Eli Pinkus
CS 155
Set #1

## Problem 1

### Problem A

A hypothesis set represents the set of candidate functions that a particular learning model could implement to best approximate the objective function in question.

### Problem B

A linear model in $d$ dimensions implements $\theta(w^T x + b)$, where $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ and $\theta$ is some function (i.e. sign function, sigmoid, etc). The parameters of the model are $w$ and $b$, thus the hypothesis set of the linear model in $d$ dimensions can we written as the product of the set of all $d$-tuples, $w \in \mathbb{R}^d$ and the set of all real numbers $b \in \mathbb{R}$. So the hypothesis set is tuples

$$\mathcal{H} = \{w, b|\ (w, b) \in \mathbb{R}^d \times \mathbb{R}\}$$

### Problem C

Overfitting occurs when a model fits noise in the training data excessively and at the expense of generalization performance. If the test error is drastically larger than the training error than overfitting is likely taking place.

### Problem D

One can prevent overfitting by using a validation set or cross validation which involves the withholding of a subset of the data (or a “fold” of the data) from the training algorithm and scores each candidate model relative to that withheld set which ideally would serve as a better estimate of out of sample performance since it was scored on data that the algorithm hasn’t seen and therefore can’t over fit upon.

Another way one can prevent overfitting is by using a regularized model. In generic terms, a regularizer puts restrictions on the flexibility of a model to fit (and over fit) the training data. Reduction of this freedom results in a hypothesis set with a lower variance which is a good remedy for overfitting.

### Problem E

Training data is the data that a learning algorithm uses to choose the ‘best’ hypothesis out of the hypothesis set. The learning algorithm generally evaluates some sort of error on the training data and seeks to minimize this error. Test data is a set of data that is not considered in the training of the model. After training upon the training data, one would evaluate or score the model on the test data that in order to better ascertain how the model generalizes to predict data that it hasn’t seen yet. You don’t change your model based on information from the test data because if this is done the resulting model will have different generalization than what would be predicted by scoring the test data. This is because altering the model in consideration of the test data is essentially the same as doing some sort of training on the test data which means that model has a bias toward that data and scoring on that test data will give different results than the ultimate generalization behavior.

## Problem F

We assume that our dataset is sampled identically and independently from $P(x, y)$.

## Problem G

We can consider the 'bag of words' approach in which the input space $\mathcal{X}$ consists of tuples that represent either the presence (1) or lack (0) of given words in the message. So,

$$\mathcal{X} = \{x \mid x \in \{0,1\}^d\}$$

Where d is the number of words being considered.
The output space could be an element in the set $\{0,1\}$ where 0 represents not spam and 1 represents spam. So,

$$\mathcal{Y} = \{0,1\}$$

## Problem H

In k-fold cross validation, the training data is split into k many equal partitions (or as close as possible).

The following procedure is then followed for each of the k partitions:

    Leave out the given partition and train on the other 4 partitions
    Score upon the left out partition

The k many scores that are produced upon each left out partition are used as the validation error (usually formed by adding/averaging the k scores) which is considered to be a better predictor of out of sample generalization so you can choose between models based on the results of the validation error produced by the above procedure.

Eli Pinkus
CS 155
Set #1

## Problem 2

### Part A

We begin with

$$\mathbb{E}_S[E_{\text{out}}(f_S)] = \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right]\right] = \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - y(x)\right)^2\right]\right]$$

$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x) + F(x) - y(x)\right)^2\right]\right]$$

$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2 + 2\left(f_S(x) - F(x)\right)\left(F(x) - y(x)\right)\right]\right]$$

$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right] + 2\left(F(x) - y(x)\right) \cdot \mathbb{E}_S[(f_S(x) - F(x))]\right]$$

$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right] + 2\left(F(x) - y(x)\right) \cdot \mathbb{E}_S[f_S(x)] - \mathbb{E}_S[F(x)]\right]$$

$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right] + 2\left(F(x) - y(x)\right) \cdot [F(x) - F(x)]\right]$$

$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right] + 2\left(F(x) - y(x)\right) \cdot 0\right]$$

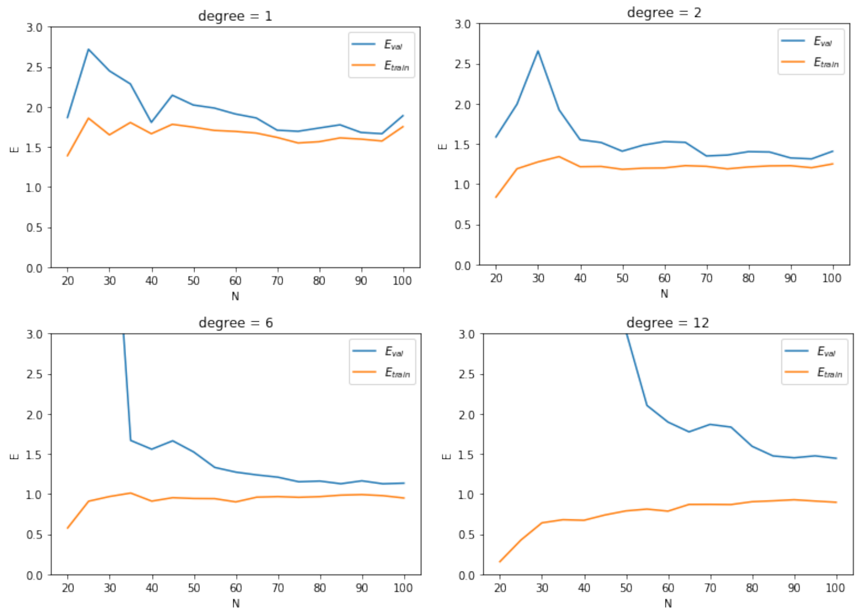
$$= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right]\right]$$

$$= \mathbb{E}_x[\text{Var}(x) + \text{Bias}(x)]$$

Eli Pinkus
CS 155
Set #1

## Part B

Learning curves



## Part C

The highest bias model is the model that has the worst test error since a high bias is associates with a sets inability to fit the nature of the target function. If we take $E_{\text{val}}$ as an approximation of generalization error than the degree 1 polynomial has the highest bias.

## Part D

The degree 12 polynomial model shows clear signs of overfitting the noise of the training data. It has the greatest difference between training error and validation error which indicates that the model with the high degrees of freedom with a 12 degree polynomial, fit the noise at the expense of generalization performance which means that the variance of the hypothesis set is rather high.

## Part E

The $E_{\text{val}}$ for the second-degree model leveled off fairly quick, so if we take $E_{\text{val}}$ as an approximation of generalization error we can see that the effectiveness of the model would not appreciably increase with the addition of more training points

## Part F

Training error is generally lower than validation error because training error is calculated on data that the polynomial fitting algorithm was able to consider in the production of a best fit polynomial. Thus, the resulting model is biased toward the training set and tends to replicate points from the training set better than it predicts entirely new data points which are used to calculate $E_{\text{val}}$

## Part G

If we take $E_{\text{val}}$ to be a reasonable approximation of out of sample generalization, than we would expect the model with the best $E_{\text{val}}$ at a high $N$ to be the best performer on unseen data from the same distribution as the training data. For the degree 6 polynomial, the $E_{\text{val}}$ flattens at about 1.2 which is noticeably better than the other models. Therefore, we would expect the degree 6 model to perform the best on unseen data given the above assumption that we take $E_{\text{val}}$ to be a reasonable approximation of out of sample generalization.

