Perceptron Learning

Xiaogang Su, Ph.D.
Department of Mathematical Sciences
University of Texas at El Paso (UTEP)
xsu@utep.edu

Contents

We will study several special classifiers in the next few lectures. They may not be so widely used in applications nowadays, especially statistical applications, but they had their peak times, as they either represent a group of classification methods with some unique characteristics or they are building blocks for modern classifiers.

Perceptron learning algorithm played a crucial role in the history of machine learning. It supplies a typical example of online learning. At the same time, it is the root of many other modern learning tools such as boosting, artificial neural networks (ANN) models, and support vector machines (SVM).

1 Online Learning

The logistic regression models (typically estimated with Fisher scoring or the iteratively weighted least squares) belong to *batch learning*, where we are first given a training set to learn with, and

then apply the fitted models (or hypotheses in machine learning language). Comparatively, *online learning* is a model of induction that learns one instance (or observation) at a time. In online learning settings, the algorithm has to make predictions continuously and possibly improve its estimators even while it is still learning. The key defining characteristic of on-line learning is that soon after the prediction is made, the true label of the instance is discovered. This information can then be used to refine the prediction hypothesis used by the algorithm.

Given data $\{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$, the algorithm first sees $\mathbf{x}_1$ and is asked to predict what it thinks y_1 is. After making its prediction, the true value of y_1 is revealed to the algorithm (and the algorithm may use this information to perform some learning). The algorithm is then shown $\mathbf{x}_2$ and again asked to make a prediction, after which y_2 is revealed, and it may again perform some more learning. This proceeds until we reach $(\mathbf{x}_n; y_n)$.

2 Perceptron

We will give a bound on the online learning error of the perceptron algorithm. To make our subsequent derivations easier, we will use the notational convention of denoting the class labels by $\mathbf{y} \in \{-1, +1\}$. The perceptron model makes prediction according to a decision function $d(\mathbf{x})$ as follows

$$\hat{y} = d(\mathbf{x}) \;=\; \text{sgn}(\mathbf{x}^T\boldsymbol{\beta}) \;=\; \begin{cases} +1 & \text{if } \mathbf{x}^T\boldsymbol{\beta} \geq 0 \\ -1 & \text{if } \mathbf{x}^T\boldsymbol{\beta} < 0 \end{cases} \tag{1}$$

where the parameter $\boldsymbol{\beta} \in \mathbb{R}^p$ is often called the 'weight' (hence denoted as w and $\mathbf{w}$) in the ANN setting. For the sake of simplicity, we have slightly modified the definition of sgn so that $\text{sgn}(x) = +1$ if $x \geq 0$, instead of $x > 0$. We assume that the first component of $\mathbf{x}$ is $x_0 = 1$ to absorb the intercept (or bais) term β_0 so that $\mathbf{x}^T\boldsymbol{\beta} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$ is an affine transformation of original predictors. An ANN graphical representation of perceptron model is given in Figure 1. Logistic regression can be viewed as a natural extension of perceptron by replacing the hard-threshold function with a smooth sigmoid (i.e., the expit or logistic) function and applying a cutoff point $c = 0.5$ (instead of 0) to the predicted probabilities.

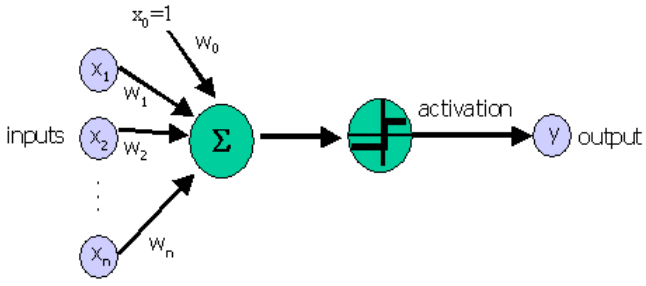

Figure 1: Illustration of Perceptron Algorithm as an Artificial Neural Network (ANN) Model.

