CS/DSC/AI 391L: Machine Learning

Homework 2 - Theory

Patrick Brown

Lecture: Prof. Adam Klivans

Keywords: Perceptron, SGD, Boosting

Instructions: Please either typeset your answers (LaTeX recommended) or write them very clearly and legibly and scan them, and upload the PDF on edX. Legibility and clarity are critical for fair grading.

Problem 1

No, the Perceptron algorithm does not necessarily make the same number of mistakes nor end up with the same final weights when the training set is presented in a different order.

Counterexample:
Consider a training set S with three examples in $\mathbb{R}^2$:

$$\begin{aligned}
&\text{Example 1:} \quad \mathbf{x}_1 = (1,0), && y_1 = +1\\
&\text{Example 2:} \quad \mathbf{x}_2 = (0,1), && y_2 = +1\\
&\text{Example 3:} \quad \mathbf{x}_3 = (-1,-1), && y_3 = -1
\end{aligned}$$

Let the initial weight vector be $\mathbf{w}_0 = (0,0)$.
Now, consider two different orders of the same training set:

- Order S: $[(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3)]$
- Order S': $[(\mathbf{x}_3, y_3), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2)]$

Perceptron Algorithm on Order S:

1. **First example** $(\mathbf{x}_1, y_1)$:
$$\mathbf{w}_0^\top \mathbf{x}_1 = 0, \quad \text{prediction} = 0 \neq y_1 = +1$$
Update: $\mathbf{w}_1 = \mathbf{w}_0 + y_1\mathbf{x}_1 = (1,0)$
2. **Second example** $(\mathbf{x}_2, y_2)$:
$$\mathbf{w}_1^\top \mathbf{x}_2 = 0, \quad \text{prediction} = 0 \neq y_2 = +1$$
Update: $\mathbf{w}_2 = \mathbf{w}_1 + y_2\mathbf{x}_2 = (1,1)$
3. **Third example** $(\mathbf{x}_3, y_3)$:
$$\mathbf{w}_2^\top \mathbf{x}_3 = -2, \quad \text{prediction} = -1 = y_3 = -1$$
No update needed.

Total mistakes on S: 2

Perceptron Algorithm on Order S':

1. **First example** $(\mathbf{x}_3, y_3)$:
$$\mathbf{w}_0^\top \mathbf{x}_3 = 0, \quad \text{prediction} = 0 \neq y_3 = -1$$
Update: $\mathbf{w}_1 = \mathbf{w}_0 + y_3\mathbf{x}_3 = (-1,-1)$

2. **Second example** $(\mathbf{x}_1, y_1)$:

$$\mathbf{w}_1^\top \mathbf{x}_1 = -1, \quad \text{prediction} = -1 \neq y_1 = +1$$

Update: $\mathbf{w}_2 = \mathbf{w}_1 + y_1\mathbf{x}_1 = (0, -1)$

3. **Third example** $(\mathbf{x}_2, y_2)$:

$$\mathbf{w}_2^\top \mathbf{x}_2 = -1, \quad \text{prediction} = -1 \neq y_2 = +1$$

Update: $\mathbf{w}_3 = \mathbf{w}_2 + y_2\mathbf{x}_2 = (0, 0)$

Total mistakes on S': 3

Conclusion:
The Perceptron algorithm makes a different number of mistakes depending on the order of the training examples:

- On order S: 2 mistakes.
- On order S': 3 mistakes.

Although the final weights happen to be the same in this example ($\mathbf{w} = (0, 0)$), in general, the final weights can also differ with the order of the training set.

Therefore, the order of the training examples can affect both the number of mistakes made and the final weights obtained by the Perceptron algorithm.

Problem 2

We have mainly focused on squared loss, but there are other interesting losses in machine learning. Consider the following loss function which we denote by $\phi(z) = \max(0, -z)$. Let S be a training set $(x^1, y^1), \ldots, (x^m, y^m)$ where each $x^i \in \mathbb{R}^n$ and $y^i \in \{-1, 1\}$. Consider running stochastic gradient descent (SGD) to find a weight vector w that minimizes $\frac{1}{m}\sum_{i=1}^m \phi(y^i \cdot w^\top x^i)$. Explain the explicit relationship between this algorithm and the Perceptron algorithm. Recall that for SGD, the update rule when the i-th example is picked at random is

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla \phi\left(y^i w^\top x^i\right).$$

Note: You do not need to be overly concerned about the discontinuity at $\phi(0)$, so you can ignore this when calculating the gradient for this problem.

Solution

We are to explain the explicit relationship between the stochastic gradient descent (SGD) algorithm minimizing the loss function

$$\phi(z) = \max(0, -z)$$

and the Perceptron algorithm.

