CS 5350/6350: Machine Learning Fall 2018

Homework 2

Abhinav Kumar (u1209853)

October 2, 2018

1 Warm up: Linear Classifiers and Boolean Functions

[10 points] Please indicate if each of the following boolean functions is linearly separable. If it is linearly separable, write down a linear threshold unit equivalent to it.

1. [2 points] $\neg x_1 \vee x_2 \vee \neg x_3$
   Linearly Separable.
   $-x_1 + x_2 - x_3 \geq -1$

2. [2 points] $(x_1 \vee x_2) \wedge x_3$
   Linearly Separable.
   $x_1 + x_2 + 2x_3 \geq 3$

3. [2 points] $(x_1 \wedge \neg x_2) \vee \neg x_3$
   Linearly Separable.
   $x_1 - x_2 - 2x_3 \geq -1$

4. [2 points] x_1 xor x_2 xor x_3
   Not Linearly Separable

5. [2 points] $x_1 \wedge \neg x_2 \wedge x_3$
   Linearly Separable.
   $x_1 - x_2 + x_3 \geq 2$

2 Feature transformations

[10 points] Consider the concept class C consisting of functions f_r defined by a radius r as follows:

$$f_r(x_1, x_2) = \begin{cases} +1 & 17x_1^4 - 16x_2^3 \leq r \\ -1 & \text{otherwise} \end{cases}$$

Note that the hypothesis class is *not* linearly separable in $\mathbb{R}^2$.

Construct a function $\phi(x_1, x_2)$ that maps examples to a new space, such that the positive and negative examples are linearly separable in that space. The answer to this question should consist of two parts:

1. A function ϕ that maps examples to a new space.
   $\phi : (x_1, x_2) \rightarrow (x_1^4, x_2^3)$

2. A proof that in the new space, the positive and negative points are linearly separated. You can show this by producing such a hyperplane in the new space (i.e. find a weight vector $\mathbf{w}$ and a bias b such that $\mathbf{w}^T\phi(x_1, x_2) \geq b$ if, and only if, $f_r(x_1, x_2) = +1$.
   Choose $\mathbf{w} = (-17, 16)$ and $b = -r$.

   So, $\mathbf{w}^T\phi(x_1, x_2) - b = -17x_1^4 + 16x_2^3 + r$
   The point is assigned label $+1$ only when
   $\mathbf{w}^T\phi(x_1, x_2) - b \geq 0$ or $-17x_1^4 + 16x_2^3 + r \geq 0$ or $17x_1^4 - 16x_2^3 \leq r$ (as desired)
   and -1 otherwise.

Hint: The feature transformation ϕ should not depend on r.

3 Mistake Bound Model of Learning

In both the questions below, we will consider functions defined over n Boolean features. That is, each example in our learning problem is a n-dimensional vector from $\{0, 1\}^n$. We will use the symbol $\mathbf{x}$ to denote an example and $\mathbf{x}_i$ denotes its i^{th} element. (We will assume that there is no noise involved.)

For all questions below, it is not enough to just state the answer. You need to justify your answer with a short proof.

1. Consider the concept class $\mathcal{C}_1$ defined as follows: Each element of $\mathcal{C}_1$ is defined using a fixed instance $\mathbf{z} \in \{0, 1\}^n$ as follows:

$$f_{\mathbf{z}}(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{z} \\ 0 & \mathbf{x} \neq \mathbf{z}. \end{cases}$$

   That is, the function $f_{\mathbf{z}}$ predicts 1 if, and only if, the input to the function is $\mathbf{z}$.

   Our goal is to come up with a mistake bound algorithm that will learn any function $f \in \mathcal{C}_1$.

   (a) [5 points] Determine $|\mathcal{C}_1|$, the size of concept class.
   $\mathbf{z} \in \{0, 1\}^n$ happens to be a n dimensional vector with each dimension of $\mathbf{z}$ can take value 0 or 1. Hence, the size of concept class is 2^n

   (b) [15 points] Write a mistake bound learning algorithm for this concept class that makes no more than *one* mistake on any sequence of examples presented to it. Please write the algorithm concisely in the form of pseudocode.

