**Homework #2 Programming Assignment**

CSCI567, Spring 2019  
Victor Adamchik  
**Due: 11:59 pm, Feb. 08, 2019**

**Before you start:**

There is a known issue with vocareum that caused by vocareum website, not us. If you submit your homework twice in a row without refresh the page, the grade book score might not be updated. So I suggest you to refresh your page before doing another submission.  
Also, this project have two part. You need to submit each part seperately on vocareum.

**Office Hour for Project Assignment 2**

Question for Part 1, we suggest you go to: 2nd Floor-201A: 1:00am - 3:00am, Thursday, January 31, 2019 for Yang, Fang  
Question for Part 2, we suggest you go to: 2nd Floor-201A: 3:00am - 5:00am, Thursday, January 31, 2019 for Cheng-Ju, Lin  
We will also hold office hour for the first week of February, we will announce the time shortly.  
  
Also, you can post your question on Piazza under pa-2 folder. We will try our best to answer all questions as soon as possible. Everyone in the class is welcomed to answer these questions. We really appreciate that. Please make sure you read previous posts before creating a new post in case your question has been answered before. We prefer Piazza than emails. However, if you have any urgent issue, feel free to send an email to both of us.   
Yang, Fang: yangfang@usc.edu  
Cheng-Ju, Lin: chengjul@usc.edu

**Problem 1 Regression (40 points)**

For this Assignment you are asked to implement linear regression. You are asked to implement 6 python functions for linear regression in linear\_regression.py. The input and output of the functions are specified in linear\_regression.py. Note that we have already appended the column of 1’s to the feature matrix, so that you do not need to modify the data yourself. We have provide a linear\_regression\_test.py for you to run some of your code and do some testing. Read the code, run it and check the result during your implementation.   
Submission: All you need to submit is linear\_regression.py

**Q1.1 Mean Square Error (5 points)**

First, You need to implement the mean sqaure error. This is very similar to the RSS you learned from the lecture note page 15 except you need take the arithmetic mean of the error. numpy already have a mean function for that, feel free to use it. There should be simple as one or two lines of code and there is only one thing you need to be careful: please check the dimension of the input w, X and y before you do matrix dot operation. You might need to take a transpose of some matrix in order to get the right shape.  
Report the mean square error of the model on the give n test set, no rounding for the result. The type of the error is , you can do type(err) to check yours.

* (5 points) TODO 1 You need to complete def mean\_square\_error(w, X, y) in linear\_regression.py

**Q1.2 Linear Regression (5 points)**

Now, you need to train your model. In linear regression model, it means find the weight for each feature. Check the lecture note page 21 and find the expression of general least square solution. Again, be careful with your matrix shape. Also, use numpy inverse funciton so you don't need to create your own. The shape of your return w should be (12,) if you train your model with the data we provide you  
Implement linear regression with no regularization and return the model parameters. Again, the implementation should be as simple as one or two lines of code. Use numpy instead of crazy nested for loop in your implementation. You don't need to worry about non-invertible matrix. We will take care of it in Q1.3.

* (5 points) TODO 2 You need to complete def linear\_regression\_noreg(X, y) in linear\_regression.py

Once you finish Q1.1 and Q1.2, you should be able to run your linear\_regression\_test.py. Read the output, check your dimension of w, and MSE for training, evaluation and testing dataset. The MSE for all of them should between 0.5~0.6. Otherewise, there must be something wrong with your implementation.

**Q1.3 Handle Non-invertible Matrix (5 points)**

There are cases that during the calculation, the matrix are non-invertible. We manually created that situation in the data\_loader.py line 40. We simply manully set one row and one column in the dataset to be zero. Thus, we will get similar result as letcure note page 29. Now, you need to implement the solution on lecture note page 31. Here is the rul: in this assignment, if the smallest absolute value of all eigenvalue of a matrix is smaller than 10−510−5, the matrix is non-invertible. If the matrix is non-invertible, keep adding 10−1∗I10−1∗I until it is invertible. You can use numpy functions to get eigen value for a matrix, get the absolute value ,find the minimum and create identity matrix, just search for them.

* (10 points) TODO 3 You need to complete def linear\_regression\_invertible(X, y) in linear\_regression.py

Once you finish Q1.3, run linear\_regression\_test.py again, the MSE should be between 0.5~0.6. Otherewise, there must be something wrong with your implementation.

**Q1.4 Regularized Linear Regression (5 points)**

To prevent overfitting, we usually add regularization. For now, we will focus on L2 regularization, same as the one in lecture note page 50. Implement the regularized\_linear\_regression(X,y,lambda)

* (10 points) TODO 4 You need to complete def regularized\_linear\_regression(X, y, lambd) in linear\_regression.py

Once you finish Q1.4, run linear\_regression\_test.py again. Compare your result with Q1.3, did you find something interesting? Think about why or check the statement in red on lecture note page 50.