Loss Function

First, consider the loss function $\phi(z)$:

$$\phi(z) = \begin{cases} 0, & \text{if } z \geq 0, \\ -z, & \text{if } z < 0. \end{cases}$$

This function penalizes misclassified examples (where $z = y^i w^\top x^i < 0$) by an amount proportional to $-z$, and does not penalize correctly classified examples (where $z \geq 0$).

Gradient Computation

The gradient of $\phi(z)$ with respect to w is:

$$\nabla_w \phi(y^i w^\top x^i) = \phi'(y^i w^\top x^i) \cdot y^i x^i.$$

The derivative $\phi'(z)$ is:

$$\phi'(z) = \begin{cases} -1, & \text{if } z < 0, \\ 0, & \text{if } z > 0. \end{cases}$$

(Note: At $z = 0$, $\phi'(z)$ is undefined, but we can ignore this discontinuity as per the problem note.)

SGD Update Rule

The SGD update rule when the i-th example is picked is:

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla_w \phi(y^i w^\top x^i).$$

Substituting the gradient, we get:

- If $y^i w^\top x^i < 0$ (misclassified example):

$$\nabla_w \phi(y^i w^\top x^i) = (-1) \cdot y^i x^i = -y^i x^i,$$

$$w_{\text{new}} = w_{\text{old}} - \eta(-y^i x^i) = w_{\text{old}} + \eta y^i x^i.$$

- If $y^i w^\top x^i > 0$ (correctly classified example):

$$\nabla_w \phi(y^i w^\top x^i) = 0,$$

$$w_{\text{new}} = w_{\text{old}}.$$

Perceptron Algorithm

The Perceptron algorithm updates the weight vector as follows:

- For each example (x^i, y^i):
  - If $y^i w^\top x^i \leq 0$ (misclassified or on the decision boundary):

$$w_{\text{new}} = w_{\text{old}} + y^i x^i.$$

  - If $y^i w^\top x^i > 0$ (correctly classified):

$$w_{\text{new}} = w_{\text{old}}.$$

Conclusion

The SGD algorithm with the loss function $\phi(z) = \max(0, -z)$ is essentially a scaled version of the Perceptron algorithm with a slight difference in the update condition:

- **Similarity**: Both algorithms update the weight vector by adding $y^i x^i$ when an example is misclassified.
- **Difference in Update Condition**: The SGD algorithm does not update when $y^i w^\top x^i = 0$, whereas the Perceptron algorithm does.
- **Learning Rate Flexibility**: The SGD algorithm allows for a customizable learning rate η, providing more flexibility in optimization compared to the fixed $\eta = 1$ in the standard Perceptron algorithm.

Explicit Relationship

Comparing the two algorithms:

1. **Update Condition**:
   - **SGD with $\phi(z)$**: Updates only when $y^i w^\top x^i < 0$ (strictly misclassified).
   - **Perceptron Algorithm**: Updates when $y^i w^\top x^i \leq 0$ (misclassified or on the boundary).
2. **Update Rule**:
   - **SGD**: $w_{\text{new}} = w_{\text{old}} + \eta y^i x^i$
   - **Perceptron**: $w_{\text{new}} = w_{\text{old}} + y^i x^i$ (implicitly with $\eta = 1$).
3. **Learning Rate (η)**:
   - **SGD**: Uses a learning rate η, which can be any positive value.
   - **Perceptron**: Typically uses $\eta = 1$.

The given SGD algorithm with the loss function $\phi(z) = \max(0, -z)$ corresponds to the Perceptron algorithm with a learning rate η, updating the weight vector only on strictly misclassified examples ($y^i w^\top x^i < 0$). The update rule is identical in form, scaling the adjustment by η. Thus, the SGD algorithm can be viewed as a generalization of the Perceptron algorithm with adjustable learning rate and a slightly stricter update condition.

Problem 3

Here we will give an illustrative example of a weak learner for a simple concept class. Let the domain be the real line, $\mathbb{R}$, and let C refer to the concept class of "3-piece classifiers", which are functions of the following form: for $\theta_1 < \theta_2$ and $b \in \{-1, 1\}$, $h_{\theta_1,\theta_2,b}(x)$ is b if $x \in [\theta_1, \theta_2]$ and $-b$ otherwise. In other words, they take a certain Boolean value inside a certain interval and the opposite value everywhere else. For example, $h_{10,20,1}(x)$ would be +1 on [10, 20], and -1 everywhere else. Let $\mathcal{H}$ refer to the simpler class of "decision stumps", i.e. functions $h_{\theta,b}$ such that $h(x)$ is b for all $x \leq \theta$ and $-b$ otherwise.

(a) Show formally that for any distribution on $\mathbb{R}$ (assume finite support, for simplicity; i.e., assume the distribution is bounded within $[-B, B]$ for some large B) and any unknown labeling function $c \in C$ that is a 3-piece classifier, there exists a decision stump $h \in \mathcal{H}$ that has error at most 1/3, i.e. $\mathbb{P}[h(x) \neq c(x)] \leq 1/3$.