2.1 The Perceptron Algorithm

To estimate $\boldsymbol{\beta}$, the 0-1 loss function

$$l(y, \hat{y}) = 1\{y\hat{y} < 0\} = 1\{y\mathbf{x}^T\boldsymbol{\beta} < 0\}$$

leads to the following empirical risk function

$$\hat{R}_{0/1}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} 1\{y_i(\mathbf{x}_i^T\boldsymbol{\beta}) < 0\}.$$

The empirical risk $\hat{R}_{0/1}(\boldsymbol{\beta})$ with 0/1 loss can be interpreted as the misclassification rate where the numerator simply counts the number of misclassified observations. However, $\hat{R}_{0/1}(\boldsymbol{\beta})$ does not produce useful gradient mainly because the 0-1 loss function is neither continuous nor convex. For convex relaxation, the continuous hinge loss $l(y, \hat{y}) = \max(0, -y\hat{y})$ is considered instead. See Figure 2 for plots of the 0-1 and hinge losses as functions of $y\hat{y}$.

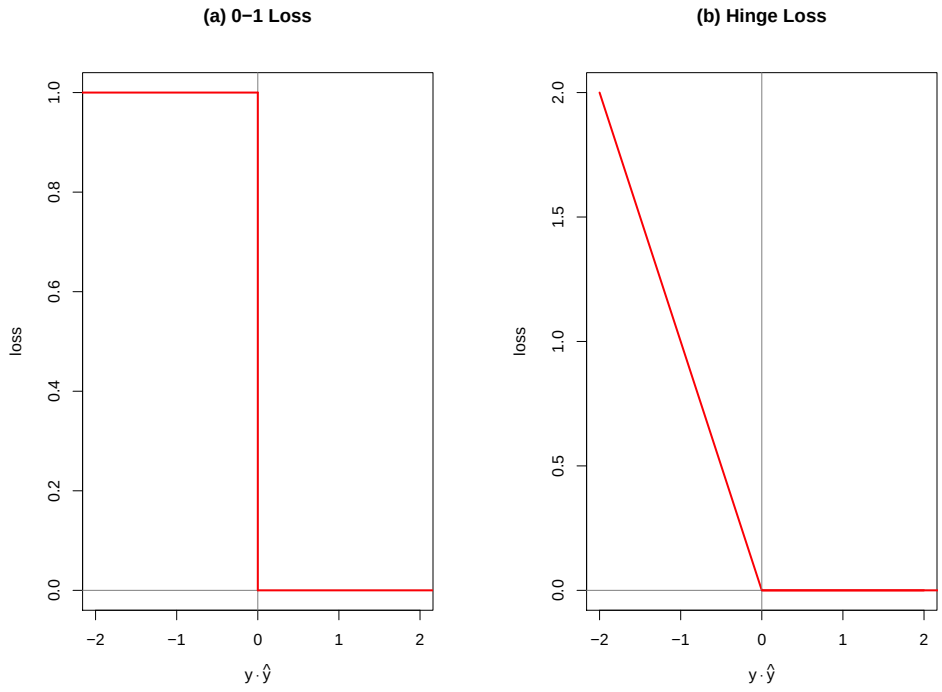

Figure 2: The 0-1 loss (a) and hinge loss (b) as a function of $y\,\hat{y}$.

The hinge loss leads to an empirical risk given below

$$\hat{R}(\boldsymbol{\beta}) \;=\; \frac{L(\boldsymbol{\beta})}{n} = \frac{1}{n}\sum_{i=1}^{n} L_i \;=\; \frac{1}{n}\sum_{i=1}^{n} \max(0, -y_i\mathbf{x}_i^T\boldsymbol{\beta}). \tag{2}$$

For interpretability, rewrite

$$\hat{R}(\boldsymbol{\beta}) \;=\; -\frac{1}{n}\sum_{i\in\mathcal{M}} y_i\mathbf{x}_i^T\boldsymbol{\beta} \;\propto\; -\sum_{i\in\mathcal{M}} y_i(\mathbf{x}_i^T\boldsymbol{\beta}),$$

where $\mathcal{M}$ is the set of misclassified observations. Rewriting that $\mathbf{x}_i^T\boldsymbol{\beta} = \beta_0 + \boldsymbol{\beta}_1^T\mathbf{x}_{i(1)}$ with normalized $\boldsymbol{\beta}_1$ (i.e., $\|\boldsymbol{\beta}_1\| = 1$), $\hat{R}(\boldsymbol{\beta})$ can be interpreted as the total distance from all misclassified points to

the super plane $\beta_0 + \mathbf{x}_{i(1)}^T \boldsymbol{\beta}_1 = 0$ in the predictor space $\mathcal{X}$ where $\mathbf{x}_{(1)}$ lives. By minimizing $\hat{R}(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, we seek a classification superplane $\mathbf{x}^T \boldsymbol{\beta}^T = 0$ that has the shortest distance to all the points that it fails to classify correctly.

