CS 5350/6350: Machine Learining Spring 2019

Homework 2

Handed out: 11 Feb, 2019
Due: 11:59pm, 25 Feb, 2019

Author: Cade Parkison

Note: Due to time constraints, I was unable to finish getting results for all of the programming parts of this assignment. My implementations made the testing very time consuming. For bagging and random forests, it took more than 3 hours each to train the classifiers according to the specifications. Because of this, I did not get the plots that were required. I do believe my implementations are mostly correct, and I hope that by looking at my code this will be noticable. I will work much harder on future assignments to improve the quality of my results.

1 Paper Problems [40 points + 6 bonus]

1. [5 points]

$$m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big)$$

   (a) [2 points]

      i. [1 point]
      An interpretation Occam's Razor principle states that given two explanations of the data, the simpler explanation is preferable. According to this, I would prefer hypothesis with the smaller hypothesis space size, H_2. This is in line with the goal of Machine learning, to find the simplest hypothesis that is consistent with the sampled data.

      ii. [1 point]
      If we assume fixed error and confidence parameters (δ and ϵ) for both learning algorithms, algorithm L_1 will require more samples (m_1) in order to produce the same error and confidence levels. This can be seen in the following inequalities:

$$m_1 > \frac{1}{\epsilon}\big(\log(|H_1|) + \log\frac{1}{\delta}\big)$$

$$m_2 > \frac{1}{\epsilon}\big(\log(|H_2|) + \log\frac{1}{\delta}\big)$$

      Using the inequality $|H_1| > |H_2|$, and since δ and ϵ are constant:

$$\frac{1}{\epsilon}\big(\log(|H_1|) + \log\frac{1}{\delta}\big) > \frac{1}{\epsilon}\big(\log(|H_2|) + \log\frac{1}{\delta}\big)$$
$$m_1 > m_2$$

This means that if both algorithms use the same number of training samples during the learning phase, algorithm L_2 will have higher accuracy $(1-\epsilon)$ and confidence $(1-\delta)$. I therefore prefer the hypothesis with the smaller hypothesis space, H_2.

(b) [3 points]
Using the given confidence and accuracy values, we have $\delta = 0.05$ and $\epsilon = 0.1$. Plugging in these as well as $|H_1| = 3^{10}$, we can plug them into the original inequality to solve for m_1.

$$\begin{aligned} m_1 &> 10\big(\log(3^{10}) + \log 20\big) \\ &> 60.72 \end{aligned}$$

Therefore, we need at least 61 training examples to get the hypothesis with desired performance in L_1.

2. [7 points]

(a) [1 points] Simple disjunctions out of n binary variables
The size of the hypothesis space of boolean disjuntions is 3^n. This is definitely PAC learnable, as seen in the previous problem with n=10.

(b) [1 points] m-of-n rules (Note that m is a fixed constant).

(c) [1 points] General conjunctions out of n binary variables ("General" means that negations are allowed to operator on each variable before the conjunction).
For purely conjunctive concepts, the hypotheses space is 3^n, since each attribute can be required to be true, false, or just ignored.

(d) [2 points] General Boolean functions of n binary variables.
For general boolean functions, the size of the hypothesis space is 2^{2^n}. As n increases, this would get exponentially harder to be PAC learnable, but still possible with enough training examples.

(e) [2 points] Can ID3 algorithm *efficiently* PAC learn the above concept classes?
ID3 would not be able to efficiently learn all these classes, due to the exponential nature of the their PAC functions. The most simple of the classes would be efficiently learnable, but not the more general cases.

3. [5 points] In our lecture about AdaBoost algorithm, we introduced the definition of weighted error in each round t,

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\big)$$

where $D_t(i)$ is the weight of i-th training example, and $h_t(x_i)$ is the prediction of the weak classifier learned round t. Note that both y_i and $h_t(x_i)$ belong to $\{1, -1\}$. Prove that equivalently,