(b) Describe a simple, efficient procedure for finding a decision stump that minimizes error with respect to a finite training set of size m. Such a procedure is called an empirical risk minimizer (ERM).

(c) Give a short intuitive explanation for why we should expect that we can easily pick m sufficiently large that the training error is a good approximation of the true error, i.e. why we can ensure generalization. (Your answer should relate to what we have gained in going from requiring a learner for C to requiring a learner for $\mathcal{H}$.) This lets us conclude that we can weakly learn C using $\mathcal{H}$.

Solution

(a) Proof that a Decision Stump Exists with Error at Most $\frac{1}{3}$:

Let's consider any distribution D over $\mathbb{R}$ (bounded within $[-B, B]$) and any concept $c \in C$, where c is a 3-piece classifier defined by parameters $\theta_1 < \theta_2$ and $b \in \{-1, 1\}$. The function $c(x)$ takes the value b on $[\theta_1, \theta_2]$ and $-b$ elsewhere.

Define the following probabilities based on D:

$$P_1 = \mathbb{P}[x < \theta_1] \quad \text{(probability mass on } (-\infty, \theta_1))$$

$$P_2 = \mathbb{P}[x \in [\theta_1, \theta_2]] \quad \text{(probability mass on } [\theta_1, \theta_2])$$

$$P_3 = \mathbb{P}[x > \theta_2] \quad \text{(probability mass on } (\theta_2, \infty))$$

Since the total probability is 1:

$$P_1 + P_2 + P_3 = 1$$

By the Pigeonhole Principle, at least one of P_1, P_2, or P_3 must be at most $\frac{1}{3}$. Without loss of generality, assume $P_1 \leq \frac{1}{3}$. Then, the decision stump $h_{\theta_1,-b}$ will have an error at most $\frac{1}{3}$.

(b) Procedure for Finding a Decision Stump (ERM):

1. **Sort the Training Set:**
   - Sort the training examples (x_i, y_i) by the feature values x_i.
   - There are at most $m+1$ thresholds.
2. **Evaluate All Possible Stumps:**
   - For each threshold θ and each $b \in \{-1, 1\}$:
     - Define the decision stump $h_{\theta,b}$ as:
       $$h_{\theta,b}(x) = \begin{cases} b, & \text{if } x \le \theta \\ -b, & \text{otherwise} \end{cases}$$
     - Compute the empirical error:
       $$\text{Error}(h_{\theta,b}) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\{h_{\theta,b}(x_i) \neq y_i\}$$
     - Keep track of the stump with the minimal error.
3. **Select the Best Stump:**
   - Choose the $h_{\theta,b}$ with the lowest empirical error.
   - Return this stump as the ERM hypothesis.

Efficiency Analysis:

- The total time complexity is $O(m \log m)$ due to the initial sorting.
- Evaluating all stumps is linear in m.

(c) Intuitive Explanation for Generalization Ability:

The key reason we can expect the training error to approximate the true error with a sufficiently large m is due to the low complexity of the hypothesis class $\mathcal{H}$:

- **Low VC-Dimension:** The class of decision stumps $\mathcal{H}$ has a VC-dimension of 2, which means it cannot shatter more than 2 points. This low capacity reduces the risk of overfitting.
- **Uniform Convergence:** For hypothesis classes with finite VC-dimension, statistical learning theory tells us that the empirical risk converges uniformly to the true risk as the sample size increases. Therefore, the training error becomes a good estimate of the true error.
- **Simplification from C to $\mathcal{H}$:** By shifting from learning the more complex class C (3-piece classifiers) to learning $\mathcal{H}$ (decision stumps), we have reduced the complexity of the model. This reduction makes it easier to generalize from the training data to unseen data.
- **Generalization Guarantee:** With the simpler class $\mathcal{H}$, we need fewer samples to achieve the same level of generalization compared to learning C. As a result, the empirical risk minimizer over $\mathcal{H}$ will generalize well to the true distribution with a reasonable sample size.

Conclusion:
By using decision stumps ($\mathcal{H}$) to weakly learn the concept class C, we leverage the low complexity of $\mathcal{H}$ to ensure that the training error closely approximates the true error. This approach allows us to guarantee generalization from the training set to the overall distribution, effectively weakly learning C using $\mathcal{H}$.

Problem 4

Consider an iteration of the AdaBoost algorithm (using notation from the video lecture on Boosting) where we have obtained classifier h_t. Show that with respect to the distribution D_{t+1} generated for the next iteration, h_t has accuracy exactly 1/2.

Solution

To show that h_t has an accuracy of exactly $\frac{1}{2}$ with respect to the new distribution D_{t+1}, we will:

1. Express the updated weights according to the AdaBoost algorithm.
2. Calculate the normalization factor Z_t.
3. Compute the weighted accuracy of h_t under D_{t+1}.

