1. A. A hypothesis set is a set of possible models to approximate the target function.

   B. Given input space $\mathcal{X}$, the hypothesis set of a linear model is a function of the form

   $$f(x) = w^T x + b \qquad \text{for } x \in \mathcal{X}$$

   For classification, we would use

   $$f(x) = \text{sign}(w^T x + b)$$

   C. Overfitting occurs when we choose a model of a sufficiently high complexity that reduces the in-sample error but increases the out-of-sample or test error.

   D. The two ways to prevent overfitting is to have more data points or to use a regularization term, or use a simpler model. Regularization tends to penalize the higher-order (more complex) terms to favor the simpler models.

   E. Training data is used to determine the parameter values of a model. Testing data is used to evaluate the performance of the model. If you change your information based on the test data, then you're effectively using test data as your training data, and your test data does not accurately reflect our-of-sample performance.

   F. We assume that our data is identically and independently sampled from the same distribution.

   G. The input space $\mathcal{X}$ could be the email address, the contents of the email using bagging of words, the title of the email (again using bagging of words). Y would be a binary classification of 'yes'-'no' for spam or not spam.

   H. The k-fold CV procedure is to split the data into k partitions. Then you train $k-1$ of the partitions and test on the one partition to get a test error on the ith partition E_i. You run through this so that every partition eventually becomes a test partition and the cross-validation error is

   $$E_{CV} = \frac{1}{N}\sum_{i=1}^{N} E_i.$$

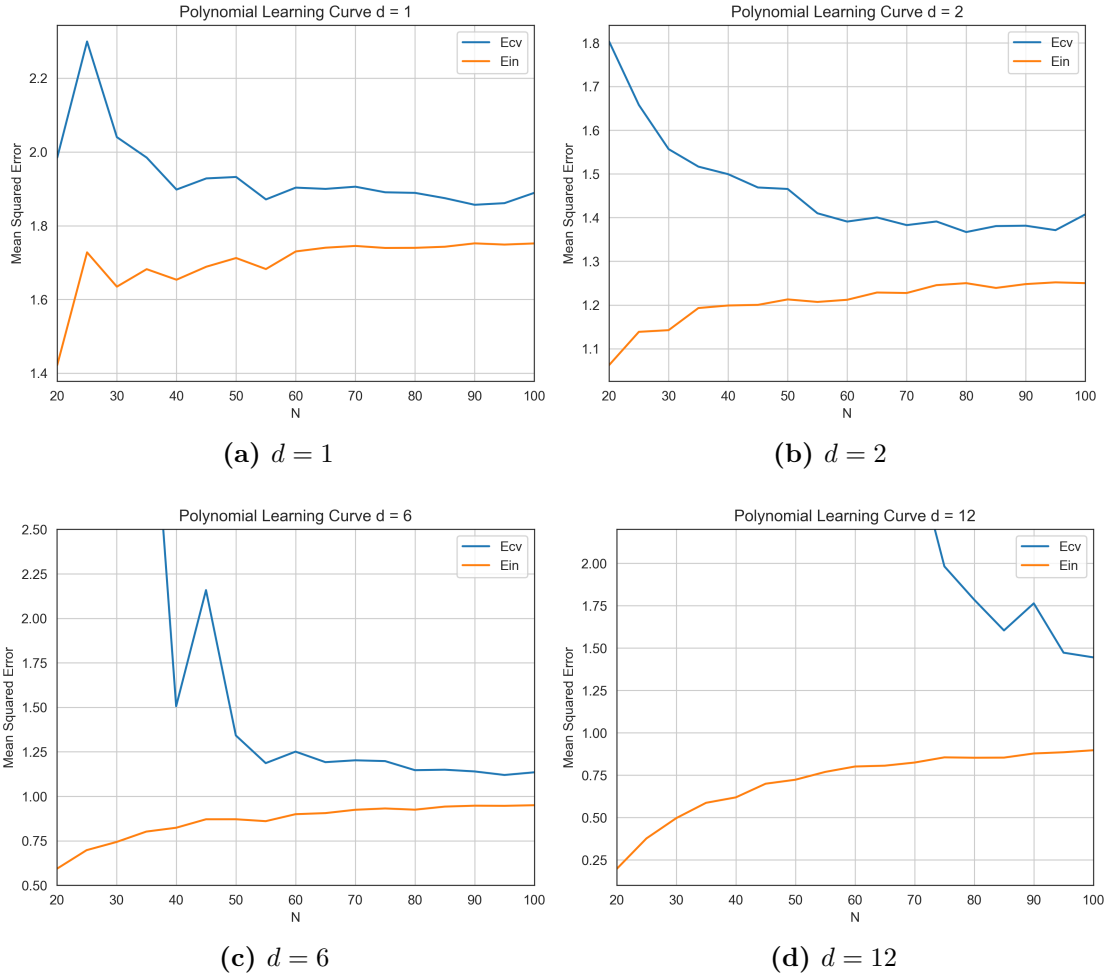
2. A.