Prove the mistake bound for this algorithm.

```
Algo 1 : Input $x, C_0$. Output true function $f_0(x)$
  • Initialize $C = C_0$, the set of all possible functions
  • For every example $x$
      – Prediction = majority of functions in $C_i$
      – If Prediction ! = $f_z(x)$,
        eliminate those functions which formed majority
```

Proof :
This is a standard halving algorithm and therefore it can't make more than $O(logn)$ mistakes. Note that here the concept class functions are mutually exclusive in giving output as 1 since each of the function gives 1 for a different z

Let the true function be $f_0(x)$ which is non-zero at z_0.

In whatever sequence the examples are taken, for all point $x \neq z_0$, only one function gives a non-zero output and the majority of the functions give a 0. Hence, the prediction will be 0 which is in agreement with the true function.

When $x = z_0$, the true function $f_0(x)$ reports a 1 but still the majority will be 0. Hence, there is an error in prediction resulting in elimination of majority. But, majority consists of every other function other than f_0, thus eliminating every other function.

Hence, only function remaining in C_i is $f_0(x)$ and then it will not make any mistake for any other example. Hence we have learnt the correct function by making no more than one mistake

2. Suppose we have a concept class $\mathcal{C}_2$ that consists of exactly n functions $\{f_1, f_2, \cdots, f_n\}$, where each function f_i is defined as follows:

$$f_i(\mathbf{x}) = \mathbf{x}_i.$$

That is, the function f_i returns the value of the i^{th} feature.

(a) [5 points] How many mistakes will the algorithm **CON** from class make on any function from this concept class?
The input vectors are boolean and therefore choosing an example randomly will atmax remove one function.

eg: Let $n = 3$ and true function be f_1. Let the input x be $(0, 1, 0)$. Then, we can have output 0 from two functions f_1 and f_3.

Thus, at max, only 1 function can be eliminated at a time. And one has to get $n-1$ mistakes to remove $n-1$ functions and therefore the number of mistakes **CON** algorithm makes will be $n-1$ and so will be $O(n)$

(b) [5 points] How many mistakes will the Halving algorithm make on any function from this concept class?
Whenever a mistake happens, more than half of the functions remaining in concept class will be removed. Therefore in the end, $1 = |C_i| \leq \frac{1}{2}|C_{i-1}| ... \leq \frac{1}{2^n}|C_0|$

So, we have $|C_0| > 2^n$ or $n < log(|C_0|)$ The **Halving** algorithm searches like a binary search and therefore the mistakes will be in $O(log(n))$

4 The Perceptron Algorithm and its Variants

For this question, you will experiment with the Perceptron algorithm and some variants on a data set.

4.1 The task and data

We will be using the Diabetic Retinopathy dataset from the UCI Machine Learning repository [1]. The dataset consists of features extracted from images and the goal is to predict whether an image contains signs of diabetic retinopathy or not.

Using this labeled data, we want to build a classifier that identifies whether a new retinal image shows signs of diabetic retinopathy or not.

The data has been preprocessed into a standard format. Use the training/development/test files called `diabetes.train`, `diabetes.dev` and `diabetes.test`. These files are in the LIBSVM format, where each row is a single training example.

4.2 Algorithms

You will implement several variants of the Perceptron algorithm. Note that each variant has different hyper-parameters, as described below. Use 5-fold cross-validation to identify the best hyper-parameters as you did in the previous homework. To help with this, we have split the training set into five parts `training00.data`–`training04.data` in the folder `CVSplits`.