Step 1: Updating Weights

In AdaBoost, after obtaining the weak classifier h_t, we update the weights w_i of each training example (x_i, y_i) as follows:

$$w_i^{\text{new}} = w_i^{\text{old}} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{if } h_t(x_i) \neq y_i \end{cases}$$

where $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$ and ε_t is the weighted error rate of h_t under D_t:

$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} w_i^{\text{old}}$$

Step 2: Calculating the Normalization Factor Z_t

The normalization factor Z_t ensures that the updated weights sum to 1:

$$Z_t = \sum_{i=1}^{m} w_i^{\text{new}} = \beta_t \sum_{i:h_t(x_i)=y_i} w_i^{\text{old}} + \sum_{i:h_t(x_i)\neq y_i} w_i^{\text{old}}$$

We can express the sums over correctly and incorrectly classified examples:

- Total weight of correctly classified examples under D_t:

$$W_{\text{correct}} = \sum_{i:h_t(x_i)=y_i} w_i^{\text{old}} = 1 - \varepsilon_t$$

- Total weight of incorrectly classified examples under D_t:

$$W_{\text{incorrect}} = \sum_{i:h_t(x_i)\neq y_i} w_i^{\text{old}} = \varepsilon_t$$

Substitute back into Z_t:

$$Z_t = \beta_t(1-\varepsilon_t) + \varepsilon_t$$

Recall that $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$:

$$\begin{aligned} Z_t &= \left(\frac{\varepsilon_t}{1-\varepsilon_t}\right)(1-\varepsilon_t) + \varepsilon_t \\ &= \varepsilon_t + \varepsilon_t \\ &= 2\varepsilon_t \end{aligned}$$

Step 3: Computing the Weighted Accuracy Under D_{t+1}

Now, we compute the probability that h_t correctly classifies an example under D_{t+1}:

$$\text{Accuracy under } D_{t+1} = \sum_{i:h_t(x_i)=y_i} D_{t+1}(i)$$

First, express $D_{t+1}(i)$:

$$D_{t+1}(i) = \frac{w_i^{\text{new}}}{Z_t}$$

Compute the sum over correctly classified examples:

$$\begin{aligned}
\sum_{i:h_t(x_i)=y_i} D_{t+1}(i) &= \sum_{i:h_t(x_i)=y_i} \frac{w_i^{\text{new}}}{Z_t} \\
&= \frac{1}{Z_t} \sum_{i:h_t(x_i)=y_i} w_i^{\text{old}} \times \beta_t \\
&= \frac{\beta_t W_{\text{correct}}}{Z_t} \\
&= \frac{\beta_t(1-\varepsilon_t)}{2\varepsilon_t}
\end{aligned}$$

Substitute $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$:

$$\begin{aligned}
\sum_{i:h_t(x_i)=y_i} D_{t+1}(i) &= \frac{\left(\frac{\varepsilon_t}{1-\varepsilon_t}\right)(1-\varepsilon_t)}{2\varepsilon_t} \\
&= \frac{\varepsilon_t}{2\varepsilon_t} \\
&= \frac{1}{2}
\end{aligned}$$

Similarly, compute the sum over incorrectly classified examples:

$$\begin{aligned}
\sum_{i:h_t(x_i)\neq y_i} D_{t+1}(i) &= \sum_{i:h_t(x_i)\neq y_i} \frac{w_i^{\text{new}}}{Z_t} \\
&= \frac{1}{Z_t} \sum_{i:h_t(x_i)\neq y_i} w_i^{\text{old}} \times 1 \\
&= \frac{W_{\text{incorrect}}}{Z_t} \\
&= \frac{\varepsilon_t}{2\varepsilon_t} \\
&= \frac{1}{2}
\end{aligned}$$

Conclusion

Under the new distribution D_{t+1}, the probability that h_t correctly classifies an example is $\frac{1}{2}$, and the probability that it incorrectly classifies an example is also $\frac{1}{2}$. Therefore, with respect to D_{t+1}:

$$\text{Accuracy of } h_t = \frac{1}{2}$$

This demonstrates that h_t has no better than random guessing accuracy under D_{t+1}, which aligns with the AdaBoost algorithm's goal of focusing on harder examples in subsequent iterations.

Key Takeaways:

- The weight update in AdaBoost decreases the weights of correctly classified examples (by multiplying with β_t) and keeps the weights of incorrectly classified examples the same.
- The normalization factor Z_t plays a crucial role in balancing the distribution for the next iteration.
- After updating and normalizing, the weak learner h_t achieves an accuracy of exactly $\frac{1}{2}$ under D_{t+1}, ensuring the boosting algorithm focuses on previously misclassified examples.