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

Solution

Starting with the following definition of Z_t, I then take the derivative w.r.t. α_t and set equal to zero:

$$Z_t = \sum_{i:h_t(x_i)=y_i} D_t(i)e^{-\alpha_t} + \sum_{i:h_t(x_i)\neq y_i} D_t(i)e^{\alpha_t}$$

$$\frac{dZ_t}{d\alpha_t} = \sum_{i:h_t(x_i)=y_i} -D_t(i)e^{-\alpha_t} + \sum_{i:h_t(x_i)\neq y_i} D_t(i)e^{\alpha_t} = 0$$

$$\sum_{i:h_t(x_i)=y_i} D_t(i) = \sum_{i:h_t(x_i)\neq y_i} D_t(i)e^{2\alpha_t}$$

Then, by definition and the proof proof presented in class, the error can be written as:

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

4. [8 points]

   (a) [1 point] $f(x_1, x_2, x_3) = x_1 \vee x_2 \vee x_3$
   Starting with the generic hyperplane formula for three dimensions, we can solve for the bias and weights to fit the above boolean disjunction.

   $$b + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

   where $\mathbf{w} = [1, 1, 1]$ and $b = -1$. Putting this together gives the following linear classifier:

   $$x_1 + x_2 + x_3 \geq 1$$

   This means that $f(x_1, x_2, x_3) = 1$ iff the above inequality is true. The separating hyperplane is therefore:

   $$x_1 + x_2 + x_3 - 1 = 0$$

(b) [1 point] $f(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge \neg x_3$

Just as the problem above:

$$b + w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$$

where $\mathbf{w} = [1, -1, -1]$ and $b = -1$. Putting this together gives the following linear classifier:

$$x_1 - x_2 - x_3 \geq 1$$

The separating hyperplane is therefore:

$$x_1 - x_2 - x_3 - 1 = 0$$

(c) [1 point] $f(x_1, x_2, x_3) = \neg x_1 \vee \neg x_2 \vee \neg x_3$

Using De Morgan's law, the negation of a conjuntion is the disjuntion of the negations. We can therefore rewrite the above equation as:

$$\begin{aligned} f(x_1, x_2, x_3) &= \neg x_1 \vee \neg x_2 \vee \neg x_3 \\ &= \neg(x_1 \wedge x_2 \wedge x_3) \end{aligned}$$

Solving for the weights and bias gives the following:

$$b + w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$$

where $\mathbf{w} = [1, 1, 1]$ and $b = -3$. Putting this together gives the following linear classifier:

$$x_1 + x_2 + x_3 < 3$$

This means that $f(x_1, x_2, x_3) = 1$ iff the above inequality is true. The separating hyperplane is therefore:

$$x_1 + x_2 + x_3 - 3 = 0$$

Comparing $f(x_1, x_2, x_3) = \neg(x_1 \wedge x_2 \wedge x_3)$ and $f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3)$, the only difference is in this case we use a $<$ instead of $\geq$.

(d) [2 points] $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$

(e) [2 points] $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$

(f) [1 point] What do you conclude about the expressiveness of decision trees and linear classifiers? Why?

Decision trees are able to represent any boolean functions, while linear classifiers can only represent certain subsets of boolean functions that are linearly separable.

5. [6 points] The following boolean functions cannot be represented by linear classifiers. Can you work out some feature mapping such that, after mapping all the inputs of these functions into a higher dimensional space, you can easily identify a hyperplane that separates the inputs with different corresponding boolean function values? Please write down the separating hyperplane as well.

(a) $f(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

(b) $f(x_1, x_2) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

(c) $f(x_1, x_2, x_3)$ is listed in the following table

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
1	0	0	1
0	1	1	0
1	1	0	0
1	0	1	0
1	1	1	1

6. [**Bonus**] [6 points] Given two vectors $\mathbf{x} = [x_1, x_2]$ and $\mathbf{y} = [y_1, y_2]$, find a feature mapping $\phi(\cdot)$ for each of the following functions, such that the function is equal to the inner product between the mapped feature vectors, $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$. For example, $(\mathbf{x}^\top \mathbf{y})^0 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1]$ and $\phi(\mathbf{y}) = [1]$; $(\mathbf{x}^\top \mathbf{y})^1 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = \mathbf{x}$ and $\phi(\mathbf{y}) = \mathbf{y}$.

(a) [2 points] $(\mathbf{x}^\top \mathbf{y})^2$

(b) [2 points] $(\mathbf{x}^\top \mathbf{y})^3$

(c) [2 points] $(\mathbf{x}^\top \mathbf{y})^k$ where k is any positive integer.