**Q1.5 Tune the Regularized Linear Hypter\_Parameter (10 points)**

Use your implementation from Q1.4, the regularized\_linear\_regression(X, y, lambd), try different lambds and find the best lambd to minimize the MSE. Use Xval and yval to tune your hyper-parameter lambda  
tune\_lambda(Xtrain, ytrain, Xval, yval, lambds). To help us grading, set your lambd as all power of 10 from 10−1910−19 to 10191019 . However, in real world, no one knows the best range of hyper-parameters like lambda.

* (10 points) TODO 5 You need to complete def tune\_lambda(Xtrain, ytrain, Xval, yval) in linear\_regression.py Once you finish Q1.5, run linear\_regression\_test.py again. You should find a lambda that is 1e−141e−14. Otherewise, there must be something wrong with your implementation.

**Q1.6 Polynomial Regression (10 points)**

We would like to introduce your another regression called polynomial regression. You can check the detail on wiki: <https://en.wikipedia.org/wiki/Polynomial_regression> However, you don't need to check that to finish the homework  
Apply polinomial regression on the data, first try the second order function: y=w1∗X+w2∗X2y=w1∗X+w2∗X2 In this case, you should change the data [X][X] to be [XX2][XX2]  
Implement the mapping\_data(x, power) function. For the higher order function, Eg.y=w1∗X+w2∗X2+w3∗X3y=w1∗X+w2∗X2+w3∗X3, thus your the data [X][X] to be [XX2X3...][XX2X3...]. Be careful here, the new X is still 2D matrix, of size(N, D\*power). Use the insert function from numpy to quickly add data to original dataset. Obeserve how the mean square error change.

* TODO 6 You need to complete def mapping\_data(X, power) in linear\_regression.py

Once you finish Q1.6, you can run linear\_regression\_test.py, it will change the data with your mapping function, find corresponding w with , make prediction and report the mean square error again.  
You will see that your MSE keep increasing while the power goes up. Think about why this happen.

This is end of part 1. Hopefully everything is clear and you didn't struggle too much. Now you should be able to take your next challenge, Part 2.

**Grading**

Your code will be graded on Vocareum with autograding script. For your reference, with my implementation the testing process is no longer than 15s including generating the grade report, and the processing time shows on the terminal is less than 0.2s. As long as you code can finish grading on Vocareum, you should be good.  
I can't provide any test case we used on the auto-grading script to you. However, I tried to help you indentify your problem with the printing statement. Please read them carefully. 

**Problem 2 Classification (60 points)**

For this assignment you are asked to implement both binary and multiclass classification using gradient descent to update weight and bias in each iteration. **PLEASE EDIT bm\_classify.py ONLY**. Any other changes that cause system issues may lead to 0 point in this part.

**Q2.1 Binary Classification - Perceptron vs. Logistic (28 points)**

In our lecture we have been discussing how to do binary classification using different loss. Now it is your time to implement perceptron loss and logistic loss for binary classification. In this problem you are given training set D={(xn,yn)n=1N},yi∈{0,1}∀i=1...N.D={(xn,yn)Nn=1},yi∈{0,1}∀i=1...N. Please note that the labels are not +1 or -1 as we discussed in lectures, so think carefully before you apply formula to this question. Your task is to learn a model wTx+bwTx+b that minimize the given loss. Please find the gradients of the above two loss functions by yourself and apply **average** gradient descent to update w,bw,b in each iteration. For perceptron loss we define Z=yn(wTxn+b)>0Z=yn(wTxn+b)>0 as correctly classified data.

* (8 points) TODO 1 For perceptron loss that is find the minimizer of

F(w,b)=∑n=1NLperceptron(yn(wTxn+b))=∑n=1NMAX(0,−yn(wTxn+b))F(w,b)=∑n=1NLperceptron(yn(wTxn+b))=∑n=1NMAX(0,−yn(wTxn+b))

* (8 points) TODO 2 For logistic loss that is find the minimizer of

F(w,b)=∑n=1NLlogistic(yn(wTxn+b))=∑n=1Nln(1+e−yn(wTxn+b))F(w,b)=∑n=1NLlogistic(yn(wTxn+b))=∑n=1Nln(1+e−yn(wTxn+b))

* (4 points) TODO 3 Also you will find out it is convenient to use sigmoid fuction σ(z)=(1+e−z)−1σ(z)=(1+e−z)−1 for logistic loss, so please complete it. You can use this function in TODO 2.
* (4 points for each)TODO 4 TODO 5 After you learn the models, how do you know it is good enough? The intuitive way is to make some predictions to see if those predicted results are correct or not. Here we want you complete the prediction functions. It will be like something greater than 0 and something put into sigmoid function greater than 0.5. You may find out an interesting fact here.

After you finish the 5 TODOs above in bm\_classify.py, you can run binary.sh to test it. If your code is programmed correctly, you should see binary.out as an output file keeping taining and testing accurancies of two loss functions for three datasets. You will see how similar between perceptron loss and logistic loss. Quite interesting isn't it? Can you figure out why this happends? We will leave this open question to you.