1. **Simple Perceptron:** Implement the simple batch version of Perceptron as described in the class. Use a fixed learning rate η chosen from $\{1, 0.1, 0.01\}$. An update will be performed on an example $(\mathbf{x}, y)$ if $y(\mathbf{w}^T\mathbf{x} + b) < 0$ as:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x},$$
$$b \leftarrow b + \eta y.$$

   Hyper-parameter: Learning rate $\eta \in \{1, 0.1, 0.01\}$

   Two things bear additional explanation.

[1] `https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set`

   (a) First, note that in the formulation above, the bias term b is explicitly mentioned. This is because the features in the data do not include a bias feature. Of course, you could choose to add an additional constant feature to each example and not have the explicit extra b during learning. (See the class lectures for more information.) However, here, we will see the version of Perceptron that explicitly has the bias term.

   (b) Second, in this specific case, if $\mathbf{w}$ and b are initialized with zero, then the fixed learning rate will have no effect. To see this, recall the Perceptron update from above.

       Now, if $\mathbf{w}$ and b are initialized with zeroes and a fixed learning rate η is used, then we can show that the final parameters will be equivalent to having a learning rate 1. The final weight vector and the bias term will be scaled by η compared to the unit learning rate case, which does not affect the sign of $\mathbf{w}^T\mathbf{x} + b$.

       To avoid this, you should initialize the all elements of the weight vector $\mathbf{w}$ and the bias to a small random number between -0.01 and 0.01.

2. **Decaying the learning rate:** Instead of fixing the learning rate, implement a version of the Perceptron algorithm whose learning rate decreases as $\frac{\eta_0}{1+t}$, where η_0 is the starting learning rate, and t is the time step. Note that t should keep increasing across epochs. (That is, you should initialize t to 0 at the start and keep incrementing it each time a new example is encountered.)

   Hyper-parameter: Initial learning rate $\eta_0 \in \{1, 0.1, 0.01\}$

3. **Margin Perceptron:** This variant of Perceptron will perform an update on an example $(\mathbf{x}, y)$ if $y(\mathbf{w}^T\mathbf{x} + b) < \mu$, where μ is an additional positive hyper-parameter, specified by the user. Note that because μ is positive, this algorithm can update the weight vector even when the current weight vector does not make a mistake on the current example. You need to use the decreasing learning rate as before.

   Hyper-parameters:

   (a) Initial learning rate $\eta_0 \in \{1, 0.1, 0.01\}$

   (b) Margin $\mu \in \{1, 0.1, 0.01\}$

   Note: When there is more than one hyper-parameter to cross-validate, you need to consider all combinations of the hyper-parameters. In this case, you will need to perform cross-validation for all pairs (η_0, μ) from the above sets.

4. **Averaged Perceptron** Implement the averaged version of the original Perceptron algorithm from the first question. Recall from class that the averaged variant of the Perceptron asks you to keep two weight vectors (and two bias terms). In addition to the original parameters $(\mathbf{w}, b)$, you will need to update the averaged weight vector $\mathbf{a}$ and the averaged bias b_a as:

   (a) $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{w}$

(b) $b_a \leftarrow b_a + b$

This update should happen once for every example in every epoch, *irrespective of whether the weights were updated or not for that example.* In the end, the learning algorithm should return the averaged weights and the averaged bias.

(Technically, this strategy can be used with any of the variants we have seen here. For this homework, we only ask you to implement the averaged version of the original Perceptron. However, you are welcome to experiment with averaging the other variants.)

5. **Aggressive Perceptron with Margin, (For 6350 Students)** This algorithm is an extension of the margin Perceptron and performs an aggressive update as follows:

   If $y(\mathbf{w}^T\mathbf{x}) \leq \mu$ then
   (a) Update $\mathbf{w}_{new} \leftarrow \mathbf{w}_{old} + \eta y \mathbf{x}$

   Unlike the standard Perceptron algorithm, here the learning rate η is given by

$$\eta = \frac{\mu - y(\mathbf{w}^T\mathbf{x})}{\mathbf{x}^T\mathbf{x} + 1}$$

   As with the margin Perceptron, there is an additional positive parameter μ.

