ADMIN NOTE: TO SEE MY CODE YOU MUST BE LOGGED INTO A CALTECH.EDU GOOGLE ACCOUNT. I DID ALL MY WORK ON COLAB WITH MY CALTECH ACCOUNT AND AM UNABLE TO SHARE OUTSIDE OF CALTECH.

Problem 1: Basics

A)

The Hypothesis set is the set of possible candidate models aimed to approximate a target function. We take our hypotheses from this set.

B)

Solutions of the form $w^T x + b$

C)

Overfitting refers to when a model is trained too closely on a training data set, becoming too specific and exact on that training data set. It fails to discern noise in the training data, and as such is built in such a way that it is unable to predict additional data correctly and reliably. This can be recognized not only when we have a much higher training error than testing error, but also when we have high variance.

D)

i) One can use techniques such as cross-validation and regularization to combat overfitting.
ii) One can also increase the size of the training set used

E)

The training data is the data used to "train" our model- ie to adjust its relevant parameters so that it might predict the data we are interested in. We use the testing data to "test" our model- we see how well it does on this separate data set, with the parameters determined from the training data (we critically do not adjust the model parameters with test data). It is important to never change your model from information from the test data, because doing so may bias the model towards the training data, making the performance metrics determined from the test data biased. (For instance, we might not be able to determine if overfitting occurred now, since we may have overfitted onto the test data additionally.)

F)

i) We first assume our datapoints are independent
ii) We secondly assume our datapoints are identically distributed

G)

X could be a vector of relevant independent variables taken from the emails. As seen in lecture, we could have a bag of words representation of the email contents, we could also perhaps add discrete variables in our vector that represent the presence or absence of certain words, and we could perhaps have other variables such as email word count in our vector. Our output space Y is just [0,1]- a discrete binary variable which indicates whether the email is a spam email or not.

H)

In K-fold validation, we partition our (randomized) training set into K equal partitions. We first train on K-1 of these partitions and then we validate on the other remaining partition. We repeat this process a total of K times such that each partition is validated on once.

Problem 2: Bias Variance Tradeoff

https://colab.research.google.com/drive/1JzQA2PzoSXxc2BoDUExxsbnKFjX1cue2?usp=sharing

A)

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

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

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

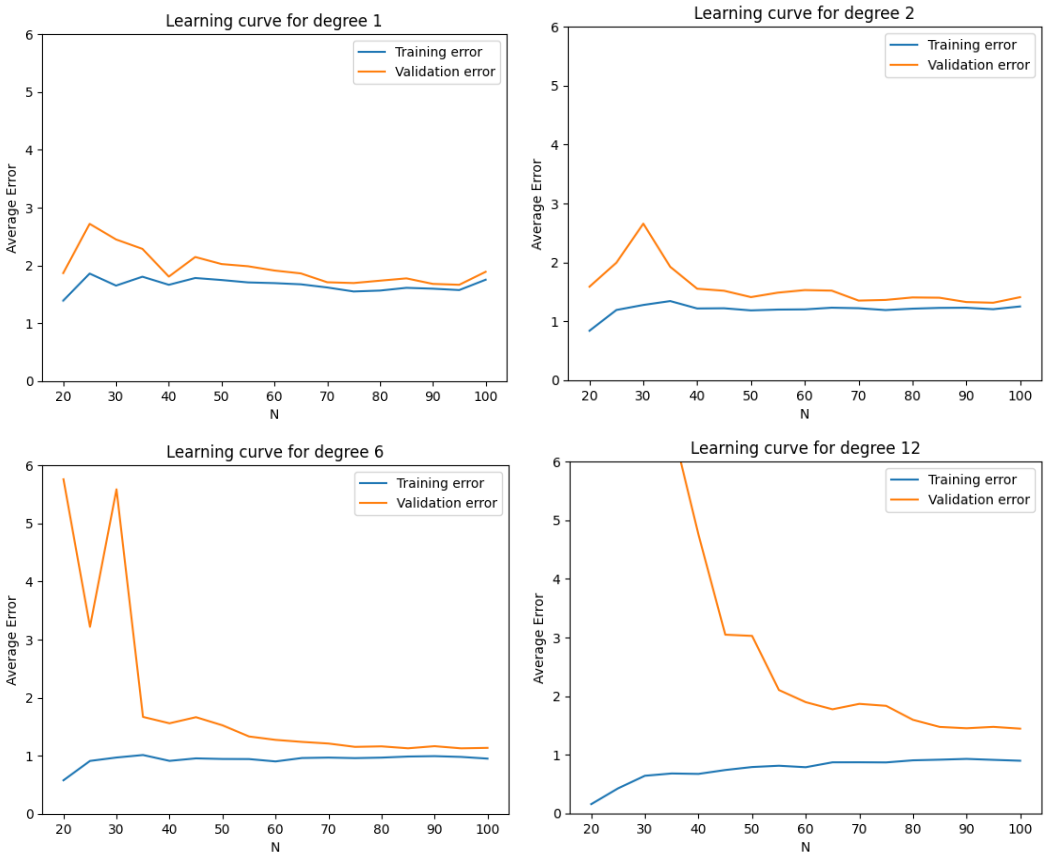
$$= E_x[E_S[(f_S(x)-F(x))^2+(F(x)-y(x))^2+2(f_S(x)-F(x))(F(x)-y(x))]]$$