The derivative of the hinge loss (2) is given as

$$g_i = \frac{dL_i}{d\boldsymbol{\beta}} = \frac{d \max(0, -y_i x_i^T \boldsymbol{\beta})}{d\,\boldsymbol{\beta}} = \begin{cases} 0 & \text{if } y_i \mathbf{x}_i^T \boldsymbol{\beta} > 0 \\ -y_i \mathbf{x}_i & \text{otherwise} \end{cases} = -(y_i - \hat{y}_i)\mathbf{x}_i/2 \ ,$$

recalling that $y_i, \hat{y}_i \in \{\pm 1\}$. The concepts of sub-derivative and sub-gradient are helpful here to deal with nondifferentiability at 0 of the hinge loss, but it really does not matter.

It follows that the gradient of $nL(\boldsymbol{\beta})$ is given as

$$\mathbf{g} \ = \ \frac{d\,L}{d\,\boldsymbol{\beta}} = -\sum_{i=1}^{n} (y_i - \hat{y}_i)\mathbf{x}_i/2,$$

where again $\hat{y}_i = \text{sgn}(\mathbf{x}_i^T \boldsymbol{\beta})$ and hence $(y_i - \hat{y}_i)/2 = y_i$ if $y_i \neq \hat{y}_i$ (missclassified) and 0 otherwise. A gradient descent algorithm would involve calculation of the sum of n items; this type of *batch learning* is computationally costly for large n.

Perceptron algorithm takes the stochastic gradient descent (SGD) approach by replacing $\mathbf{g}$ with $\mathbf{g}_i$. SGD essentially turns batch learning into on-line incremental learning, where the observations in massive data are processed one (or several) at a time.

The perceptron algorithm is an online weight-update algorithm, which is presented in Algorithm 1. After observing y_i, if it is a correct prediction $\hat{y} = y$, then it makes no change to the parameters; otherwise (i.e., $y_i \mathbf{x}_i^T \boldsymbol{\beta} < 0$), it performs an update $\boldsymbol{\beta} := \boldsymbol{\beta} - \gamma \cdot (-y_i \mathbf{x}_i) = \boldsymbol{\beta} + \gamma\, y_i \mathbf{x}_i)$, where γ $(0 < \gamma \leq 1)$ is the learning rate. The learning rate γ is applied to avoid overshooting. In optimization, the learning rate is termed as the *step size* instead. Its value can be different at each iteration; searching for optimal $\gamma > 0$ involves another one-dimensional optimization procedure called '*line search*'. However, many machine learning algorithms fix the learning rate at a constant to gain computational efficiency.

Algorithm 1 Perceptron Algorithm

```
initialize β = 0.
for i = 1, 2, ..., n do
  predict ŷ_i = sign(x_i^T β);
  if ŷ_i ≠ y_i then
    update β := β + y_i x_i;
  end if
end for
```

An intuitive interpretation of perceptron algorithm is that, only when a misclassification error occurs, we adjust the coefficients $\boldsymbol{\beta}$ (weight and bias) so that the resultant super plane moves in the direction that reduces its distance from the misclassified point.

2.2 Theoretical Properties

Since an online learning algorithm makes predictions continuously while learning, one is interested in the total number of errors made by the algorithm during the process. The following theorem

gives a bound on the online learning error of the perceptron algorithm, when it is run as an online algorithm that performs an update each time it gets an example wrong. Note that the bound below on the number of errors does not have an explicit dependence on the sample size n in the sequence, or on the dimension p of the inputs! The scenario considered here is the linearly separable setting, meaning that $+1$'s and -1's can be *perfectly* separated by a (linear) superplane of $\mathbf{x}$.