7. [9 points] Suppose we have the training data shown in Table 1, from which we want to learn a linear regression model, parameterized by a weight vector $\mathbf{w}$ and a bias parameter b.

(a) [1 point] Write down the LMS (least mean square) cost function $J(\mathbf{w}, b)$.

Starting with basic cost function and expanding we get:

x_1	x_2	x_3	y
1	-1	2	1
1	1	3	4
-1	1	0	-1
1	2	-4	-2
3	-1	-1	0

Table 1: Linear regression training data.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$\begin{aligned} J(\mathbf{w}, b) &= \frac{1}{2}\big((y_1 - (w_1x_{11} + w_2x_{12} + w_3x_{13} + b))^2 + (y_2 - (w_1x_{21} + w_2x_{22} + w_3x_{23} + b))^2 + ...\big) \\ &= \frac{1}{2}\big((1 - (w_1 - w_2 + 2w_3 + b))^2 + (4 - (w_1 + w_2 + 3w_3 + b))^2 + (-1 - (-w_1 + w_2 + b)) \end{aligned}$$

(b) [3 points] Calculate the gradient $\frac{\nabla J}{\nabla \mathbf{w}}$ and $\frac{\nabla J}{\nabla b}$

i. when $\mathbf{w} = [0, 0, 0]^\top$ and $b = 0$;

$$\begin{aligned} \frac{\delta J}{\delta b} &= (1-0) + (4-0) + (-1-0) + (-2-0) + (0-0) \\ &= 1 + 4 - 1 - 2 \\ &= 2 \end{aligned}$$

$$\begin{aligned} \frac{\delta J}{\delta w_1} &= (1-0)(1) + (4-0)(1) + (-1-0)(-1) + (-2-0)(1) + (0-0)(3) \\ &= 1 + 4 + 1 - 2 \\ &= 4 \end{aligned}$$

$$\begin{aligned} \frac{\delta J}{\delta w_2} &= (1-0)(-1) + (4-0)(1) + (-1-0)(1) + (-2-0)(2) + (0-0)(-1) \\ &= -1 + 4 - 1 - 4 \\ &= -2 \end{aligned}$$

$$\begin{aligned} \frac{\delta J}{\delta w_3} &= (1-0)(2) + (4-0)(3) + (-1-0)(0) + (-2-0)(-4) + (0-0)(-1) \\ &= 2 + 12 + 0 - 8 \\ &= 6 \end{aligned}$$

ii. when $\mathbf{w} = [-1, 1, -1]^\top$ and $b = -1$;
Using the same formulas as above, I got the following results:

$$\frac{\delta J}{\delta b} = 10$$

$$\frac{\delta J}{\delta w_1} = 22$$

$$\frac{\delta J}{\delta w_2} = -16$$

$$\frac{\delta J}{\delta w_3} = 56$$

iii. when $\mathbf{w} = [1/2, -1/2, 1/2]^\top$ and $b = 1$..

$$\frac{\delta J}{\delta b} = -4.5$$

$$\frac{\delta J}{\delta w_1} = -7.5$$

$$\frac{\delta J}{\delta w_2} = 4.0$$

$$\frac{\delta J}{\delta w_3} = 5.0$$

(c) [2 points] What are the optimal **w** and **b** that minimize the cost function?
Using $A^t Ax = A^t b$ where A is the features and b is the outcomes, solving for x (weights), I got the following:

$$b = -1$$

$$\mathbf{w} = [1, 1, 1]$$

(d) [3 points] Now, we want to use stochastic gradient descent to minimize $J(\mathbf{w}, b)$. We initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$. We set the learning rate $r = 0.1$ and sequentially go through the 5 training examples. Please list the stochastic gradient in each step and the updated **w** and b.

2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of decision trees in HW1 to your GitHub repository. Remember last time you created a folder "Decision Tree". You can commit your code into that folder. Please also supplement README.md with concise descriptions about how to use your code to

learn decision trees (how to call the command, set the parameters, etc). Please create two folders "Ensemble Learning" and "Linear Regression" in the same level as the folder "Decision Tree".

2. [36 points] We will implement the boosting and bagging algorithms based on decision trees. Let us test them on the bank marketing dataset in HW1 (bank.zip in Canvas). We use the same approach to convert the numerical features into binary ones. That is, we choose the media (NOT the average) of the attribute values (in the training set) as the threshold, and examine if the feature is bigger (or less) than the threshold. For simplicity, we treat "unknown" as a particular attribute value, and hence we do not have any missing attributes for both training and test.