   Explanation of the update. We call this the aggressive update because the learning rate is derived from the following optimization problem:

   When we see that $y(\mathbf{w}^T\mathbf{x}) \leq \mu$, we try to find new values of $\mathbf{w}$ such that $y(\mathbf{w}^T\mathbf{x}) = \mu$ using

$$\begin{aligned} \min_{\mathbf{w}_{new}} \quad & \tfrac{1}{2}||\mathbf{w}_{new} - \mathbf{w}_{old}||^2 \\ \text{such that} \quad & y(\mathbf{w}^T x) = \mu. \end{aligned}$$

   That is, the goal is to find the smallest change in the weights so that the current example is on the right side of the weight vector.

   By substituting (a) from above into this optimization problem, we will get a single variable optimization problem whose solution gives us the η defined above. You can think of this algorithm as trying to tune the weight vector so that the current example is correctly classified right after the update.

   Implement this aggressive Perceptron algorithm.

   Hyper-parameters: $\mu \in \{1, 0.1, 0.01\}$

4.3 Experiments

For all 5 settings above (4 for undergraduate students), you need to do the following things:

1. Run cross validation for **ten** epochs for each hyper-parameter combination to get the best hyper-parameter setting. Note that for cases when you are exploring combinations of hyper-parameters (such as the margin Perceptron), you need to try out all combinations.

2. Train the classifier for **20** epochs. At the end of each training epoch, you should measure the accuracy of the classifier on the development set. For the averaged Perceptron, use the average classifier to compute accuracy.

3. Use the classifier from the epoch where the development set accuracy is highest to evaluate on the test set.

4.4 What to report

1. [8 points] Briefly describe the design decisions that you have made in your implementation. (E.g, what programming language, how do you represent the vectors, etc.)

    - Programming Language = Python,
    - Vectors = As numpy array
    - `Perceptron` is a class which has all kinds of update, initialize and predict functions.
    - `learning_rate.py` is in a separate file which decides the learning rate
    - `signum.py` function has been reimplemented since we want an output of 1 if the variable is ≥ 0 and 0 if variable < 0
    - `util.py` contains frequently used utilities for doing this assignment
    - `driver.py` file is the actual driver for the code.
    - There can be instances (because of libsvm format) in which different splits do not get to know how many feature vectors are there. Used a `max_col_count` variable so that the dimensions across all the splits are consistent.

2. [2 points] *Majority baseline:* Consider a classifier that always predicts the most frequent label. What is its accuracy on test and development set?
   Accuracy on dev set = 56.00%
   Accuracy on test set = 52.24%

3. [10 points per variant] For each variant above (5 for 6350 students, 4 for 5350 students), you need to report:

    (a) The best hyper-parameters

    (b) The cross-validation accuracy for the best hyperparameter

(c) The total number of updates the learning algorithm performs on the training set

(d) Development set accuracy

(e) Test set accuracy

(f) Plot a *learning curve* where the x axis is the epoch id and the y axis is the dev set accuracy using the classifier (or the averaged classifier, as appropriate) at the end of that epoch. Note that you should have selected the number of epochs using the learning curve (but no more than 20 epochs).

Perceptron	HP		CV	Num Updates	Dev	Test
	lr	μ	Accuracy		Accuracy	Accuracy
Simple	0.1	-	63.47	6190	76.00	70.65
Simple with decay	1	-	65.60	6150	76.00	67.66
Margin	0.01	0.01	66.00	5465	67.50	69.65
Average	1	-	62.67	6224	73.00	70.65
Aggressive	-	0.01	59.73	5907	72.50	70.65

