questions. Clearly, learning is hopeless if there is no correlation between past and present rounds. Classic statistical theory of sequential prediction therefore enforces strong assumptions on the statistical properties of the input sequence (e.g., that it is sampled i.i.d. according to some unknown distribution). Online methods which make no statistical assumptions regarding the origin of the sequence of examples. The sequence is allowed to be deterministic, stochastic, or even adversarially adaptive to the learners own behavior (as in the case of spam email filtering). Naturally, an adversary can make the cumulative loss to our online algorithm arbitary large. For example, the adversary can ask the same question on each online round, wait for the learners answer, and provide the opposite answer as the correct answer. To make nontrivial statements we must further restrict the problem. We consider two natural restrictions.

A. Mistake-Bound

The first restriction is especially suited to the case of online classification. We assume that all the answers are generated by some target mapping, h: $X - Y$. Furthermore, h^* taken from a fixed set, called a hypothesis class and denoted by H, which is known to the learner. With this restriction on the sequence, which we call the realizable case, the learner should make as few mistakes as possible, assuming that both hand the sequence of questions can be chosen by an

adversary. For an online learning algorithm, A, we denote by $M_A(H)$ the maximal number of mistakes A might make on a sequence of examples which is labeled by some $h \in H$. We emphasize again that both h and the sequence of questions can be chosen by an adversary. A bound on $M_A(H)$ is called a mistake-bound.

B. Regret

Alternatively, the second restriction of the online learning model we consider is a relaxation of the realizable assumption. We no longer assume that all answers are generated by some $h \in H$, but we require the learner to be competitive with the best fixed predictor from H. This is captured by the regret of the algorithm, which measures how sorry the learner is, in retrospect, not to have followed the predictions of some hypothesis $h \in H$. Formally, the regret of the algorithm relative to h when running on a sequence of T examples is defined as:

$$Regret_T(h_*) = \sum_{t=1}^{T} l(p_t, y_t) - \sum_{t=1}^{T} l(h_*(x_t, y_t)$$

and the regret of the algorithm relative to a hypothesis class H is

$$Regret_T(H) = max_{h^* \in H}\ Regret(h_*)$$

We restate the learners goal as having the lowest possible regret relative to H. We will sometime be satisfied with low regret algorithms, by which we mean that $Regret_T(H)$ grows sub-linearly with the number of rounds, T, which implies that the difference between the average loss of the learner and the average loss of the best hypothesis in H tends to zero as T goes to infinity.

IV. Examples

To make the discussion more concrete, we list several additional online prediction problems and possible hypothesis classes.

A. Spam Detection

In Spam Detection, emails arrive one by one. Predictor classifies one email at a time as a Spam or not Spam. Sometimes it gets feedback from the users and suffer some loss $l(y_t, \hat{y}_t)$. The algorithm use this information of computation loss and again update hypothesis.

B. Recommendation Systems

Recommendation Systems is very hot topic this days. from Amazon to Netflix, this websites always attempt to customize their sites by the customers and recommendation is one of aspect that contribute to their profit as well. These automated systems perform well in recommending which is quite similar to our taste. At the server side, a predictor constantly attempt to predict what to recommend.

C. Dynamic Pricing

Dynamic pricing is one of the interesting application of Online Learning. In this, predictor attempts to adjust the price of selling items based on whether customer buys it or not buys it. e.g ebay.

D. Finance

Entire Financial Data is generated as a function of Time e.g stock forecast, also online learning technique is useful in trading decision and gambling application in financial domain.

E. adversarial Machine Learning

Online Algorithms are useful in adversarial environment, where environment constantly tries to oppose predictor, which makes online Learning very famous in information security and games.

F. Computer Vision

Computer vision research areas are highly relying on online algorithms especially in the field of object detection and tracking, in addition to it on-line object classification.e.g Autonomous Driving.

V. Approaches

Various Approaches have been discussed in the area of Online Learning to solve different problems, which are mainly:

- Online Learnring from expert advise.
- Online Learning from examples.
- General Algorithms that can be used in Online setting.