   (a) [8 points]

   Adaboost greatly improves on the results of a single fully expanded decision tree by averaging the results of many decision stump models. In my implementation, I ran into the problem where after 2 iterations, all of the errors were stuck at a constant value of around 0.002. I believe this had to do with the way I implemented the alpha and epsilon calculations. The epsilon values started over 0.5 and converged to 0.5, which caused my alphas to go to zero. When all the alphas were zero, the weights never changed and the errors remained constant. If I had more time on the assignment, I believe this issue could be fixed.

   (b) [8 points] Based on your code of the decision tree learning algorithm (with information gain), implement a Bagged trees learning algorithm. Note that each tree should be fully expanded — no early stopping or post pruning. Vary the number of trees from 1 to 1000, report how the training and test errors vary along with the tree number in a figure. Overall, are bagged trees better than a single tree? Are bagged trees better than Adaboost?

   For the bagged trees implementation, my algorithm took multiple hours to run to the desired 1000 trees. Because of this, I was unable to get a good plot of errors vs trees. But, when testing on a small number of trees, the result was a decrease in error over time. This appeared to be better than a single tree. As expected, this method decreases the variance of the predictions. Compared to Adaboost, this algorithm performed worse, but both algorithms were better than a single decision tree. Overall, I think boosting would preform better in most cases.

   (c) [6 points]

   Unfortunately, due to time constraints I was unable to complete this part of the assignment.

   (d) [8 points] Implement the random forest algorithm as we discussed in our lecture. Vary the number of random trees from 1 to 1000. Note that you need to modify your tree learning algorithm to randomly select a subset of features before each split. Then use the information gain to select the best feature to split. Vary the size of the feature subset from $\{2, 4, 6\}$. Note that if your feature subset happen to be all the features used before, simply resample the subset. Report in a figure how the training and test errors vary along with the number of random trees for each feature subset size setting. How does the performance compare with bagged trees?

(e) [6 points] Following (c), estimate the bias and variance terms, and the squared error for a single random tree and the whole forest. Comparing with the bagged trees, what do you observe? What can you conclude?

3. [**Bonus**][10 points] In practice, to confirm the performance of your algorithm, you need to find multiple datasets for test (rather than one). You need to extract and process data by yourself. Now please use the credit default dataset in UCI repository https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Randomly choose 24000 examples for training and the remaining 6000 for test. Feel free to deal with continuous features. Run bagged trees, random forest, and Adaboost with decision stumps algorithms for 1000 iterations. Report in a figure how the training and test errors vary along with the number of iterations, as compared with a fully expanded single decision tree. Are the results consistent with the results you obtained from the bank dataset?

4. [22 points] We will implement the LMS method for a linear regression task. The dataset is from UCI repository (`https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test`). The task is to predict the real-valued SLUMP of the concrete, with 7 features. The features and output are listed in the file "concrete/data-desc.txt". The training data are stored in the file "concrete/train.csv", consisting of 53 examples. The test data are stored in "concrete/test.csv", and comprise of 50 examples. In both the training and testing datasets, feature values and outputs are separated by commas.

   (a) [8 points]