Table 1: Results

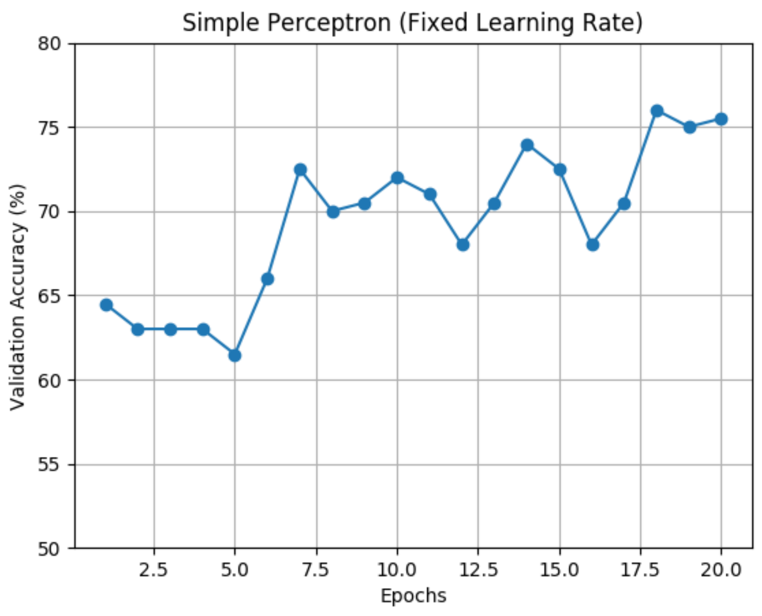

Figure 1: Learning Curve for Simple Perceptron

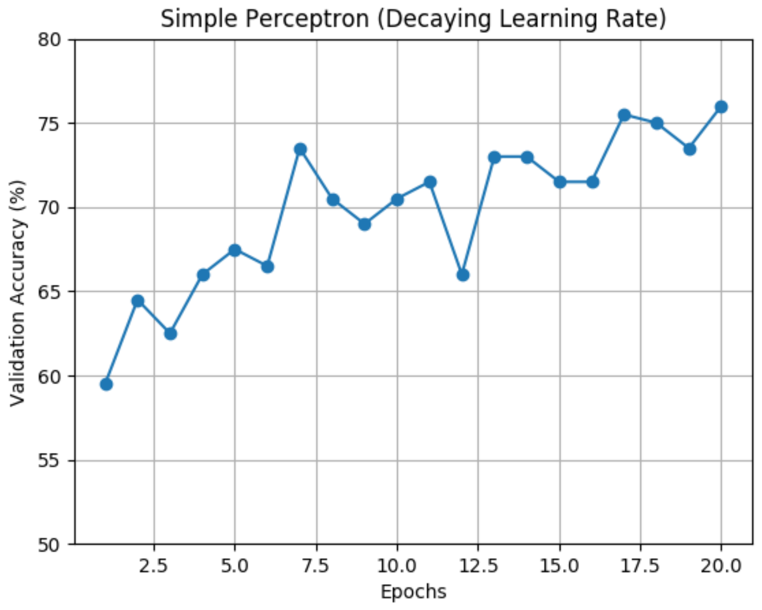

Figure 2: Learning Curve for Simple Perceptron with decaying learning rate

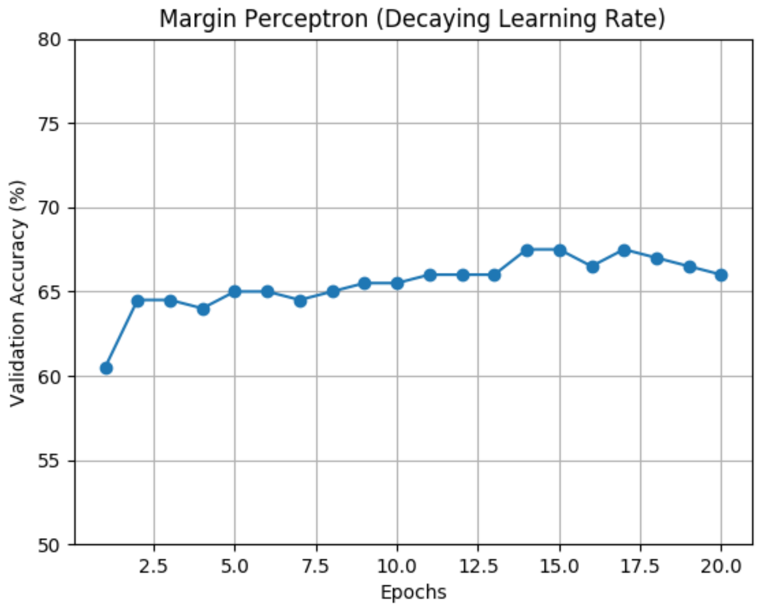