$$\begin{aligned}
\mathbb{E}_S[E_{\text{out}}(f_S)] &= \mathbb{E}_S\left[\mathbb{E}_x\left[(f_S(x) - y(x))^2\right]\right] \\
&= \mathbb{E}_S\left[E_x\left[(f_S(x) - F(x) + F(x) - y(x))^2\right]\right] \\
&= \mathbb{E}_{S,x}\big[\,(f_S(x) - F(x))^2 + (F(x) - y(x))^2 \\
&\qquad + f_S(x)F(x) - F(x)^2 - f_S(x)y(x) + F(x)y(x)\big] \\
&= \mathbb{E}_x\left[\text{Bias}(x) + \text{Var}(x)\right]
\end{aligned}$$

   where the cross terms in the last step vanish to zero.

   B. The learning curves are shown in figure 1.

   C. $d = 1$ has the highest bias. The value in which our learning curves asymptotically approaches as $N \to \infty$ is the bias, which is greatest for $d = 1$.



(a) $d = 1$ **(b)** $d = 2$

(c) $d = 6$ **(d)** $d = 12$

Figure 1: Learning Curves for Polynomial Model

D. $d = 12$ has the highest variance. The distance between the bias and the out-of-sample error E_{out} is the variance.

E. The learning curve seems to show that the model wouldn't improve with additional points.

F. Training error is lower than validation error because the training error is the quantity which our learning algorithm is trying to minimize.

G. The $d = 6$ would perform best. It has a low bias and the discreptancy between the out-of-sample and in-sample performance shows that the parameter estimation is fairly consistent across samples.

3. A. The output file for the test is under the file name 'simpleperceptron.html'.

B. In 2D, we see in Figure 2 that the perceptron shatters 3 points but not 4. Algebraically, we have $N + 1$ points in N-dimensions

$$S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_{N+1}, y_{N+1})\}$$

such that for any y_i where $i = 1, 2, \dots N+1$ that we choose, we must find a $\mathbf{w}$ and b such that

$$\mathbf{w} \cdot \mathbf{x}_i + b = y_i \qquad i = 1, 2, \dots N+1.$$

We have N unknowns in $\mathbf{w}$ and one unknown in b, which leads to $N+1$ unknowns. This gives us the matrix

$$\tilde{X}\tilde{w} = Y$$

where

$$\tilde{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1N} & 1 \\ x_{21} & x_{22} & \cdots & x_{2N} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N+1,1} & x_{N+1,2} & \cdots & x_{N+1,N} & 1 \end{pmatrix}, \qquad \tilde{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ b \end{pmatrix}, \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N+1} \end{pmatrix}.$$

If we can find non-trivial solutions to this equation, then we can shatter the data points since we can choose whatever labelling for the y_i's. However, we know that $\tilde{X}$ is invertible, because if we set $Y = 0$, then this would imply that all $N+1$ points are on the same hyperplane, implying that $\tilde{w} = 0$. This implies that $\tilde{X}$ is invertible, meaning that we have a unique solution $\tilde{w}$ for every choice of Y.