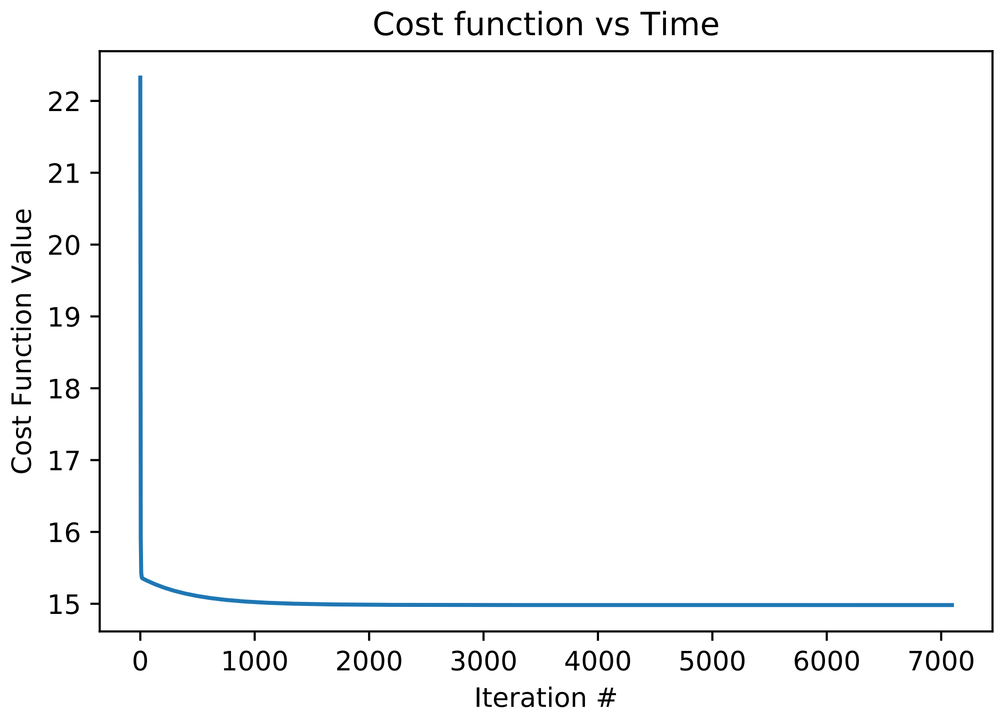
   When implementing Batch Gradient Descent, I started with learning rate of 1 and slowly decreased it until the algorithm converged. I found this convergence happened around a learning rate of $r = 0.01$. At this rate, my algorithm starts to converge very quickly, at about 1000 iterations. I also found that by initializing the weight vector to 1 instead of 0, the algorithm converged much faster.

   The learned weight vector is the following:

$$\mathbf{w} = [-0.01520, 0.90022, 0.78594, 0.8506, 1.29862, 0.12983, 1.5717, 0.99834]$$

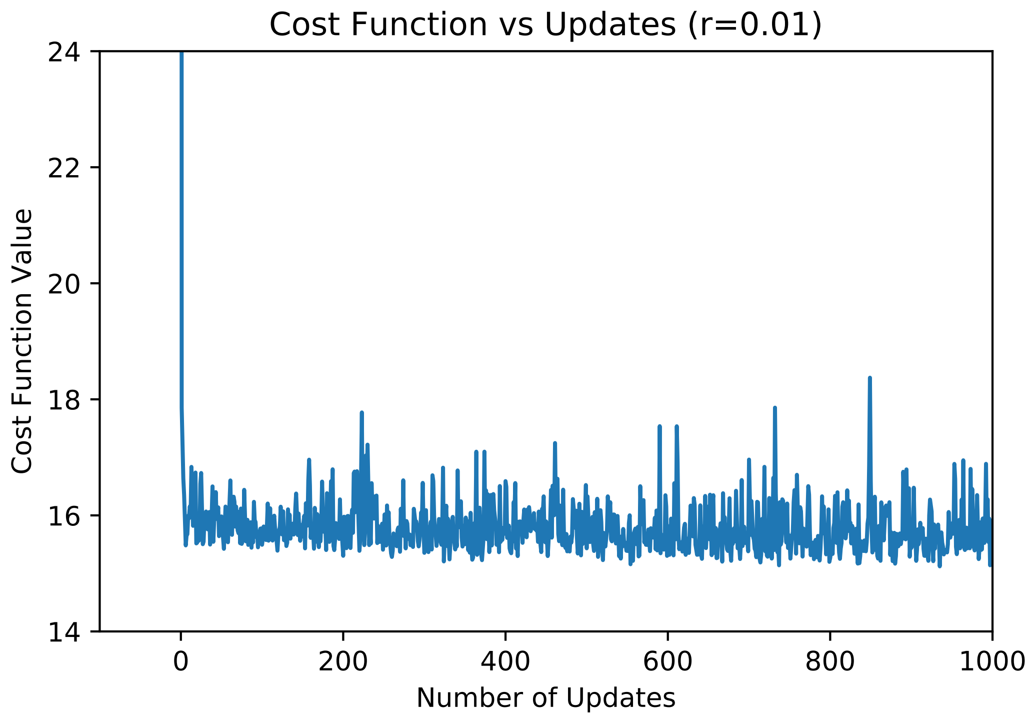
   Using this weight vector to calculate the cost function value of the test data, I get a function value of 23.361.

   The figure below shows the convergence of the cost function vs time.

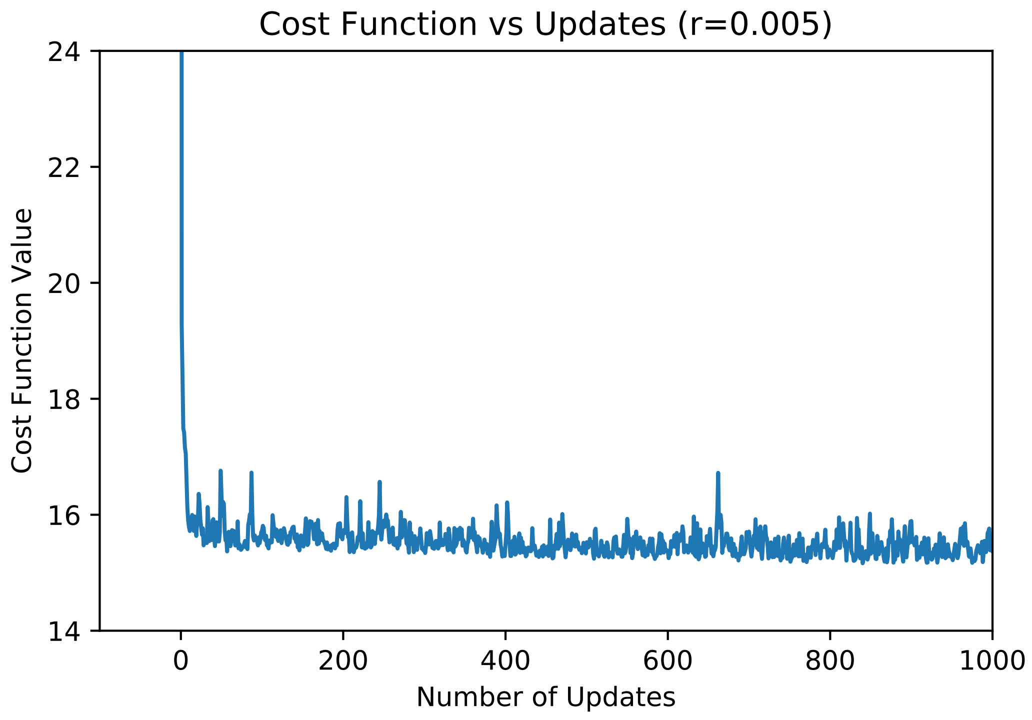


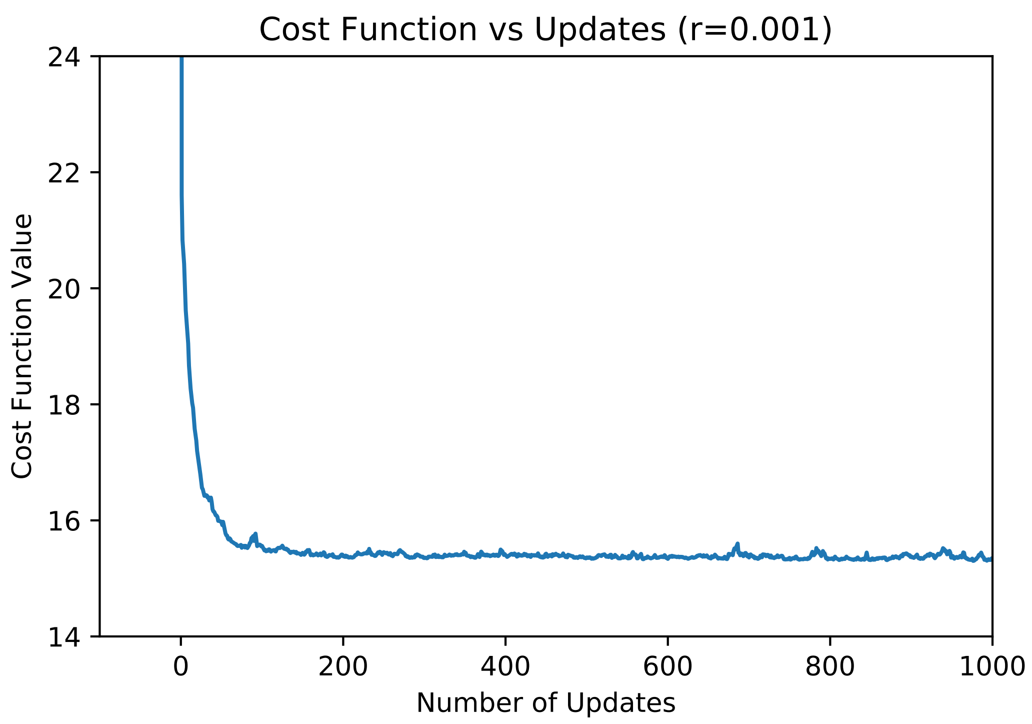
(b) [8 points] Implement the stochastic gradient descent (SGD) algorithm. You can initialize your weight vector to be $\mathbf{0}$. Each step, you randomly sample a training example, and then calculate the stochastic gradient to update the weight vector. Tune the learning rate r to ensure your SGD converges. To check convergence, you can calculate the cost function of the training data after each stochastic gradient update, and draw a figure showing how the cost function values vary along with the number of updates. At the beginning, your curve will oscillate a lot. However, with an appropriate r, as more and more updates are finished, you will see the cost function tends to converge. Please report the learned weight vector, and the learning rate you chose, and the cost function value of the test data with your learned weight vector.