A. Online Learnring from expert advise

On each online round the learner has to choose from the advice of d given experts. Therefore,$x_t \in X \subset R^d$, where $x_t[i]$ is the advice of the i^{th} expert, and D = {1,...,d}. Then, the learner receives the true answer, which is a vector $y_t \in Y = [0,1]^d$, where $y_t[i]$ is the cost of following the advice of the i^{th} expert. The loss of the learner is the cost of the chosen expert, $(p_t, y_t) = y_t[p_t]$. A common hypothesis class for this problem is the set of constant predictors, H = $\{h_1, ..., h_d\}$, where $h_i(x)$ = i for all x. This implies that the regret of the algorithm is measured relative to the performance of the strategies which always predict according to the same expert. various algorithm have been discussed in expert advice approach:

- Weighted Majority Algorithm(WMAJ)
- Exponential Majority Algorithm
- Randomized Exponential Majority Algorithm.

Weighted Majority algorithm mainly have been discussed.

*1) **Weighted Majority Algorithm**:* Imagine the process of picking good times to invest in a stock. For simplicity, assume that there is a single stock of interest, and its daily price movement is modeled as a sequence of binary events: up/down. Each morning we try to predict whether the price will go up or down that day; if our prediction happens to be wrong we lose a dollar that day, and if its correct, we lose nothing. The stock movements can be arbitrary and even adversarial. To balance out this pessimistic assumption, we assume that while making our predictions, we are allowed to watch the predictions of n experts. These experts could be arbitrarily correlated, and they may or may not know what they are talking about. The algorithms goal is to limit

its cumulative losses (i.e., bad predictions) to roughly the same as the best of these experts. At first sight this seems an impossible goal, since it is not known until the end of the sequence who the best expert was, whereas the algorithm is required to make predictions all along. For example, the first algorithm one thinks of is to compute each days up/down prediction by going with the majority opinion among the experts that day. But, this algorithm does not work because a majority of experts may be consistently wrong on every single day. The weighted majority algorithm corrects the trivial algorithm. It maintains a weighting of the experts. Initially all have equal weight. As time goes on, some experts are seen as making better predictions than others, and the algorithm increases their weight proportionately. The algorithms prediction of up/down for each day is computed by going with the opinion of the weighted majority of the experts for that day.

Pseudo-Algorithm: **Initialization:** fix an $\eta \leq \frac{1}{2}$

For t = 1,2,...,T :

1. Make the prediction that is the weighted majority of the experts predictions based on the weights $w_1(t),...,w_n(t)$. That is, predict up or down depending on which prediction has a higher total weight of experts advising it (breaking ties arbitrarily).

2. For every expert i who predicts wrongly, decrease his weight for the next round by multiplying it by a factor of $(1-\eta)$:

$$w_i^{(t+1)} = (1-\eta)w_i^t \ (update\ rule)$$

- **Theorem 1:** After T steps, let $m_i(T)$ be the number of mistakes of expert i and M(T) be the number of mistakes our algorithm has made. Then we have the following bound for every i:

$$M^T \leq 2(1+\eta)m_i^T + \frac{2\log(n)}{\eta}$$

In particular, this holds for i which is the best expert, i.e. having the least $m_i(T)$

B. Online Learning from examples

Learning from examples is different from using Expert advice, as we don't need to previously define rebuild experts we will derive our predictions from. We need , however, to know what Concept Class we want to search over. A concept class is a set of functions (concepts) that subscribe to a particular model. The possible Concept Classes are below:

- The set of all monotone disjunction of N variables
- The set of non monotone disjunction of N variables
- Decision list with N variable
- Linear threshold formulas
- DNF(Disjunctive normal form) formula

The Winnow Algorithm is one of the simple example that leads monotone disjunctions online. It learns any concept(function), provided the concept belongs to concept class of monotone disjunctions.

The winnow algorithm is a technique from machine learning for learning a linear classifier from labeled examples. It is very similar to the perceptron algorithm. However, the perceptron algorithm uses an additive weight-update scheme, while Winnow uses a multiplicative scheme that allows it to perform much better when many dimensions are irrelevant (hence its name). It is a simple algorithm that scales well to high-dimensional data. During training, Winnow is shown a sequence of positive and negative examples. From these it learns a decision hyperplane that can then be used to label novel examples as positive or negative. The algorithm can also be used in the online learning setting, where the learning and the classification phase are not clearly separated.