Eli Pinkus
CS 155
Set #1

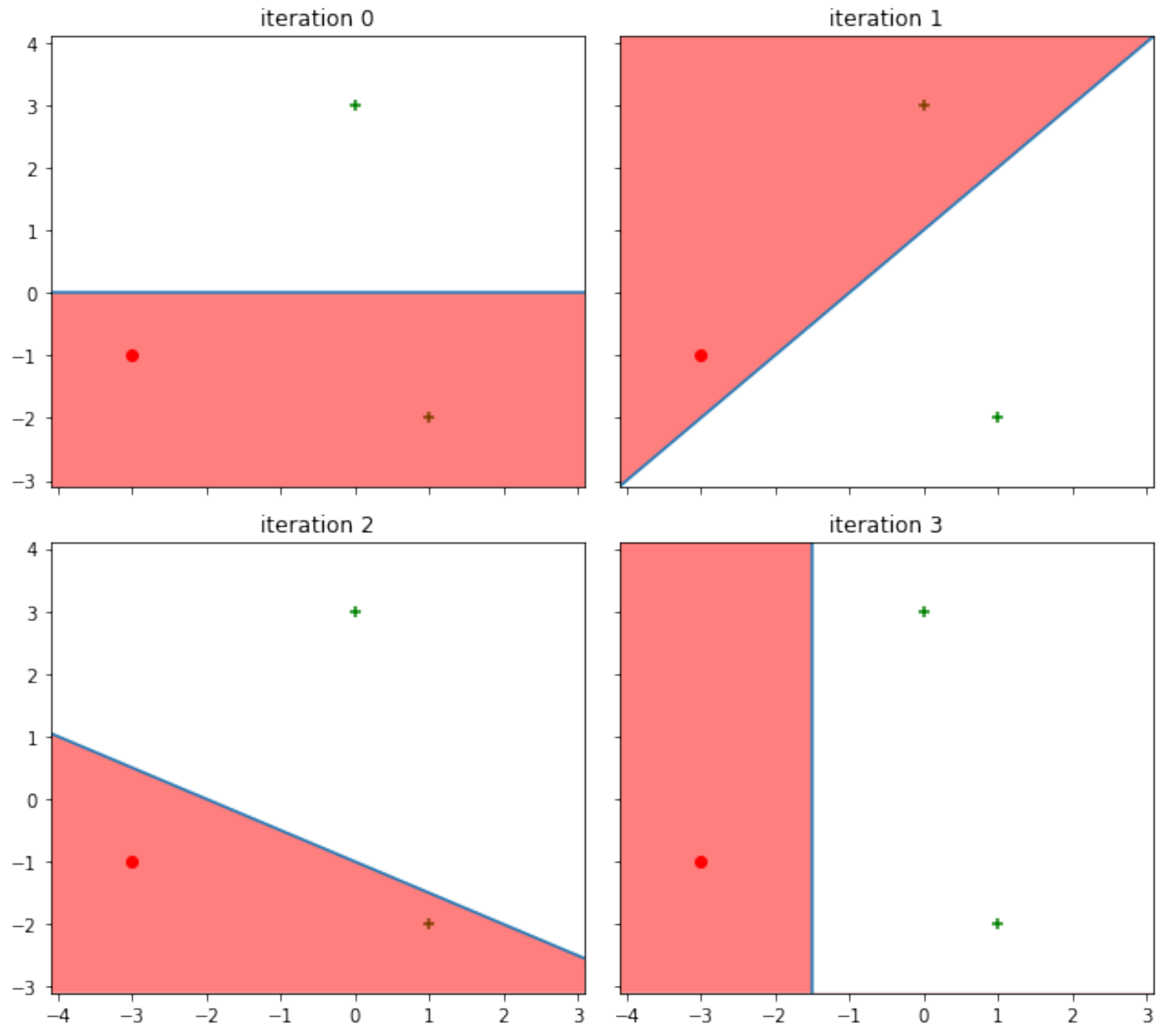
## Problem 3

### Part A

Table of results:

| T | B | $w_1$ | $w_2$ | $x_1$ | $x_2$ | y |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | -2 | 1 |
| 1 | 1 | 1 | -1 | 0 | 3 | 1 |
| 2 | 2 | 1 | 2 | 1 | -2 | 1 |
| 3 | 3 | 2 | 0 | | | |

Visualization of iterations:



Eli Pinkus
CS 155
Set #1

## Part B

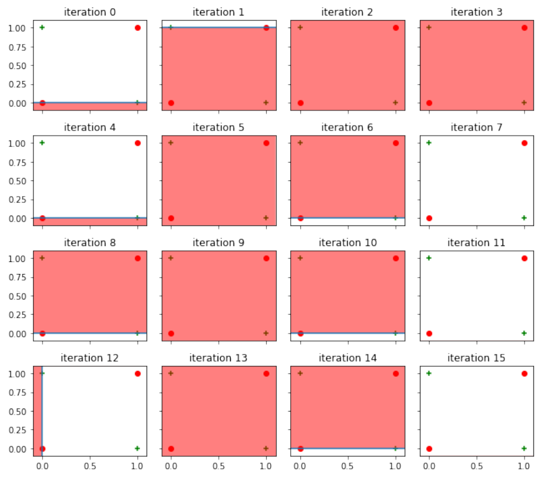
The smallest data set that is not linearly separable, such that no three points are collinear contains 4 data points. It is clear that any non-conlinear 3 point data set can be separated by a line because the line could always draw the line to mark either 0,1,2 or 3 of the points as + and the remaining points as – or vice versa. You can create a 4 point set that can't be separated by a line with any number of generating rules, for example the XOR function as seen in part A. Thus, for a 2D dataset the smallest dataset that is not linear separable is of size 4.

With the 3D case you can use a similar reasoning. With any 4 point set, such that the points aren't all coplanar, you can always place a plane such that any 4,3,2,1, or 0 points can be 'cut' off from the rest of the points via a dividing plane. Thus, you could create all possible dichotomies of a 4 point set. However, with 5 points you could conceive of an arrangement of points that could not be separated by a plane. For example, picture a cube with vertex points +1 and in the center of the cube there is a -1 point, this configuration cannot be properly classified by a plane. Thus, for a 3D dataset the smallest dataset that is not linear separable is of size 5.

For the N-dimensional case, the smallest data set that is not linearly separable, such that no $N + 1$ points are coplanar is of size $N + 2$.