Two of the datasets you are going to do binary classification:

* Synthetic data:
* Two Moon data:

**Q2.2 Multiclass classification - SGD vs. GD (32 points)**

Well done! Ready to take our next challenge? Let's get into multiclass classification. In this question you are going to build a model to classify data into more than just two classes. Also you are going to implement both SGDSGD and GDGD for multiclass classification and compare performances of the two approaches. Training dataset are similar to question Q2.1, but yi∈{0,1,...,C−1}∀i=1...N.yi∈{0,1,...,C−1}∀i=1...N. Your task is to learn models for multiclass classification based on minimizing logistic loss.

Here let me give a short review of SGDSGD.

From our lecture we know multiclass logistic loss is

F(W)=∑n=1Nln(1+∑k≠yne(wk−wyn)Txn).F(W)=∑n=1Nln(1+∑k≠yne(wk−wyn)Txn).

Now we try to apply SGDSGD. First we randomly pick a data xnxn and minimize logistic loss

g(W)=ln(1+∑k≠yne(wk−wyn)Txn).g(W)=ln(1+∑k≠yne(wk−wyn)Txn).

And then find the derivative ▽wg(W)▽wg(W), where ▽wg(W)▽wg(W) is a CxDCxD matrix.

Let's look at each row k.

If k≠ynk≠yn:

▽wkg(W)=e(wk−wyn)Txn1+∑k′≠yne(wk′−wyn)TxnxTn=ℙ(k|xn;W)xTn▽wkg(W)=e(wk−wyn)Txn1+∑k′≠yne(wk′−wyn)TxnxnT=P(k|xn;W)xnT

else:

▽wkg(W)=−∑k′≠yne(wk′−wyn)Txn1+∑k′≠yne(wk′−wyn)TxnxTn=(ℙ(yn|xn;W)−1)xTn▽wkg(W)=−∑k′≠yne(wk′−wyn)Txn1+∑k′≠yne(wk′−wyn)TxnxnT=(P(yn|xn;W)−1)xnT

where ℙP is softmax function.

In the end, our update for WW is

W←W−ηℙ(y=0|xn;W)⋮ℙ(y=yn|xn;W)−1⋮ℙ(y=C−1|xn;W)xTnW←W−η[P(y=0|xn;W)⋮P(y=yn|xn;W)−1⋮P(y=C−1|xn;W)]xnT

That is the whole idea of SGDSGD of logistic loss for multiclass classification.

* (8 points) TODO 6 Complete the SGDSGD part in def multiclass\_train, and don't forget the bias bb. To randomly pick a data from dataset, you can use np.random.choice one time in each iteration.
* (16 points) TODO 7 Complete the GDGD part in def multiclass\_train. Compare to SGDSGD, GDGD does not randomly pick a data xnxn. Instead, GDGDconsiders all training data points to compute derivative. Please think about how to compute GDGD, and again we want **average** gradient descent. Also there is a tricky point. When dataset is large, GDGD will takes a large amount of time. How to reduce the time? Make sure you use **numpy programming** instead of nested for loops, otherwise you will not finish your test on Vocareum within the time limit.

Hint 1: If you need to run a for loop for NN times to accumulate CCxDD matrices, why not design an equivalent computation as a CCxNN matrix dot another NxDNxDmatrix.

Hint 2: You may find it useful to use a special (one-hot) representation of the labels, where each label yiyi is represented as a row of zeros with a single 1 in the column, that corresponds to the class yiyi. So this one-hot should be an NNxCC matrix.

Advice: To avoid numerical issues such as overflow and underflow caused by np.exp. Let xx be a input vector to the softmax function. Use x̃ =x−max(x)x~=x−max(x)instead of using xx directly for the softmax function xx . That is, if you want to compute f(x)if(x)i , compute f(x̃ )i=exp(x̃ i)∑Dj=1exp(x̃ j)f(x~)i=exp(x~i)∑j=1Dexp(x~j) instead, which is clearly mathematically equivalent but numerically more stable.

* (8 points) TODO 8 You need to complete the predict function in def multiclass\_predict. For your convenience, you can use np.argmax.

After you complete TODO 6 TODO 7 TODO 8, please run multiclass.sh to test your code. If your code is programmed correctly, you should see multiclass.out as a output file keeping processing time, taining and testing accurancies of SGDSGD and GDGD for each given dataset. You shall see how fast SGDSGD process compared to GDGD, but its accuracy is hard to catch up with the other's. One more open question to leave you. Is there any chance that SGDSGDreach to the same accuracy as GDGD does using less time?

**Grading**

When you finish all TODO parts, please click submit button on Vocareum. Your code will be tested on some other datasets, and we will grade your solution based on how many testcases you pass for each TODO.  
If your answer takes a long time for grading, Vocareum will kill the process automatically. You are responsible to make sure your code can be graded successfully on Vocareum. It takes about 100 seconds to process the grading script on solution code, and a few more sceonds to get the grading report. Sometimes you may need to come back to check grading report later.

That is all for this assignment.

Two of the datasets you are going to do multicalss classification:

* Clear smile data:
* Blur smile data: