I am using two late hours on this assignment

1a. The hypothesis set is all the possible models that could be learned on the given input space, differentiating two different sides on the given space. The hypothesis set is determined by the type of model being learned with the given data.

1b. The hypothesis set of a linear model is the set of all straight lines in the given space. These lines take the form of $f(x|w,b) = w^Tx - b$.

1c. Overfitting is when the test error is a lot higher than the training error. This means that the model accurately maps the input information, but it does not do a good job approximating new data.

1d. We can prevent overfitting by using validation methods by splitting the data into training and testing sets. Some other methods include adding more data or decreasing the complexity of the model.

1e. The training data is the data used to create a model, and the test data is used to see how accurately the trained model maps outside data. If we look at the test set, it will bias our model towards the test data (won't accurately deal with the data used for training), when the model should be created just based on the information from the training data.

1f. We assume that our dataset is gathered in a random fashion and in a way that represents the entirety of the data space.

1g. The input space could be all the possible emails converted to a bag of words feature vector, and the output space could be a Boolean value (-1 or 1). Our machine learning algorithm would take in an email and assign a value to whether it is spam.

1h. The k-fold cross-validation procedure includes dividing the original training data into k equal parts and training on k - 1 parts and testing on the one leftover. By doing this k times on all the partitions of the original training set, it allows us to see how well the given training data/model will map outside data.

2a. Using the given definitions:

$$E_S[E_{out}(f_S)] = E_S[E_x[(f_S(x) - y(x))^2]] = E_x[E_S[(f_S(x) - y(x))^2]]$$

Adding F(x) and subtracting F(x) is the same as adding zero:

$$E_x[E_S[(f_S(x) - y(x))^2]] = E_x[E_S[(f_S(x) - y(x) + F(x) - F(x))^2]]$$

We can group the terms and then square them:

$$E_x[E_S[(f_S(x) - y(x) + F(x) - F(x))^2]] = E_x[E_S[((f_S(x) - F(x)) + (F(x) - y(x)))^2]]$$

$$= E_x[E_S[(f_S(x) - F(x))^2 + (F(x) - y(x))^2 + 2(f_S(x) - F(x))(F(x) - y(x))]]$$

$$= E_x[E_S[(f_S(x) - F(x))^2]] + E_x[E_S[(F(x) - y(x))^2]] + E_x[E_S[2(f_S(x) - F(x))(F(x) - y(x))]]$$

Stated in the definition, $F(x) = E_S[f_S(x)]$:

$$= E_x[E_S[(f_S(x) - F(x))^2]] + E_x[E_S[(F(x) - y(x))^2]] + E_x[E_S[2(\underline{f_S(x) - f_S(x)})(F(x) - y(x))]]$$

$$= E_x[E_S[(f_S(x) - F(x))^2]] + E_x[E_S[(F(x) - y(x))^2]] + 0$$

Simplifying:

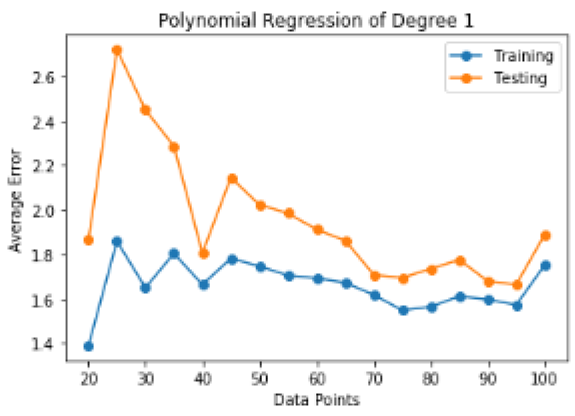
$$E_x[E_S[(f_S(x) - F(x))^2]] + E_x[E_S[(F(x) - y(x))^2]] = E_x[E_S[(f_S(x) - F(x))^2]] + E_x[(F(x) - y(x))^2]$$

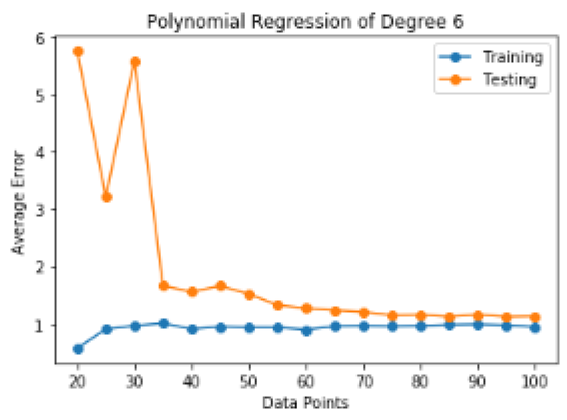
$$= E_x[E_S[(f_S(x) - F(x))^2] + (F(x) - y(x))^2]$$