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

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

$$= E_x[Var(x)+Bias(x)]$$

B)



C)

The degree 1 polynomial regression model has the highest bias. This is because the average mean squared error for both training and validation error tends to the highest value seen among all 4 models (~2). This higher error suggests higher bias which makes sense given the greater simplicity of this model (and thus its tendency to perhaps underfit).

D)

The degree 12 polynomial regression model has the highest variance. This is because at low N we see very high (highest) validation mean squared error- which reduces as we increase N. This implies that on the lower training data cases, the model strongly overfitted on the training data. We also see the highest discrepancy between training and validation error even upon convergence with respect to all the other models- also indicating over fitting and higher variance. (This makes sense given the greater complexity of this model, and thus its tendency to perhaps overfit).

E)

The learning curve indicates that additional training points will not be so useful for the quadratic model. We see a plateau in the learning curve at about 40 points, with not much

improvement in either validation or training error after this point, even as we increase to 100 datapoints.

F)

The model is trained and optimized on the training data. Training data does not convey all properties of the whole data distribution- so it will not encapsulate all of the independent validation data properties. We are thus unable to truly capture the whole nature of the real data, we optimize against a subset, the training data- and we are thus unable to capture the validation data perfectly, moreover we have not optimized on the validation data- so it makes sense that the validation errors are a bit worse than the training errors.

G)

I would expect the 6 degree model to perform best on unseen data drawn from the same distribution as the training data as it has the lowest observed validation error after about 50 data points- implying strong generalization.

Problem 3: Stochastic Gradient Descent

https://colab.research.google.com/drive/1VXcIypfSioZR07tfwv-OovdXJRIkqhR8?usp=sharing

https://colab.research.google.com/drive/1sW-iFQDvu2Z5xfYOdEaJSSLwfvgNHwkz?usp=sharing

A)

Suppose for a contradiction that for local minimum x', $\nabla f(x') \neq 0$.

By taylor's theorem, $\forall x, h \ \exists c \in (0,1)$ s.t. $f(x+h) = f(x) + \nabla f(x + c \cdot h)^T h$.

Let us choose $h = -\alpha \nabla f(x) \quad \alpha > 0$

As $h \to 0$ (by manipulating α) we notice

$\nabla f(x' + c \cdot h) \cdot \nabla f(x')$

$\to \nabla f(x') \cdot \nabla f(x') \geq \|\nabla f(x')\|^2 \geq \frac{1}{2} \|\nabla f(x')\|^2$

It is thus reasonable to say that as ∇f is continuous, we choose an h very small s.t.

$\nabla f(x' + c \cdot h) \cdot \nabla f(x') \geq \frac{1}{2} \|\nabla f(x')\|$

Now let us reconsider

$$f(x+h) = f(x) + \nabla f(x + c \cdot h)^T h.$$

$$f(x' + h) = f(x') - \nabla f(x' + c \cdot h)^T \alpha \nabla f(x)$$

$$= f(x') - \alpha \nabla f(x' + c \cdot h) \cdot f(x)$$

$$\leq f(x') - \frac{1}{2} ||\nabla f(x')||$$

Because we assumed $\nabla f(x') \neq 0$,

$||\nabla f(x')|| \neq 0$.