Figure 3: Learning Curve for Margin Perceptron

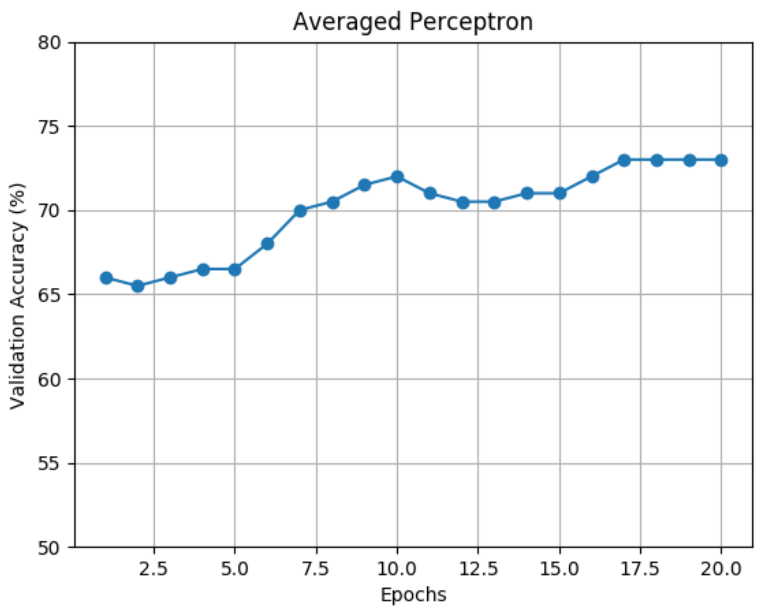

Figure 4: Learning Curve for Averaged Perceptron

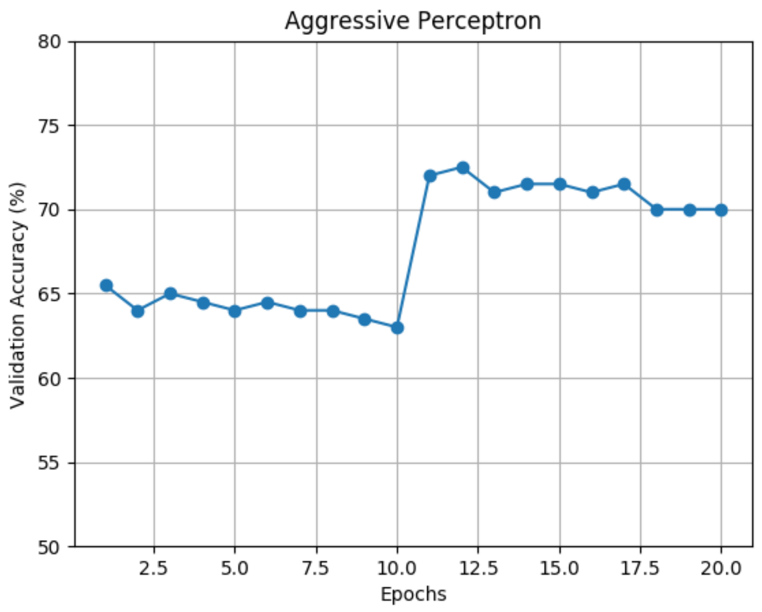

Figure 5: Learning Curve for Aggressive Perceptron