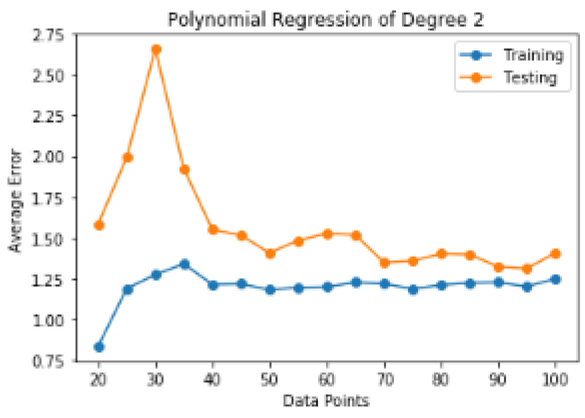
$$= E_x[\mathrm{Var}(x) + \mathrm{Bias}(x)]$$

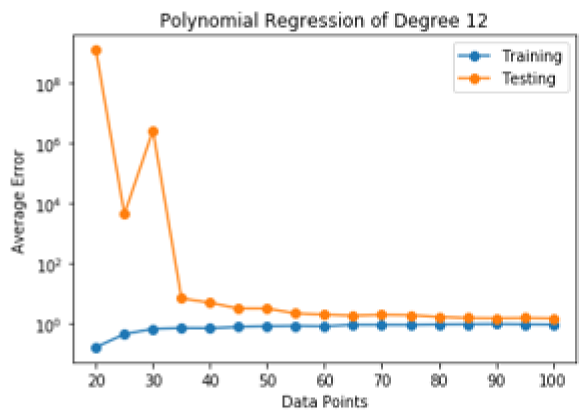
Thus, $E_S[E_{out}(f_S)] = E_x[\mathrm{Bias}(x) + \mathrm{Var}(x)]$.

2b.









2c. The 1st- degree regression has the highest bias. We can tell this because it has the highest average training error, which is a result of the low model complexity. This is an example of underfitting.

2d. The 12th- degree regression has the highest variance. We can tell this from the huge spike at the beginning of the testing error/it has the highest average testing error, while the training error stays low. This suggests that the model is too complex to accurately handle new information, hence overfitting.

2e. The quadratic model suggests that generalize better with more data points. The graph shows that the testing error dramatically improves from N = 20 to N = 40 points, and it gradually improves as N goes to 100 points. The training error evens out around N = 40 points, but it slightly improves as more data points are added. Overall, the model will improve as more data points are added, but the rate at which it improves decreases.

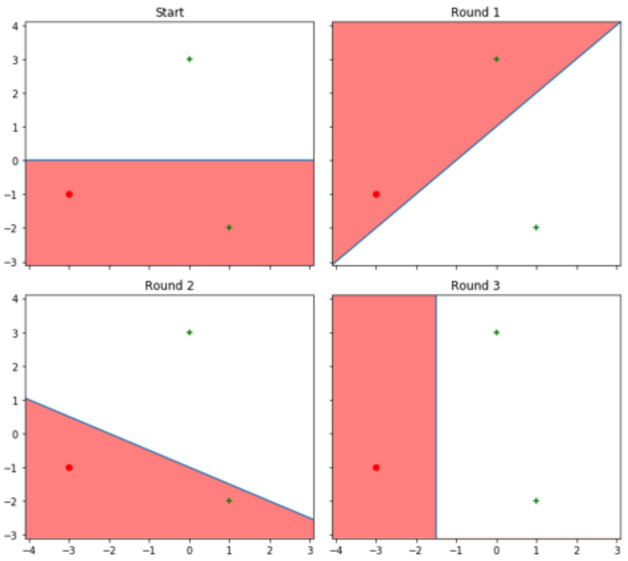
2f. The model is fit to accurately represent the data given, so it would make sense that the training error would be lower than the validation error since the model has not been exposed to the data in the validation set. The validation set should differ than the points given to train the model, so there will more areas where the model could inaccurately classify points.