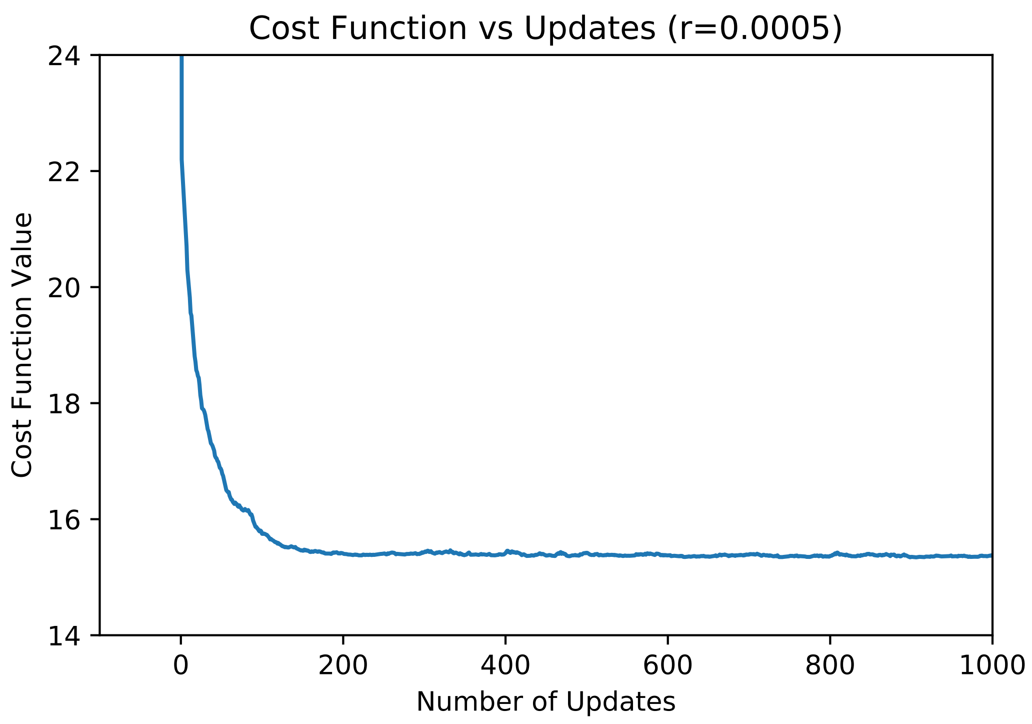
For stochastic gradient descent, I started with the learning rate used in batch gradient descent, $r = 0.01$, and calculated the cost function vs the number of updates. With this learning rate, the cost fluctuated a lot. This can be seen in the figure below.