The basic algorithm, Winnow, is as follows. The instance space is $X = \{0,1\}^n$, that is, each instance is described as a set of Boolean-valued features. The algorithm maintains non-negative weights w_i for $i \in \{1,\ldots,n\}$, which are initially set to 1, one weight for each feature. When the learner is given an example $(x_1,\ldots,x_n)$, it applies the typical prediction rule for linear classifiers:

- if $\sum_{i=1}^n w_i x_i \geq \theta$, then predict 1.
- Otherwise predict 0

Here Θ is a real number that is called the threshold. Together with the weights, the threshold defines a dividing hyperplane in the instance space. Good bounds are obtained if $\Theta = n/2$.

For each example with which it is presented, the learner applies the following update rule:

- If an example is correctly classified, do nothing.
- If an example is predicted incorrectly and the correct result was 0, for each feature $x_i = 1$, the corresponding weight w_i is set to 0 (demotion step):
  $\forall x_i = 1, w_i = 0$
- If an example is predicted incorrectly and the correct result was 1, for each feature $x_i = 1$, the corresponding weight w_i multiplied by (promotion step):
  $\forall x_i = 1, w_i = \alpha w_i$
- Mistake Bound for Winnow Algorithm is $O(k\log_2 d)$, Where d is number of feature attributes.

C. Online Learning from General Algorithm

1) Stochastic Gradient Descent: Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration.

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$$

where the parameter w which minimizes Q(w) is to be estimated. In stochastic (or "on-line") gradient descent, the true gradient of Q(w) is approximated by a gradient at a

single example:

$$w := w - \eta \nabla Q_i(w).$$

As the algorithm sweeps through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm converges. If this is done, the data can be shuffled for each pass to prevent cycles. Typical implementations may use an adaptive learning rate so that the algorithm converges.

In pseudocode, stochastic gradient descent can be presented as follows:

- choose an initial vector of parameters w and learning rate η
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For i=1,2,...,n do:
    - w:=w-$\eta \nabla Q_i(w)$

The convergence of stochastic gradient descent has been analyzed using the theories of convex minimization and of stochastic approximation. Briefly, when the learning rates η decrease with an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges almost surely to a global minimum when the objective function is convex or pseudoconvex, and otherwise converges almost surely to a local minimum.

2) Perceptron Online Algorithm: In machine learning, the Perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not).It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

Perceptron Online Pseudo-Algorithm: We first define some variables:

- $y = f(\mathbf{z})$ denotes the output from the Perceptron for an input vector $\mathbf{z}$.
- $D = (\mathbf{x}_1, d_1), \ldots, (\mathbf{x}_s, d_s)$ is the training set of s samples, where:
  - $\mathbf{x}_j$ is the n-dimensional input vector.
  - d_j is the desired output value of the perceptron for that input.

We show the values of the features as follows:

- $x_{j,i}$ is the value of the i^{th} feature of the j^{th} training input vector.
- $x_{j,0}$=1.

To represent the weights:

- w_i is the i^{th} value in the weight vector, to be multiplied by the value of the i^{th} input feature.
- Because $x_{j,0} = 1x_{j,0} = 1$, w_0 is effectively a bias that we use instead of the bias constant b.

To show the time-dependence of $\mathbf{w}$, we use:

- $w_i(t)$ is the weight i at time t.

Unlike other linear classification algorithms such as logistic regression, there is no need for a learning rate in the perceptron algorithm. This is because multiplying the update by any constant simply rescales the weights but never changes the sign of the prediction.

Steps:

- Initialize the weights and the threshold. Weights may be initialized to 0 or to a small random value. In the example below, we use 0.
- For each example j in our training set D, perform the following steps over the input $\mathbf{x}_j$ $\mathbf{x}_j$ and desired output d_j :
- Calculate the actual output:

$$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j]$$
$$= f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]$$

- Update the weights:

$$w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{j,i}, for\ all\ features\ 0 \le i \le n.$$

The Perceptron is a linear classifier, therefore it will never get to the state with all the input vectors classified correctly if the training set D is not linearly separable, i.e. if the positive examples can not be separated from the negative examples by a hyperplane. In this case, no "approximate" solution will be gradually approached under the standard learning algorithm, but instead learning will fail completely. But if the training set is linearly separable, then the Perceptron is guaranteed to converge, and there is an upper bound on the number of times the Perceptron will adjust its weights during the training.