2g. We would expect the 6th- degree model to perform best on some unseen data. This is basically seeing which model would have the lowest validation error for large numbers of data. Although there are spikes in validation error for all models between N = 20 and N = 40, the 6th – degree model flattens out to a low validation error for N > 40, and this means that most accurately fits the model we want.

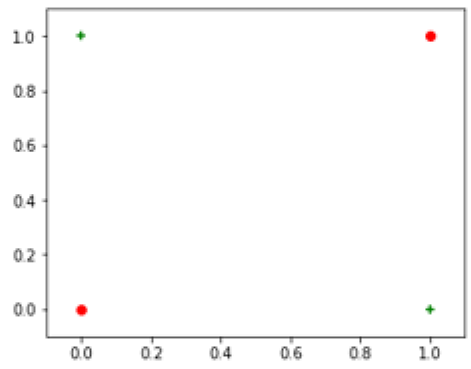
3a. My solution matches the table given:

```
Weight: [0. 1.]  Bias: 0.0  Point: [ 1 -2]   Value: 1
Weight: [ 1. -1.]  Bias: 1.0  Point: [0 3]   Value: 1
Weight: [1. 2.]  Bias: 2.0  Point: [ 1 -2]   Value: 1
Weight: [2. 0.]  Bias: 3.0

final w = [2. 0.], final b = 3.0
```

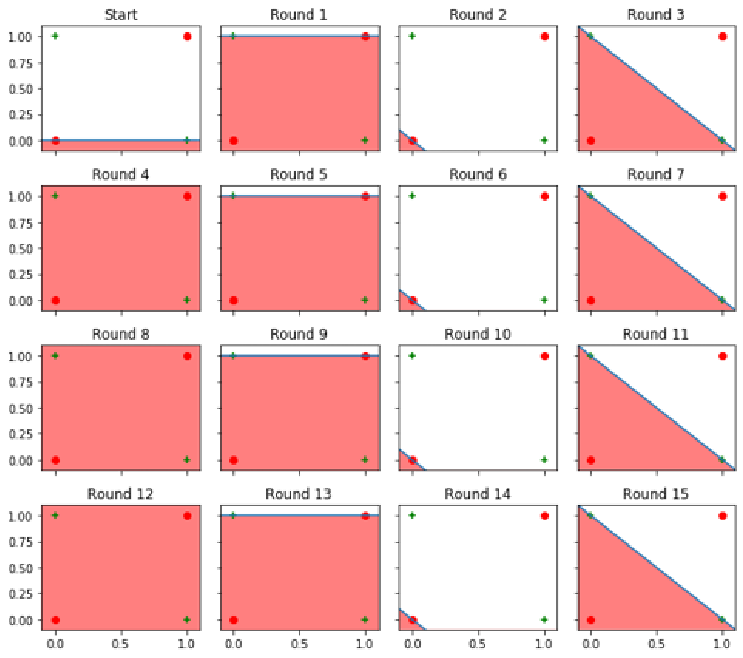


3b. In a 2D dataset, four points are in the smallest dataset that is not linearly separable. As shown in part 3c, if we create a square with opposite corners, there is no possible way to separate the data. The PLA will run in a continuous loop until stopped:



In a 3D dataset, five points are in the smallest dataset that is not planarly separable. If we take the situation above and create a pyramid where the base is not coplanar (shifting the depth of the base points), we create a situation where a single plane cannot separate the data correctly. Also, it makes sense that adding a dimension would just increase the number of points by one. For the N-dimensional set, N + 2 data points cannot be separated.

3c. If the dataset is not linearly separable, the PLA will never converge. For the PLA to converge, there must be a time where there are no misclassified points to update the weight vector. However, as shown by the graphs, there is always a case where there is a misclassified point in a set that is not linearly separable. The weight vector and bias term will loop endlessly unless stopped manually.



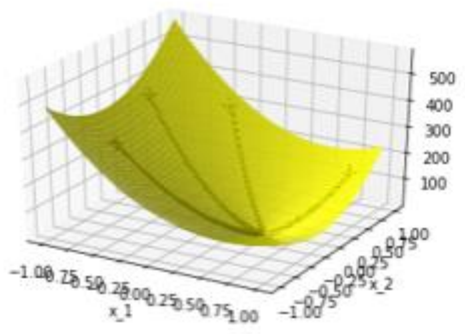
4a. We should define x and w to have a zeroth term that includes the bias. Since we will be multiplying these terms in the dot product, we need to put b in w and 1 in x. Thus, we have x = (1, x_1, x_2, …, x_d) and w = (b, w_1, w_2, …, w_d).