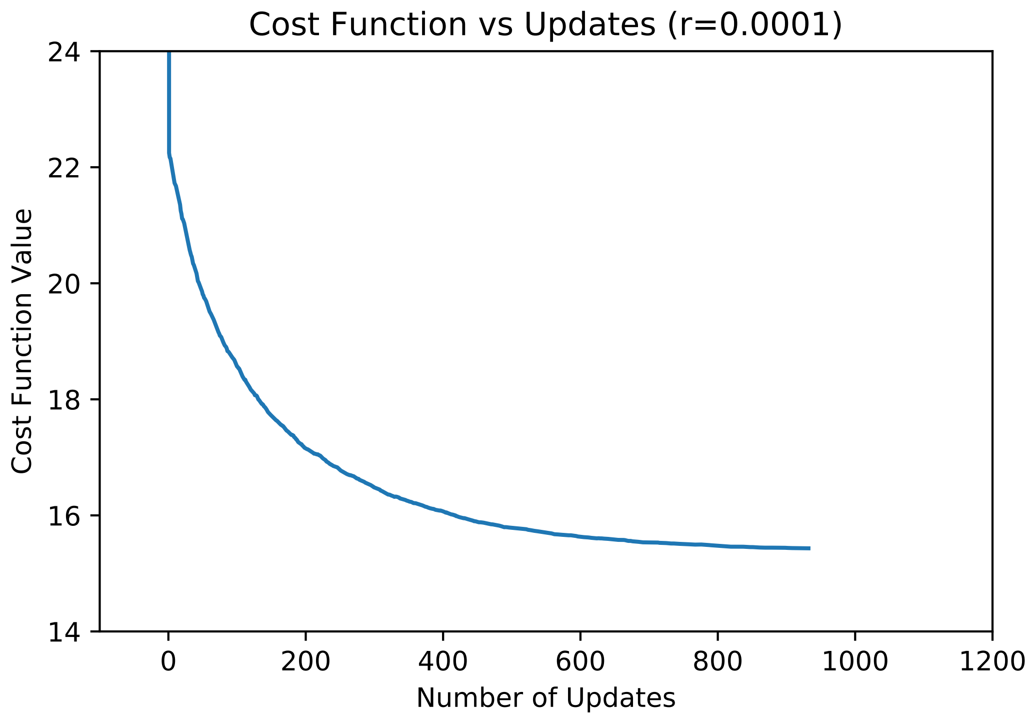


Next, I started to lower the learning rate and compare the plots. Below are the plots for learning rates of 0.005, 0.001, 0.0005, and 0.0001. As the learning rates decrease, the number of updates required for convergence increases. Also, the plots get much smoother, as can be seen below.




Cost Function vs Updates (r=0.001)
Cost Function Value
Number of Updates
24
22
20
18
16
14
0
200
400
600
800
1000


Cost Function vs Updates (r=0.0005)
Cost Function Value
Number of Updates
24
22
20
18
16
14
0
200
400
600
800
1000



From the plots above, it appears a learning rate between 0.0005 and 0.0001 will provide a constant steady state value while also converging quickly.

For the learning rate of 0.0001, the weight vector calculated is the following:

$$\mathbf{w} = [-0.03712, -0.05441, -0.22296, -0.2290, 0.45588, -0.03449, 0.20263, -0.01664]$$

Using this weight vector, I calculated the cost function of the test data to be $f = 22.415$, which is fairly close to the cost value using batch gradient descent.

(c) [6 points] We have discussed how to calculate the optimal weight vector with an analytical form. Please calculate the optimal weight vector in this way. Comparing with the weight vectors learned by batch gradient descent and stochastic gradient descent, what can you conclude? Why?

I calculated the analytical form of the weight vector with the following equation:

$$A^t Ax = A^t b$$

where x is the weight vector, A is the feature matrix, and b is the output vector. Solving for x (w), the resulting weights are:

$$\mathbf{w} = [0.03833, -1.05154, -1.38454, -1.0843, -0.34476, -0.24711, -1.36924, -0.65715]$$

Calculating the cost of the training data gives $f = 25.75$. Using this weight vector to calculate cost of the test data, I get $f = 15.4873$. This value is very close to the converging cost in both BGD and SGD algorithms. Comparing the three weight vectors, all weights are very different. Even though the various weights are different, the calculated costs seem to be similar. In conclusion, I can say that both batch gradient descent and stochastic gradient descent are good at fitting to the training data, but not great at generalizing.