Homework 2

CS 5350 - Machine Learning
Nick R. Porter

September 25, 2017

Exercise 1. Boolean Functions

- 1. $x_1 \vee \neg x_2 \vee x_3$
  This is linearly separable.
  $$x_1 - x_2 + x_3 >= 0$$
- 2. $(x_1 \vee x_2) \wedge (x_2 \vee x_3)$
  This is linearly separable.
  $$(x_1 + x_2) + (x_2 + x_3) >= 2$$
- 3. $(x_1 \wedge \neg x_2) \vee x_3$
  This is linearly separable.
  $$(x_1 - x_2) + x_3 >= 0$$
- 4. x_1 xor x_2 xor x_3
  This is not linearly separable.
- 5. $\neg x_1 \wedge x_2 \wedge \neg x_3$
  This is linearly separable.
  $$-x_1 + x_2 - x_3 >= 1$$

Exercise 2. Mistake Bound Model of Learning

(a) $|C| = 80$

(b) We need to define what it means to make a mistake. When the below inequality is true we have made a mistake, and when it is false we have made a correct prediction.

$$y^t((|x_1^t| - l) * (|x_2^t| - l)) \leq 0$$

(c) If we get an error we need to change l by the amount that it was wrong. We will also know if our prediction is a + or a - label. If our prediction is a + and the actual value is a - we update l by -1. If our prediction is a - and the actual value is a +, we update l by 1.

(d) Code below:

```
L = size(C) / 2
for all example in training:
    prediction = model.predict(example)
    if madeMistake(prediction, example):
        if prediction > 0
            L += -1
        else:
            L += 1
```

The max number of errors it can make is half notthe size of the **concept class:** $\frac{|C|}{2}$.

2. The algorithm will stop making mistakes when we have exactly M experts left. In the previous example we get ride of approximately half of the experts at each error giving us $O(logN)$. At each error we get ride of approximately half of the proportion of the individual who are wrong.

For every t at which there is a mistake, at least x experts in C_t are wrong so we can derive:

$$|C_{i+1}| \leq \frac{|C_i|}{2}$$

If we let M = the number of

$$|C_I| \leq \frac{|C_1|}{2^{N/M}}$$

We know that

$$|C_I| \geq 1$$

Now we can get

$$1 \leq \frac{N}{2^{N/M}}$$

If we rearrange we get

$$M \leq log_2(N/M)$$

Exercise 3. Experiments

- 1. I used Python and I used numpy array's to represent the feature vectors. I parsed through all the training data and created sparse vectors to represent the features. The foundation of my code is the Perceptron class which contains the weights, learning rate, bias, bias2, average weights, and number of updates. It has a initializer which takes a learning rate. The perceptron class has a predict(x) function which takes the example x and returns either 1 or -1 based on what the perceptrons weights and bias' are. It also has a train method which takes the training data and the number of epochs. It runs through all the training data and makes the appropriate updates to the weights, learning rate, bias, and averaged weights. main.py uses the perceptron algorithm and loads the data and evaluates it on the perceptron model. It also looks through all the hyper-parameters.

- 2. The most frequent label in the training data is **+1**. If we predict a 1 for our test dataset we get an accuracy of **57.3%** and when we do this on the dev data set we get an accuracy of **54.9%**.

- **Simple Perceptron**

  (a) All learning rates performed quite similarly. However, I found that $\eta = 0.10$ to generalize the best.

  (b) Cross-validation accuracies using the hyper-parameter above:

  - training00.data: **92.6%**
  - training01.data: **86.4%**
  - training02.data: **81.4%**
  - training03.data: **82.6%**
  - training04.data: **84.6%**
  - Average: **85.52%**