Theorem 2.1. *(Block, 1962 and Novikoff, 1962). Let a sequence of observations* $\{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ *be given. Suppose that, for all* i*,*

(i) $\| \mathbf{x}_i \| \leq M$*;*

(ii) There exists a unit-length vector $\boldsymbol{\beta}_\star$ *(i.e.,* $\|\boldsymbol{\beta}_\star\| = 1$*) such that* $y_i \cdot (\mathbf{x}_i^T \boldsymbol{\beta}_\star) \geq \gamma$ *for some* $\gamma > 0$*.*

Then the total number of mistakes that the perceptron algorithm makes on this sequence is at most $(M/\gamma)^2$*.*

Proof. Let $\boldsymbol{\beta}_k$ be the weights being used when the perceptron algorithm has made its k-th mistake. Suppose this occurs to the i-th observation $(\mathbf{x}_i, y_i)$. Then

$$(\mathbf{x}_i^T \boldsymbol{\beta}_k) \cdot y_i \ \leq \ 0, \tag{3}$$

i.e., the prediction $\hat{y}_i$ and y_i have opposite signs.

According to the learning rule, $\boldsymbol{\beta}_{k+1} := \boldsymbol{\beta}_k + y_i \mathbf{x}_i$. We have

$$\begin{aligned} \boldsymbol{\beta}_{k+1}^T \boldsymbol{\beta}_\star &= \boldsymbol{\beta}_k^T \boldsymbol{\beta}_\star + y_i \mathbf{x}_i^T \boldsymbol{\beta}_\star \\ &\geq \boldsymbol{\beta}_k^T \boldsymbol{\beta}_\star + \gamma, \quad \text{by assumption (ii)} \\ &\geq \boldsymbol{\beta}_{k-1}^T \boldsymbol{\beta}_\star + \gamma + \gamma, \ \text{by induction} \\ &= k\gamma, \quad \text{noting that } \boldsymbol{\beta}_1 = \mathbf{0} \text{ by initialization.} \end{aligned}$$

Also, we have

$$\begin{aligned} \| \boldsymbol{\beta}_{k+1} \|^2 &= \| \boldsymbol{\beta}_k + y_i \mathbf{x}_i \|^2 \\ &= \| \boldsymbol{\beta}_k \|^2 + \| \mathbf{x}_i \|^2 + 2 y_i (\mathbf{x}_i^T \boldsymbol{\beta}_k) \\ &\leq \| \boldsymbol{\beta}_k \|^2 + \| \mathbf{x}_i \|^2, \quad \text{since } y_i (\mathbf{x}_i^T \boldsymbol{\beta}_k) < 0 \\ &\leq \| \boldsymbol{\beta}_k \|^2 + M^2 \\ &\leq \| \boldsymbol{\beta}_{k-1} \|^2 + M^2 + M^2, \quad \text{by induction} \\ &= k \cdot M^2. \end{aligned}$$

Putting together, we have

$$\sqrt{k}\, M \geq \| \boldsymbol{\beta}_{k+1} \|^2 \geq \boldsymbol{\beta}_{k+1}^T \boldsymbol{\beta}_\star \geq k\, \gamma,$$

by Cauchy-Schwartz inequality since $\| \boldsymbol{\beta}_\star \| = 1$. Therefore, $k \leq (M/\gamma)^2$. □

Condition (ii) is equivalent to saying that

$$\mathbf{x}_i^T \boldsymbol{\beta}_\star \geq \gamma \ \text{ if } y_i = +1$$

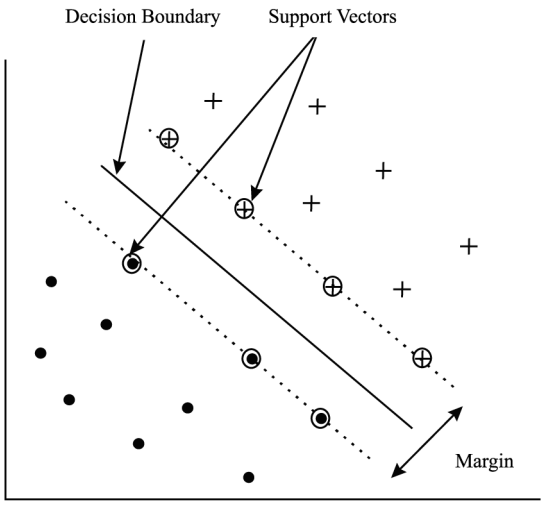

Figure 3: Illustration of Classification Margin.

and

$$\mathbf{x}_i^T \boldsymbol{\beta}_\star \leq -\gamma \;\text{ if } y_i = -1.$$

The parameter γ is referred to as the *margin*, which is the minimum distance from data points to the decision boundary. Namely, the (true) direction $\boldsymbol{\beta}_\star$ separates the $+1$'s from -1's with a margin of at least γ. The bigger the margin, the easier the classification problem is and the more confident we are about our prediction. This concept leads to one of the most exciting developments in the ML field - support vector machines (SVM). An two-dimensional illustration is provided in Figure 3.