VI. Computational Aspects of On-line K-Mean Algorithm

As we discussed, The On-line Algorithms are fast and memory efficient. Let's see the comparison for K-mean batch and on-line Algorithm. This practical comparison has been performed in Python using Sci-kit Learn Library. Following are the results for both the Algorithm: The Sequential K-Mean Algorithm is Faster as well as memory efficient as compared to K-mean Algorithm but K-mean Algorithm converges better than sequential K-mean.[fig.1, Fig.2]. Train time for k-mean is 0.39s, whereas for sequential K-mean is 0.03s, whereas Inertia for k-mean is 41825.153264 and for sequential K-mean is 44262.894798. Also, memory required to fit the data by k-mean is 8 Mb, whereas for sequential k-mean is 2 Mb for data size of around 10 Mb.

VII. Online Convex optimization

In recent years, the design of many efficient online learning algorithms has been influenced by convex optimization tools. In this section we will discuss convexification techniques, showing how to utilize the online convex optimization framework in nonconvex problems. online prediction problems do not seem to fit into the online convex optimization framework. For example, in online classification problems, the predictions domain or the loss functions are not convex. In this section we describe two convexification

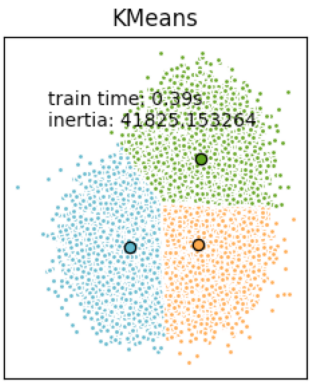

Fig. 2. Train Time and Inertia for K-Mean

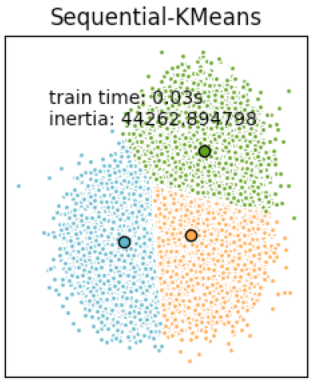

Fig. 3. Train Time and Inertia for Sequential K-Mean

techniques that allow us to utilize the online convex optimization framework in additional scenarios.

1) Convexification by Randomization: To demonstrate the randomization technique, consider the problem of prediction with expert advice, where on each online round the learner has to choose from the advice of d given experts. Denote by $p_t \in \{1,...,d\}$ the chosen expert. Then, the learner receives a vector $y_t \in [0,1]^d$, where $y_t[i]$ is the cost of following the advice of the i^{th} expert. The learner pays the loss $y_t[p_t]$. In this prediction problem, the decision space is discrete, hence nonconvex. However, by allowing the learner to randomize his predictions we can cast the problem in the online convex optimization framework, and therefore can obtain low regret algorithm for this problem. Formally, let $S = \{w \in R^d : w \geq 0, w1 = 1\}$ be the probability simplex, which forms a convex set. At round t, the learner chooses $w_t \in S$ and based on w_t picks an expert at random according to $P[p_t = i] = w_t[i]$. Then, the cost vector y_t is revealed and the learner pays for his expected cost:

$$\mathbb{E}[y_t[p_t]] = P[p_t = i]y_t[i] =< w_t, y_t >$$

Now we can cast the problem as online convex optimization since S is a convex set and the loss function, $f_t(w) =< w, y_t >$, happens to be a linear function (hence, convex).