$\Rightarrow f(x' + h) < f(x')$ for some h.

A contradiction to the assumption that x' was a local minimum – as we have now found a real point smaller in value.

B)

We include an extra static term in x, a term that is always just 1. Analogously we include an extra term in w, which is just the bias term. As such we do include the bias when we multiply x and w.

C)

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

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

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

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

In gradient descent you find the average gradient of from all the datapoints and update based on this, whereas in stochastic gradient descent you choose one random data point to update on. This means one epoch is O(n) for gradient descent, vs O(1) in stochastic gradient descent. Hence, SGD is much less computationally costly.

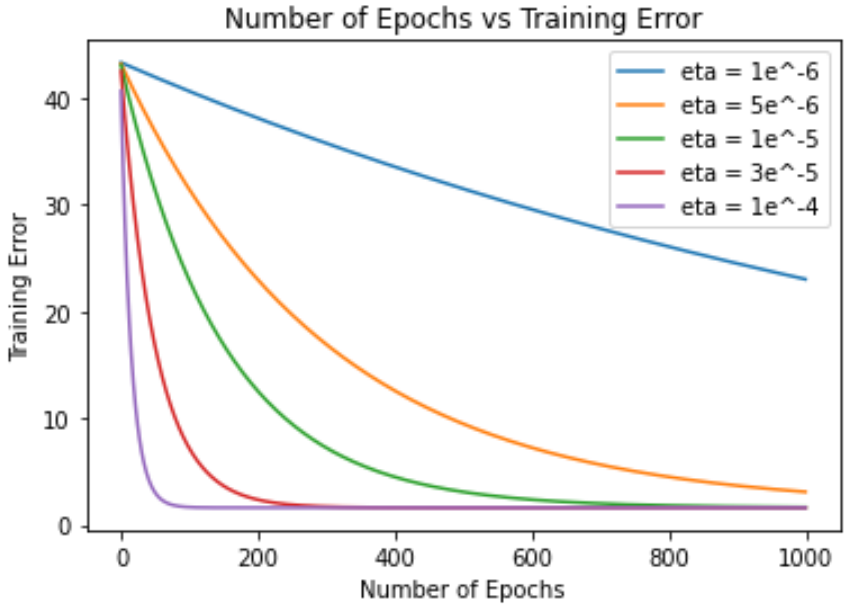
D)

See Code

E)

The convergence behavior is fairly consistent regardless of starting point: we always see a straight line convergence to the same global minimum. This is consistent between datasets 1 and 2.

F)

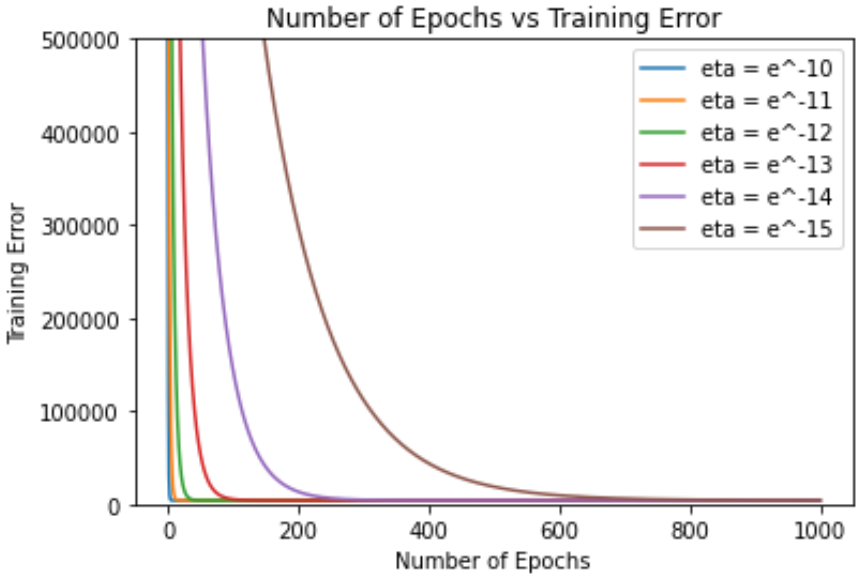


As eta grows larger, the system converges faster. Looks somewhat like exponential decay (higher eta faster decay).