It also has been established that, if the training data have an exact solution that is linearly separable, then perceptron is guaranteed to find an exact solution in a finite number of steps. This result is referred to the perceptron convergence theorem. Even if it converges, the perceptron solution is not unique in general unless further constraints are added. If the data are not linearly separable, then perceptron may never converge. It is worth noting that perceptron does not generate output such as predicted event probabilities; nevertheless, the estimated linear combinations $\mathbf{x}_i^T \hat{\boldsymbol{\beta}}$ may play the similar role for various purposes, e.g., making ROC curve.

Perceptron algorithm has a dual form from the optimization perspective. The dual approach utilizes the fact that $\widehat{\boldsymbol{\beta}}$ are linear combinations of $y_i \mathbf{x}_i$ and hence can be solved by updating the coefficient of $y_i \mathbf{x}_i$. In this dual algorithm, the original data only appears in terms of inner product $\mathbf{x}_i^T \mathbf{x}_{i'}$ (or Gram matrix $\mathbf{X}\mathbf{X}^T$), which makes an immediate connection to the 'kernel' trick. On this basis, a nonlinear version of perceptron is conveniently developed.

Perceptron can be extended to multiclass classification problems as follows. Let the support of Y be $\mathcal{Y} = \{1, \ldots, K\}$. Given an observation with $\mathbf{x}$, perceptron fits a superplane $\mathbf{x}^T \boldsymbol{\beta}_k$ for each category $k \in \mathcal{Y}$. Its decision function $d(\mathbf{x})$ is then defined as

$$\hat{y} = d(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\; \mathbf{x}^T \boldsymbol{\beta}_k.$$

The online algorithm proceeds as in Algorithm 2. Namely, $\boldsymbol{\beta}_k$'s will be updated only when a misclassification occurs. The update step adds $\mathbf{x}_i$ to the $\boldsymbol{\beta}_k$ for the same category k as y_i and

subtract $\mathbf{x}_i$ from all other $\boldsymbol{\beta}_k$. In general, extending binary classifier to multiclass classification has several principled ways, some of which may suffer from ambiguous decision regions. We shall explore this later.

Algorithm 2 Multiclass Perceptron Algorithm

```
initialize β_k = 0 for k = 1, ..., K.
for i = 1, 2, ..., n do
  predict ŷ_i = argmax_k(x_i^T β_k);
  if ŷ_i ≠ y_i then
    for k = 1, 2, ..., K do
      if k = y_i then
        update β_k := β_k + x_i;
      end if
      if k ≠ y_i then
        update β_k := β_k − x_i;
      end if
    end for
  end if
end for
```

3 The Committee Learning Problem

In committee learning, we have p experts. Each experts predicts $+1$ or -1 in each round. Assume that there is a panel of k experts (out of p members); when we take the majority vote of these k experts, we always get the right prediction. To avoid ambiguity, we assume k is an odd number. Same as the concept of linear separable classification, this setting presents a classification problem without error in the truth, which involves a perfect subcommittee that makes exact classification altogether. This may not be realistic in reality. For the applicability of committee learning, one may relate it to ensemble learning where each committee represents a base learner. This connection motivates the boosting method.

3.1 Application of Perceptron

Now we would like to use perceptron algorithm to find out this k subcommittee for the learning problem. We are interested in finding out how many mistakes perceptron algorithm will make along the way.

The resultant data consist of $\{\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \{+1, -1\}^p$ is the votes of p experts and $y_i \in \{+1, -1\}$ is the observed response. Define a (selection) vector as $\boldsymbol{\beta} \in \mathbb{R}^p$, which has k 1's and $(p-k)$ 0's as its elements. For example, $\boldsymbol{\beta} = (0, 1, 1, 0, \ldots, 0)^T$. Here $\boldsymbol{\beta}$, corresponding to $\boldsymbol{\beta}$ in perceptron, can be viewed the (true) selection operator that helps identify the k experts. When we take the majority vote of those k experts that correspond to positions of 1's in $\boldsymbol{\beta}$, the prediction is always right, i.e., $y_i = \text{sign}(\mathbf{x}_i^T \boldsymbol{\beta})$.