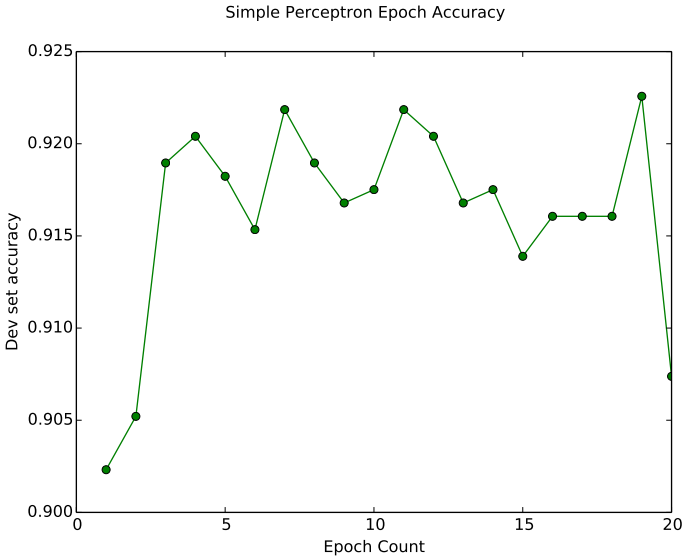
  From here on out, for the simple perceptron I used 19 epochs as I found it to produce the best results.

  (c) My perceptron algorithm performed **2349** updates on *phishing.train* when using the hyper-parameters above.

  (d) On the development set data my algorithm had an accuracy of: **91.5%**

  (e) On the test data my algorithm had an accuracy of: **92.5%**

  (f) Below is a plot of the learning curve where the x-axis is the epoch and y is the dev set accuracy. From 1 epoch to 20 epochs.



- **Perceptron with dynamic learning rate**

  (a) I found the best hyper-parameter to be $\eta = 1.0$. It had the best average accuracy when running 5-fold CV.

  (b) Cross-validation accuracies using the hyper-parameter above:

  - training00.data: **87.9%**
  - training01.data: **89.5%**
  - training02.data: **88.9%**
  - training03.data: **89.4%**
  - training04.data: **87.2%**
  - Average: **88.58%**

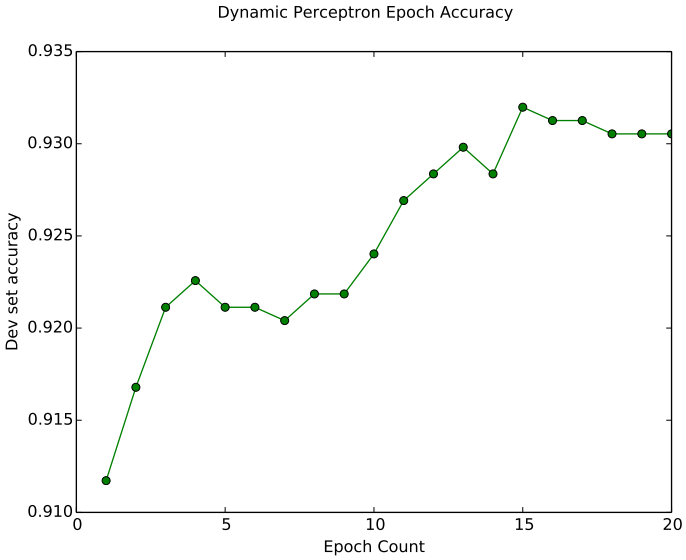
  From here on out, for the dynamic perceptron I used 16 epochs as I found it to produce the best results.

  (c) My perceptron algorithm performed **2079** updates on *phishing.train* when using the hyper-parameters above.

  (d) On the development set data my algorithm had an accuracy of: **91.6%**

  (e) On the test data my algorithm had an accuracy of: **92.5%**

  (f) Below is a plot of the learning curve where the x-axis is the epoch number and y is the dev set accuracy. From 1 epoch to 20 epochs.



- **Margin Perceptron**

  (a) I found the best hyper-parameters to be $\eta = 1$ and $\mu = 0.1$.