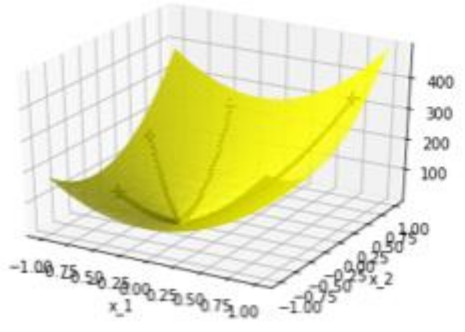
4b. Taking the gradient, we just use chain rule:

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

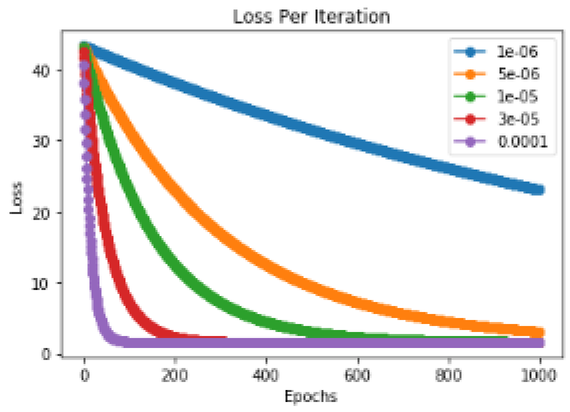
4c. Answer is just code implementation in Jupyter notebook.

4d. Different starting points did not change the result of the convergence, as all points converged to the same spot. However, the rate of convergence was faster for points farther away from the point of convergence. Both datasets 1 and 2 converge to their points of convergence with the general trends given.





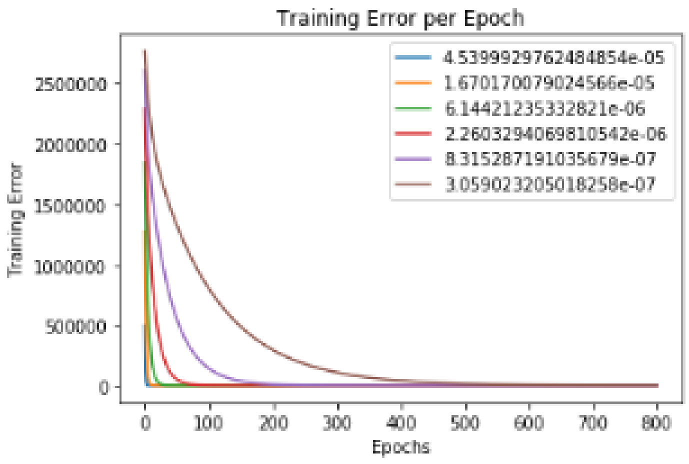
4e. As the step size gets bigger, the rate of convergence is a lot faster. As shown, the biggest step size converges to a respectable loss amount under 100 epochs while the smallest step size still has a lot of loss at 1000 epochs.



4f. The code in the notebook gave:

```
[ -0.22720591 -5.94229011 3.94369494 -11.72402388 8.78549375]
```

4g. Like part e, the largest step size converged the fastest, while the smallest step size converged the slowest. This also show that the starting weights were terrible as the starting training error for all lines is extremely high.



4h. Using the method for linear regression, I got

```
[ -0.31644251 -5.99157048 4.01509955 -11.93325972 8.99061096]
```

which is very close to what I got for the weights in SGD.

4i. It depends on the situation. When there are small amounts of data given in smaller dimensions, the closed form solution gives a fast, easy way to accurately calculate the weights that an SGD would inch towards for many iterations. However, if the data is given in large amounts and high complexities, the SGD is more efficient that the calculations needed given in a closed form solution.

4j. Since we observed that the convergence rate plateaus at a certain point, a stopping condition could be when the change in loss is below a certain threshold. This way the SGD will stop when the change in loss is basically zero and will not change anymore.

4k. The perceptron convergence behavior starts in a default position and changes to either side of the target function until it gradually converges from either side. The SGD convergence behavior starts on one side of the convergence target and slowly inches towards it.