G)

Weights: [-5.97852053, 3.98840727, -11.85698757, 8.91131382]
Bias: -0.22788582

H)



As eta grows larger, the system converges faster. (An explanation: small step sizes reduce convergence time, at each epoch we make less progress!).

I)

Got:

Weights: [-5.99157048, 4.01509955, -11.93325972, 8.99061096]
Bias: -0.31644251

This is not a perfect match (bias seems to have greatest discrepancy), but it is very close.

J)

Closed form solution is likely very computationally expensive for large datasets and lots of independent variables- this means SGD could be a good computationally cheaper alternative.

K)

We could use a stopping condition related to the change between training or validation error (or even the change in weights) between epochs. When this change is sufficiently small, we could stop the algorithm.

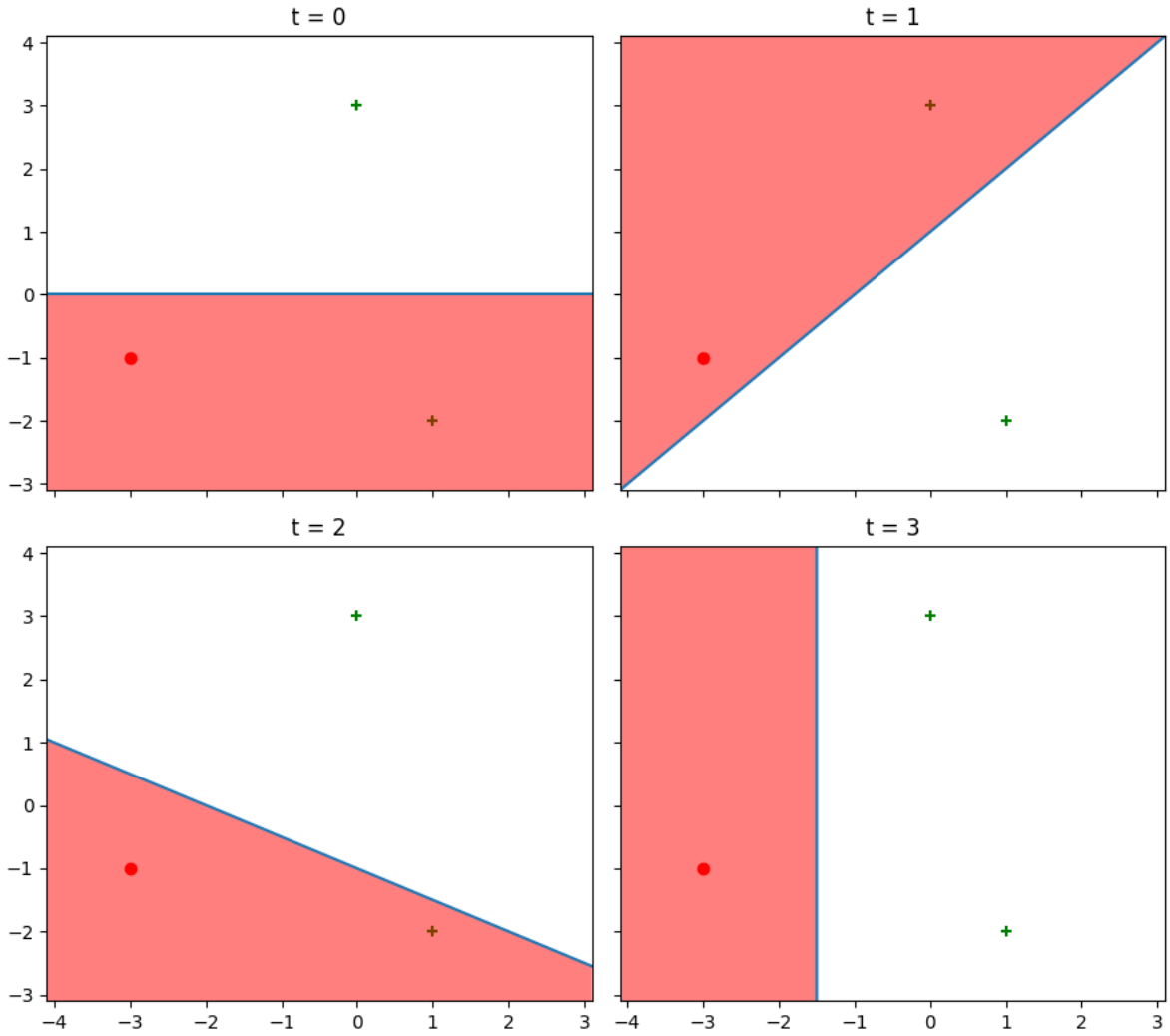
Problem 4: The Perceptron

https://colab.research.google.com/drive/1e4lpaalnbTXWElnH3eplv-JarftALd7t?usp=sharing

A)

t	b	w1	w2	x1	x2	y
0	0.0	0.0	1.0	1	-2	1
1	1.0	1.0	-1.0	0	3	1
2	2.0	1.0	2.0	1	-2	1
3	3.0	2.0	0.0			

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



B)