To determine the bound of perceptron algorithm, we first normalize $\mathbf{x}_i$ and $\boldsymbol{\beta}$ to have unit

lengths. Noting that $\| x_i \| = \sqrt{p}$ and $\| \boldsymbol{\beta} \| = \sqrt{k}$. Thus set

$$\mathbf{x}_i \;:=\; \mathbf{x}_i/\sqrt{p} \quad \text{and} \quad \boldsymbol{\beta} \;:=\; \boldsymbol{\beta}/\sqrt{k}.$$

After normalization, $\| \mathbf{x}_i \| = 1 = M$, where the definition of M is given in Theorem 2.1. And it can be seen that $y_i(\mathbf{x}_i^T \boldsymbol{\beta})$ is always some integer times $1/\sqrt{p\,k}$. Therefore,

$$y_i(\mathbf{x}_i^T \boldsymbol{\beta}) \geq 1/\sqrt{p\,k}.$$

That is, the margin γ is $1/\sqrt{p\,k}$. According to Theorem 2.1, the number of mistakes is bounded by $\left(\frac{M}{1/\sqrt{p\,k}}\right)^2 = p\,k$ (since again $M = 1$). which is the product of the number of experts and the size of the panel of correct experts. What we don't like about this bound is that it's linearly increasing with the total number of experts p, which means we can't afford to have too many experts. In practice, we often have a lot of experts, or a lot of features/attributes to select from. For this committee learning problem, another method, Winnow Algorithm, is advantageous.

3.2 Winnow Algorithm

Winnow algorithm is very similar to perceptron algorithm, but makes fewer mistakes especially when the number of experts, p, increases. The main idea of Winnow algorithm is to weigh those experts who were right more than those who were wrong. See Algorithm 3 below for details.

Algorithm 3 Winnow Algorithm

initialize $\boldsymbol{\beta} = \mathbf{1}$ and set $\eta > 0$.
for $i = 1, 2, ..., n$ **do**
  predict $\hat{y}_i = \text{sign}(\mathbf{x}_i^T \boldsymbol{\beta})$;
  if $\hat{y}_i \neq y_i$ **then**
    update $\boldsymbol{\beta} := \text{diag}\left\{\exp\left(\eta \cdot y_i\, \mathbf{x}_i\right)\right\} \boldsymbol{\beta}$;
  end if
end for

Similar to perceptron, Winnow algorithm only updates estimates when a mistake is made in prediction, i.e., $y_i\, \mathbf{x}_i^T \boldsymbol{\beta} < 0$ or y_i and $\mathbf{x}_i^T \boldsymbol{\beta}$ with opposite signs. The update step

$$\boldsymbol{\beta} := \text{diag}\left\{\exp\left(\eta \cdot y_i\, \mathbf{x}_i\right)\right\} \boldsymbol{\beta}$$

is equivalent to

$$\beta_j := \exp\left(\eta \cdot y_i\, x_{ij}\right) \beta_j$$

for $j = 1, \ldots, p$. Since $\eta > 0$, the method multiplies the current weights (β_j) of those experts who were right $(y_i x_{ij} = 1 > 0)$ by $\exp(\eta) > 1$ and multiplies the current weights (β_j) of those experts who were wrong $(y_i x_{ij} = -1 < 0)$ by $\exp(-\eta) < 1$. In other words, the weights of experts who are right will be increased while the weights of experts who are wrong will be decreased. Winnon algorithm estimates $\boldsymbol{\beta}$ with a multiplicative update while perceptron update $\boldsymbol{\beta}$ additively. For this reason, $\boldsymbol{\beta}$ can not be initialized to $\mathbf{0}$ in Algorithm 3. It can be shown that, under similar conditions, for this committee learning problem, Winnow algorithm gives a bound that is more tolerant of the total number of experts p. What makes perceptron and Winnow algorithms important is that perceptron motivated SVM while Winnow algorithm motivated boosting.

References

Block, H. D. (1962). The perceptron: A model for brain functioning. *Reviews of Modern Physics*, **34**(1): 123–135.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, **37**(3): 277–296.

Gallant, S. I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, **1**(2): 179–191.

Minsky, M. L. and Papert, S. A. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Novikoff, A. B. (1962). On convergence proofs on perceptrons. *Symposium on the Mathematical Theory of Automata*, **12**: 615–622.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory. *Psychological Review*, **65**(6): 386–408.