  (b) Cross-validation accuracies using the hyper-parameter above:

  - training00.data: **92.3%**
  - training01.data: **91.1%**
  - training02.data: **89.5%**
  - training03.data: **88.8%**
  - training04.data: **89.4%**
  - Average: **90.22%**

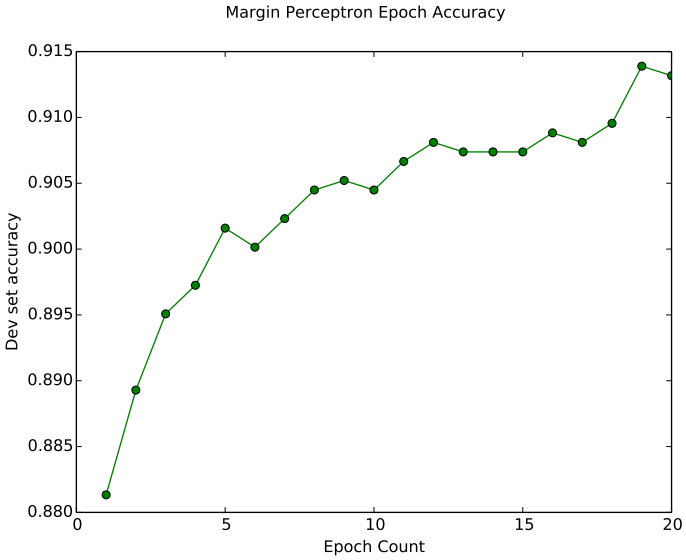
  From here on out, for the margin perceptron I used 16 epochs as I found it to produce the best results.

  (c) My perceptron algorithm performed **2922** updates on *phishing.train* when using the hyper-parameters above.

  (d) On the development set data my algorithm had an accuracy of: **92%**

  (e) On the test data my algorithm had an accuracy of: **93.1%**

  (f) Below is a plot of the learning curve where the x-axis is the epoch and y is the dev set accuracy.



- **Averaged Perceptron**

  (a) In my experiments all learning rates performed very similarly. I found $\eta = 0.10$ to perform the best.

  (b) Cross-validation accuracies using the hyper-parameter above:

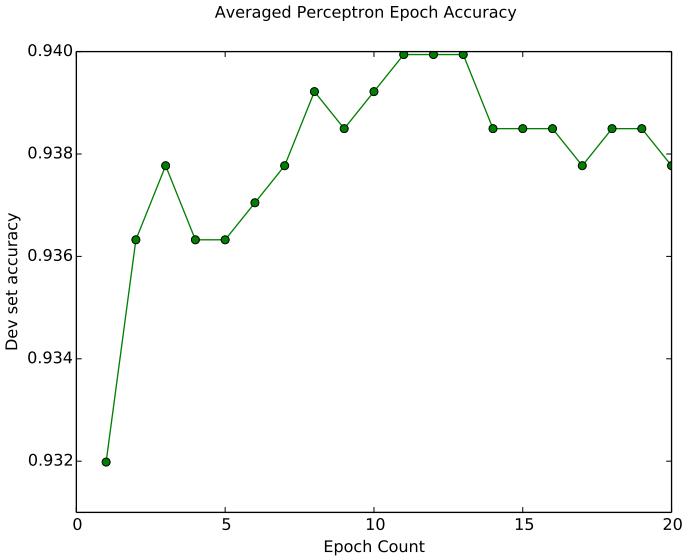
  - training00.data: **94.9%**
  - training01.data: **90.2%**
  - training02.data: **86.5%**
  - training03.data: **86.5%**
  - training04.data: **88.2%**
  - Average: **89.26%**

  (c) My perceptron algorithm performed **1425** updates on *phishing.train* when using the hyper-parameters above. However, since the averaged perceptron updates every time regardless of the result it was actually updated quite a bit more. So it was updated once for every training example.

  (d) On the development set data my algorithm had an accuracy of: **94.2%**

  (e) On the test data my algorithm had an accuracy of: **94.1%**

  (f) Below is a plot of the learning curve where the x-axis is the epoch and y is the dev set accuracy.



For part (b) I simply kept a counter in my perceptron class that incremented each time an update was performed.
As you can see increasing the total number of epochs isn't always a good idea for all variants of the perceptron. We may want to use the number of epochs as a hyper-parameter and use cross-validation to find the best number of epochs. Sometimes increasing the number of epochs doesn't have much of an effect as you can see with the averaged perceptron.
I updated t for each training example.