## Part C

The PLA will never converge. This is because convergence of the PLA depends on there being a point during the iterations of the PLA where there are no misclassified points. If at any step there is a misclassified point, the PLA will continue to update with respect to that (or any other) misclassified point. Since the data is not linearly separable there will never come a time at which all points are correctly classified.



Eli Pinkus
CS 155
Set #1

## Problem 4

### Part A

We should define $w$ to be a vector of dimension $d+1$ of the form $\vec{w} = (w_0, w_1, \ldots, w_d)$, where $w_0$ is the bias term. In order to make this work we must also define an artificial coordinate in all the $x$ vectors. Namely we will introduce an $x_0$ term that we will set equal to 1. That way we compute $w^T x = w_0 x_0 + w_1 x_1 + \cdots + w_d x_d = w_0 + w_1 x_1 + \cdots + w_d x_d$ and if $w_0 = -b$ we have:

$w^T x = -b + w_1 x_1 + \cdots + w_d x_d$ as desired.

### Part B

We have:

$$L(f) = \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Taking derivatives, and applying the linearity of the derivative gives:

$$\nabla_w L(f) = \sum_{i=1}^{N} \partial_w (y_i - w^T x_i)^2 = -2(y_i - w^T x_i)x = -2\sum_{i=1}^{N}(y_i - w^T x_i)x$$

### Part C

Implemented in code

### Part D

When the SGD starts at the different values of $w$ as in the animation, we can see that the algorithm ultimately converges at around the same time regardless of the starting point. This likely has to do with the fact that in our SGD implementation the effective learning rate has an implicit correlation with gradient size which means that the algorithm will take larger steps when it is on a 'steeper' part of the surface. Since the squared loss surface is a paraboloid, The further away from the minimum, the larger the gradient and therefore the larger step. This compensation for distance away from the optimum essentially allows runs with further 'distance' to travel to reach the minimum to catch up to the other ones so that they all basically converge at the same time.

Between the two different datasets, the convergence behavior is fairly similar. In both cases, the algorithm seems to take a fairly direct path from the starting point to the global minimum of the squared loss for each data set.

Eli Pinkus
CS 155
Set #1

## Part E

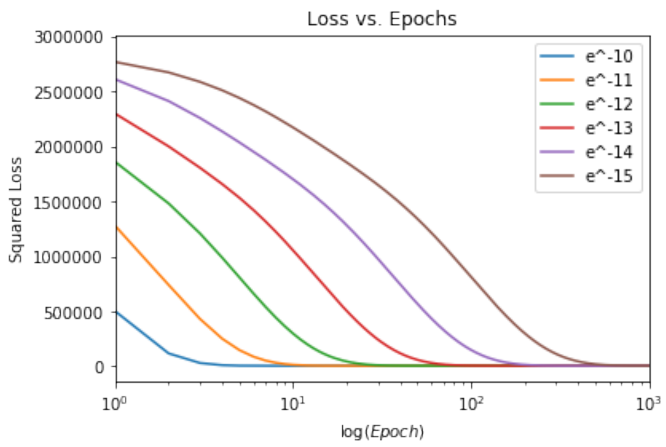
As $\eta$ decreases, the rate of change of the loss function decreases. The result is that it takes more epochs for the algorithm to approach the global minimum of the squared loss function. In the animation, this is shown by the higher values of $\eta$ converging before the lower values.

## Part F

Resulting weight vector:

```
[ -0.22789113,  -5.97854131,   3.988383  , -11.85701362,   8.91128886]
```

## Part G



The training error for all $\eta$ values ultimately converge to around the same minimum, however the larger $\eta$ values do so in fewer epochs similarly to the plot in part E.

## Part H

The SGD from part F returned $w = (-0.228, -5.979, 3.988, -11.857, 8.911)$

The analytical solution returned $w = (-0.316, -5.992, 4.015, -11.933, 8.991)$

So the solutions are very similar and they agree, whoever they have smalls differences that result from the iterative and stochastic nature of SGD relative to the close formed solution

## Part I

In situations where a closed form solution to an optimization problem is computationally difficult it may be better to use SGD to optimize, however in the case of linear regression, because the closed form solution is fairly simple if $d$ isn't huge, one should probably implement the closed form solution as it is the most accurate.

## Part J

A better stopping condition should relate to the amount of change from one iteration to the next. If the norm of the vector differences between one epoch and the next is below some minimum, one could terminate on that condition and it would be more sophisticated and efficient than choosing a predetermined number of epochs

## Park K

In SGD, the weight vector converges such that squared loss is minimized. In PLA, the weight vector will only converge once binary classification loss is exactly 0, which may never happen since correcting for an individual misclassified point may further misclassify other points. If the data is not linearly separable, PLA will never converge, while SGD with squared loss will still converge to minimum with respect to squared loss.