For $N+2$ points, this may not be the case. If we add an extra row $\mathbf{x}_{N+2}, 1$ to $\tilde{X}$, then we are not guaranteed that $\tilde{X}$ is invertible because $\tilde{Y} = 0$ does not imply $\tilde{w} = 0$. Therefore, $N+1$ points are always separable in N-dimensions, but not $N+2$ points.

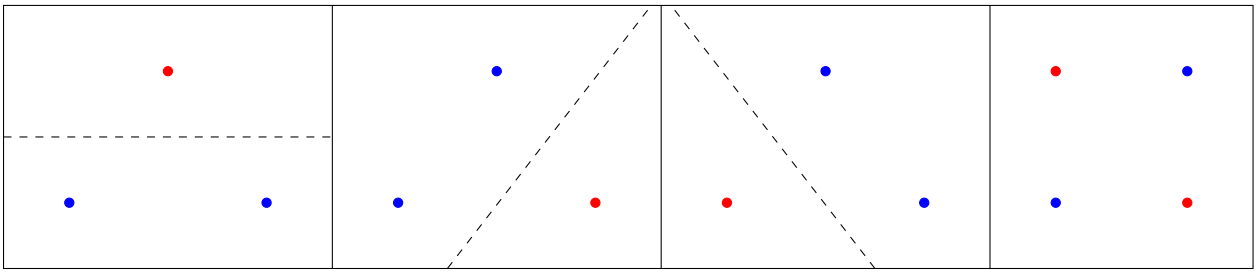


Figure 2: In two-dimensions, we see that in the case of three points, the data is always separable. However, the data can be non-separable in the case of four points.

C. The visualization is under the .html file "perceptronns.html". The algorithm never converges because there will always be one misclassified point. The PLA algorithm takes a misclassified point and modify the hyperplane until the misclassified point is correctly classified, but because there has to be at least one misclassified point, the PLA algorithm will never converge.

4. A. Just add a $x_0 = 1$ term. This way, $w_0 = b$ so that we can ignore the bias term and write

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}.$$

B. We have

$$\nabla_{\mathbf{w}} L(f) = -2 \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i$$

For SGD, we discard the summation and evaluate at a single point.

$$\nabla_{\mathbf{w}}^{\text{SGD}} L(f(\mathbf{x}_i, y_i)) = -2(y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i.$$

C. The plots are stored as gifs in the "animation$n$.gif" files.

D. In the first dataset, the SGD converges fairly evenly for the different w_0 points. In the second dataset, the SGD converges quicker for the w_0 points that are closer to the origin, i.e. $||w_0||$ implies faster convergence. This is simply due to the nature of the loss function for each dataset. In the first dataset, the further points had a steeper gradient so the different w_0 points had similar convergences while in the second dataset the closer points had a steeper gradient so the convergences differed.

E. The plot is shown in Figure 3. We see that if we make η too large ($\eta = 1$), then the w-updates are too large and never converge to the minimum-loss value, but instead just jumps around it, as shown in animation6.gif. When η becomes too large ($\eta = 10$), w actually diverges because the gradient of the loss function becomes very large. When there are multiple optimas, different w_0's can converge to different w_N's. So it is important to have a large enough η where we can randomly jump out of an optima, but small enough where it will converge.