2) Convexification by Surrogate Loss Functions: To explain the second convexification technique we again

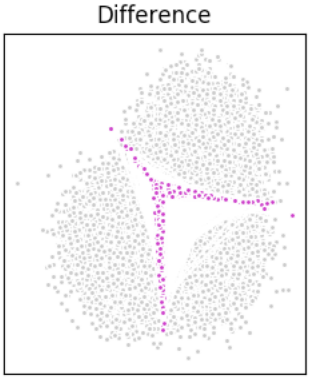

Fig. 4. Difference In clustering

start with the specific problem of online classification with a finite hypothesis class. As we are aware that 0-1 loss function for classification is non-convex and hence we surrogate or substitute this loss function by taking convex upper bound of this loss, which is hindge loss. For an intended output $t = 1$ and a classifier score y, the hinge loss of the prediction y is defined as :

$$\ell(y) = \max(0, 1 - t \cdot y)$$

It can be seen that when t and y have the same sign (meaning y predicts the right class) and $|y| \geq 1$, the hinge loss $\ell(y) = 0$, but when they have opposite sign, $\ell(y)$ increases linearly with y (one-sided error). Fig.5 and Fig.6 show how 0-1 loss is convex upper bounded by hindge loss.

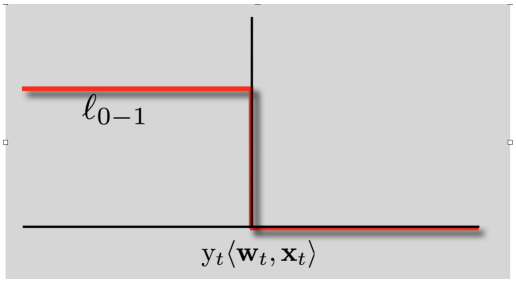

Fig. 5. 0-1 Loss

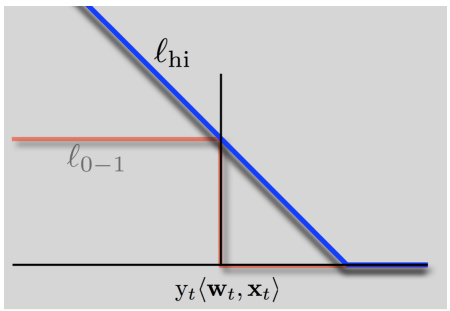

Fig. 6. Hindge Loss

VIII. CONCLUSIONS

Online learning is a well established learning paradigm which has both theoretical and practical appeals. The goal of online learning is to make a sequence of accurate predictions given knowledge of the correct answer to previous prediction tasks and possibly additional available information. Online learning has been studied in several research fields including game theory, information theory, and machine learning. It also became of great interest to practitioners due the recent emergence of large scale applications such as online advertisement placement and online web ranking. Online Learning Algorithms are useful in two scenarios. Firstly, When the application gets data sequentially. In this case, the Online Algorithms are natural mode for such application e.g stock price prediction. Secondly, when it is computationally inefficient to train a model on the entire dataset, in this case Online Algorithms comes to rescue e.g stochastic gradient descent.

REFERENCES

[1] A. Blum, On-line algorithms in machine learning, Online Algorithms,1998.

[2] N. Cesa-Bianchi, A. Conconi, and C. Gentile, On the generalization ability of on-line learning algorithms, IEEE Transactions on Information Theory, 2004.

[3] N. Cesa-Bianchi and G. Lugosi, Prediction, Learning, and Games. Cambridge University Press, 2006.

[4] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linearthreshold Algorithm. Machine Learning, 2(1):285-318, 1988.

[5] Lecture notes on Online Learning, Alexander Raklin, 2009.

[6] Online Learning and Online Convex Optimization By Shai Shalev-Shwartz.

[7] Wikipedia

[8] Scikit-Learn.

[9] princeton univ. F13 cos 521: Advanced Algorithm Design.

[10] A. Rakhlin, Lecture Notes on Online Learning. draft, 2009.

[11] S. Shalev-Shwartz, Online learning: Theory, algorithms, and applications,Ph.D. Thesis, The Hebrew University, 2007.

[12] V. N. Vapnik, Statistical Learning Theory. Wiley, 1998.

[13] N. Littlestone and M. K. Warmuth, The weighted majority algorithm, Information and Computation, 1994.

[14] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry. The MIT Press, 1969.

[15] A. Kalai and S. Vempala, Efficient algorithms for online decision problems,Journal of Computer and System Sciences,2005.