In the 2D case, the smallest data set that is not linearly separable it seems to me would be 4 datapoints. One could image a system where the line between 2 points of a type and another line of another 2 points of another type intersect. In this case there would be no way to separate the 2 point types.

IE:

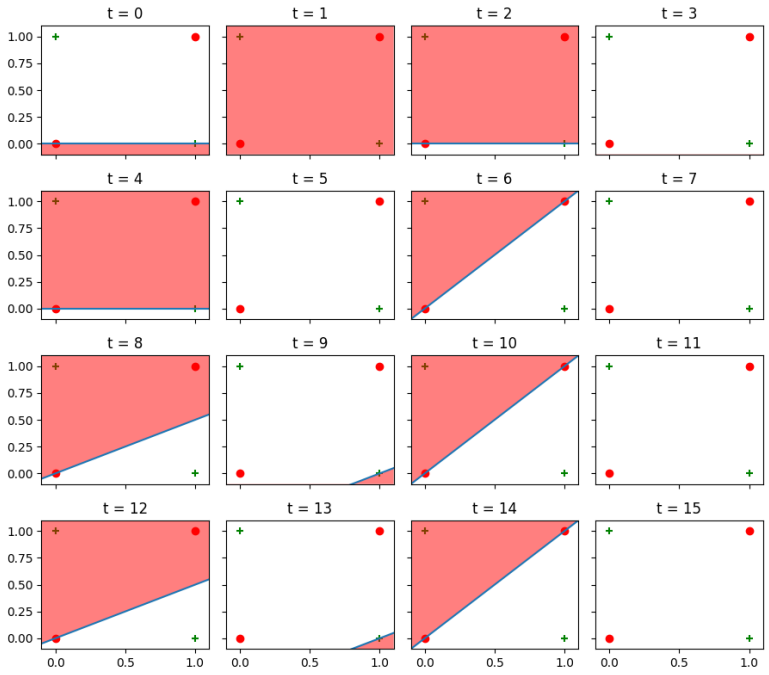
```
        +
x               x
        +
```

(line between +s and xs intersects, these cannot be separated)

In the 3D case, it would seem to me that the smallest value is 5 datapoints. Instead of having a lines between types intersect, I imagine we could have a plane (of 3 data points of the same type) intersect with a line between 2 datapoints of another type. In this case it would not be possible to linearly separate the types.

In general, it seems that in the N Dimensional case, the minimum size of a non-linearly separable data set is N+2.

C)



The Perceptron Learning Algorithm will never converge. This is because it is impossible to separate these points via a hyperplane for linearly inseparable points (by construction). Thus there will always exist some misclassified point, forcing the algorithm to keep going.

D)

In SGD we see a straight line, smooth, gradual convergence directly to the global minimum (at least for smaller eta step size). In the perceptron, we see a more jagged fluctuating convergence that is not so smooth and direct.