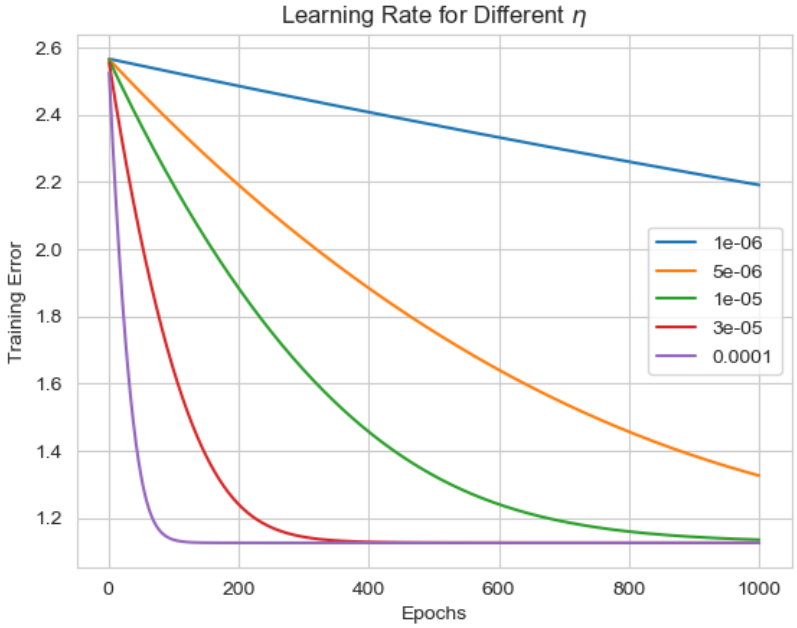


Figure 3: Learning Curve for Differing η.

F. For the larger dataset from the sgd_data.csv file with $\eta = e^{-15}$, we get

$$w = [-0.22788371, -5.97853503, 3.98839083, -11.85700488, 8.91129436]$$

G. We get a learning curve as shown. We can see that the larger η results in faster convergence up to a certain limit. When the curves flatten out, this means that the weights have converged and any SGD update to the weights will just oscillate around the optimal value.

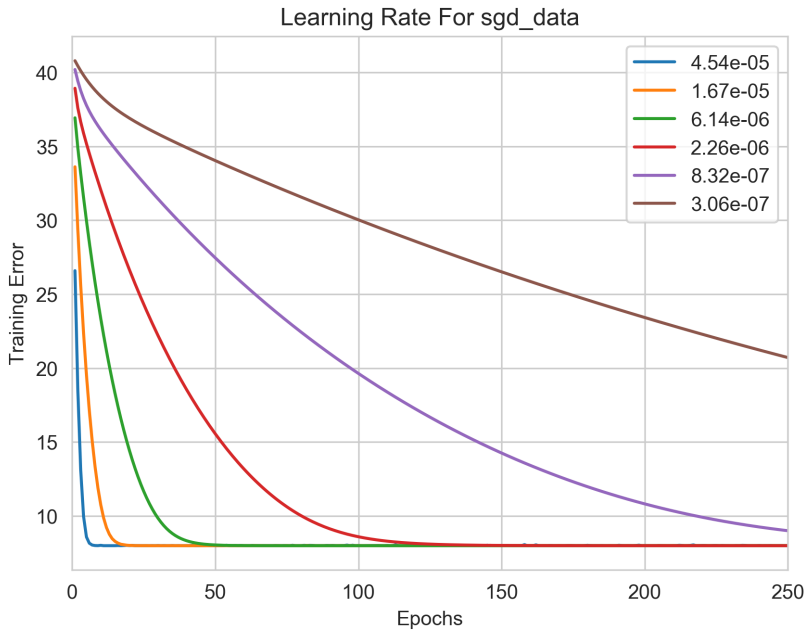


Figure 4: Learning Curves for Larger Dataset

H. The SGD value is

$$w_{\text{SGD}} = [-0.31646018, -5.97846778, 4.0156245, -11.94060225, 8.98594131]$$

and the analytical value is

$$w_{\text{analytical}} = [-0.31644251, -5.99157048, 4.01509955, -11.93325972, 8.99061096].$$

This is less than a 0.1 percent difference. More explicitly,

$$\frac{||w_{\text{SGD}} - w_{\text{analytical}}||}{||w_{\text{analytical}}||} < 0.001.$$

I. Yes, when the data is very large and we have online learning where the data keeps updating, we don't want to re-train the entire data set (matrix multiplication has awful computational times for large matrices). SGD is much more computationally friendly.

J. We can use

$$|L(w_i) - L(w_{i+1}| < \epsilon$$

where ϵ is some precision we use. We could also use

$$||w_i - w_{i+1}|| < \epsilon.$$

The latter is probably better because two separate weights could have similar losses.

K. The SGD algorithm will converge such that the weight vector converges to a specific value. The perceptron may not converge if the data is not separable. Thus, we would have to decide when to stop the PLA algorithm if it's not separable manually, whether that is checking for the lowest error.