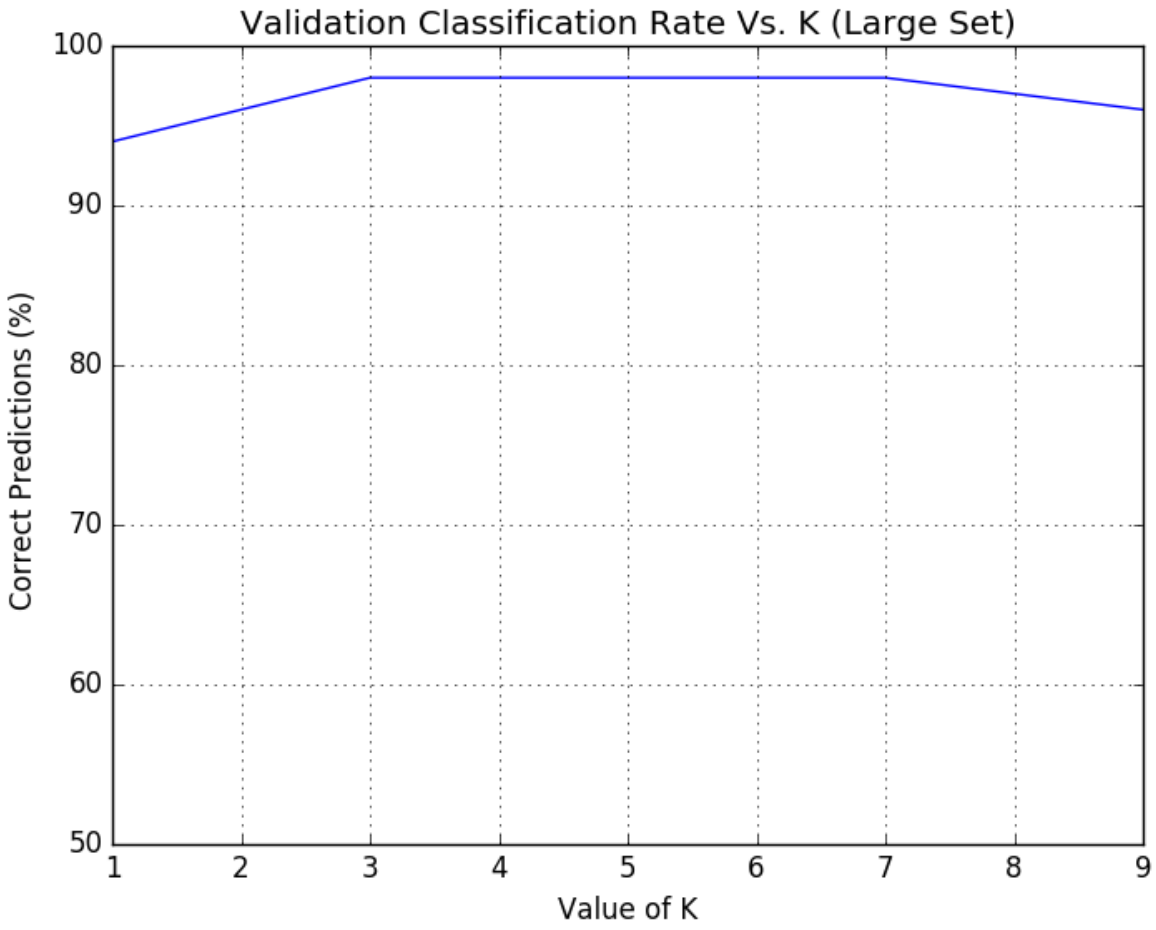
Joel Cheverie
1002924393

CSC 2515: Assignment 1

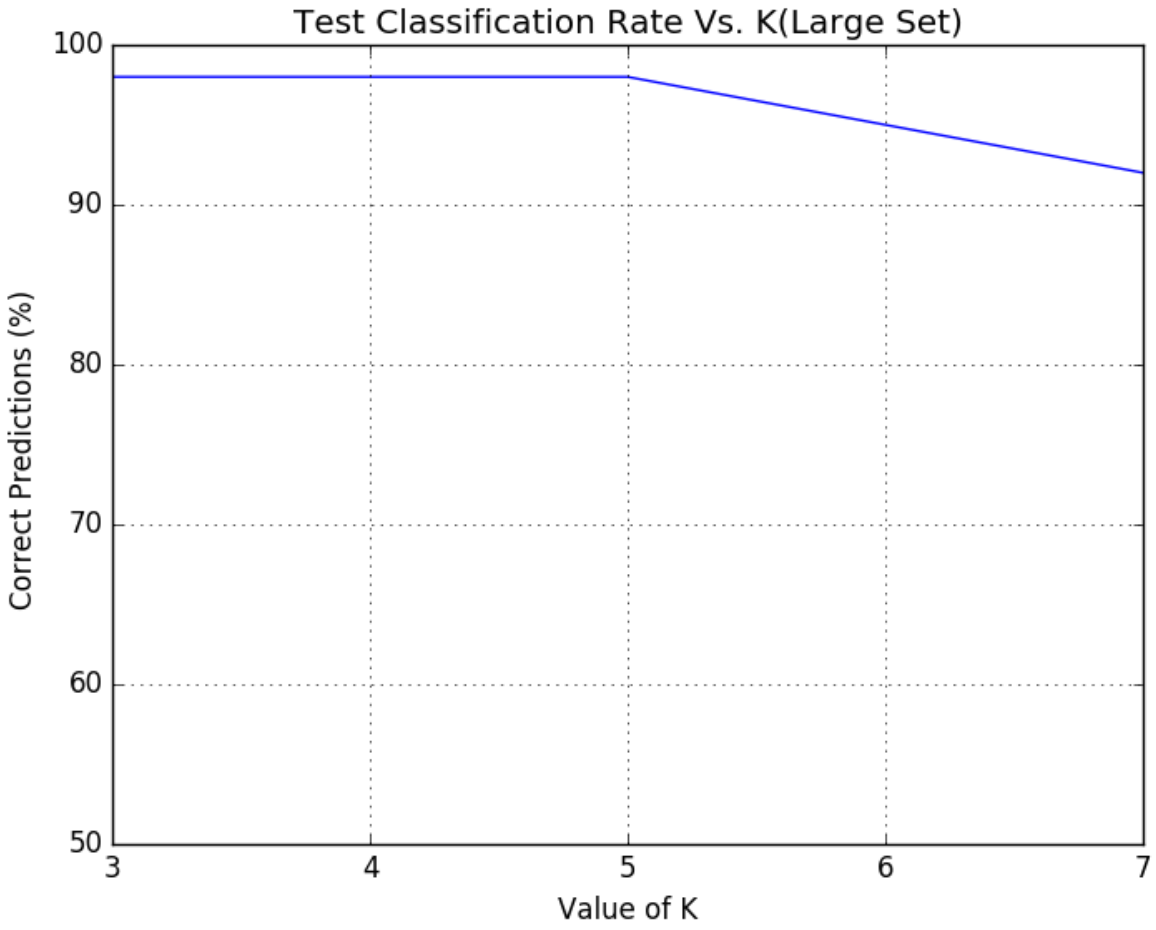
Q. 3

3.1 k-Nearest Neighbours



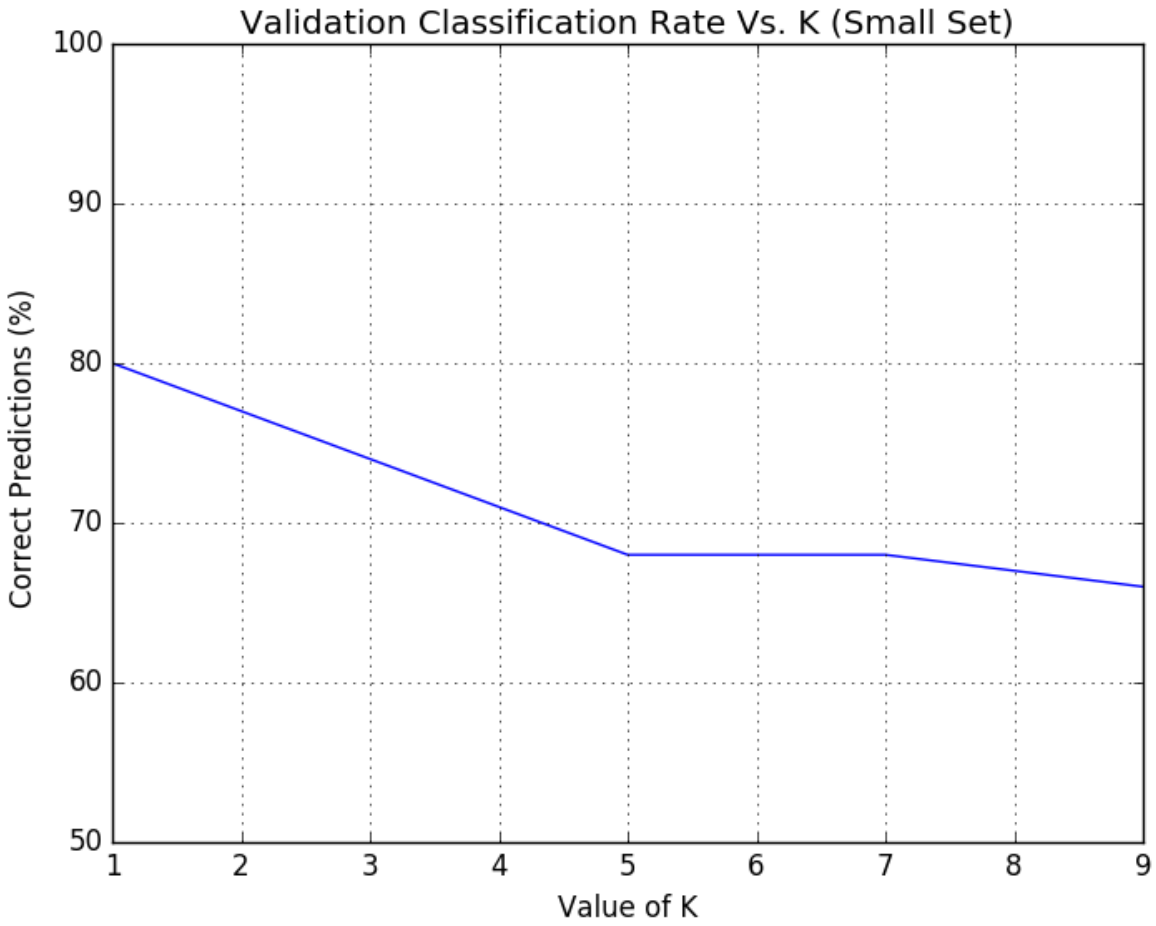
K	Validation Accuracy (%)
1	94
3	98
5	98
7	98
9	96

a) Above is a plot of the validation performance as a function of k for the large training set. The classifier performance first increases, and then decreases with k. Overall it is quite stable and accurate on the validation set, as the average prediction accuracy is 96.8 %. It has the highest accurracy for the k values of 3, 5, and 7 (where the prediction accuracy is 98%). I'm going to take k* = 5 as the validation accuracy has the highest value here and the accuracy is the same for k* + 2 and k* - 2. Here are the plotted accuracy results for these 3 k values on the test set:



K	Test Accuracy (%)
3	98
5	98
7	92

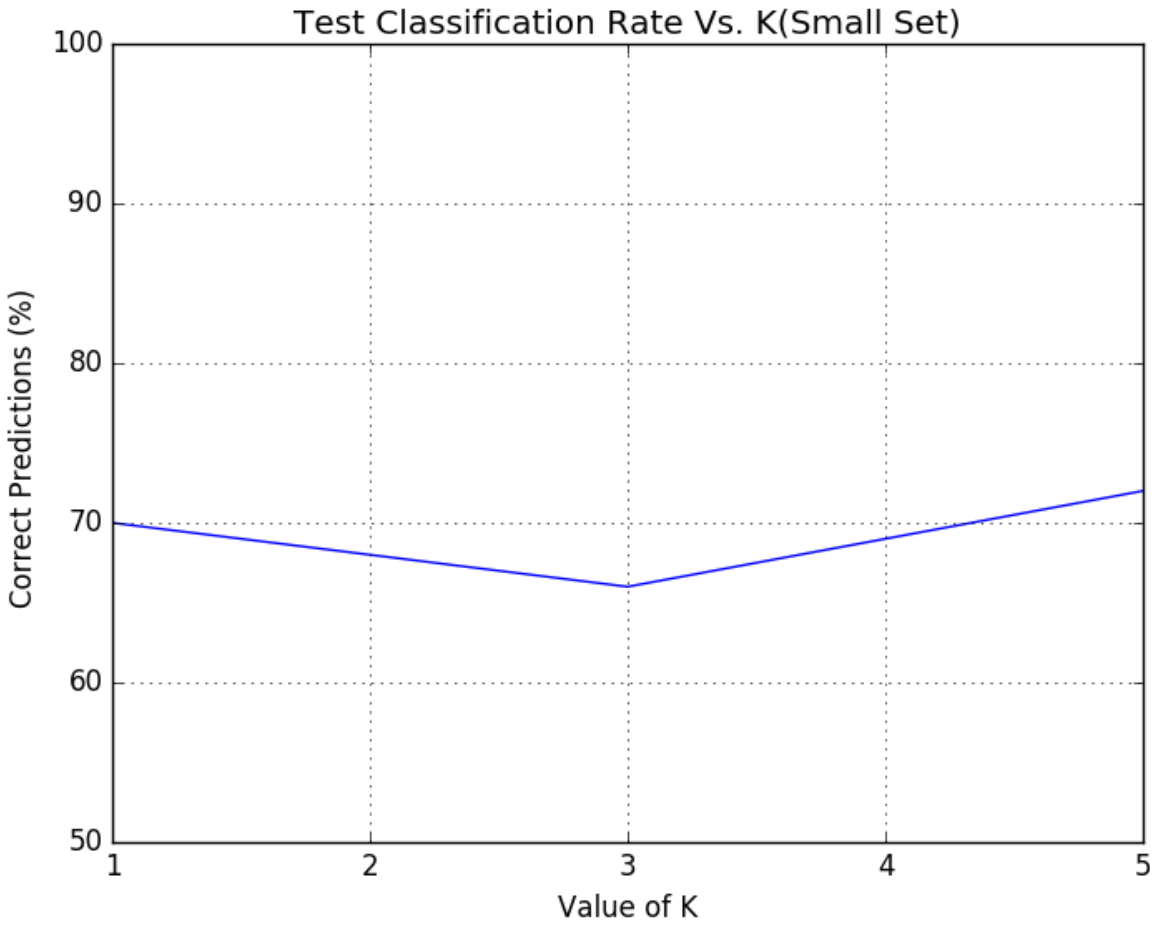
Now looking at the small training set, we see that the results are different:



K	Validation Accuracy (%)
1	80
3	74
5	68
7	68
9	66

Here the classifier performance steadily decreases as k increases. Overall it is less accurate than with the large training set as the averaged accuracy is 70.4 %. It has the highest accurracy for the k value of 1 at 80%. I'm going to take k* = 3 here as the validation accuracy is higher than the average accuracy (at 74%). This also allows me to consider the k with the highest accuracy, k=1 (at k*-2) for

the test set. Hence I'm taking the median of the highest accuracy region of k in the k values of (1,3,5). Here are the plotted accuracy results for these 3 k values on the test set:



K	Test Accuracy (%)
1	70
3	66
5	72

b) Interestingly, the test accuracy for the large test set is identical for k = 3 and k =5. For k=7 the accuracy is lower for the test set than for the validation set (by 6%). This implies the class noise and/or the data variance are likely higher for the test set as with a higher value of k, the accuracy decreases more rapidly. More specifically, the data is likely less clustered into neighbourhoods in the test set than the validation set. Overall the performance is quite similar however, so it can be said that the performance does correspond.

The results are different for the test set with the small training set. Overall it appears less accurate than in the validation set. For k =1 it is less accurate by 10%, for k =3 it is less accurate by 8%, although for k =5 it is more accurate by 4%. I suspect this means that there is more variance and smaller neighbourhoods in the validation set than the test set in this case, hence why k=1 is more accurate for the validation set. Overall the performance is not very similar between the two sets so I don't think it can be said that the performance corresponds.

3. 2 Logistic Regression

a) For my hyperparameters, I have set:
learning_rate = 0.06
num_iterations = 1000

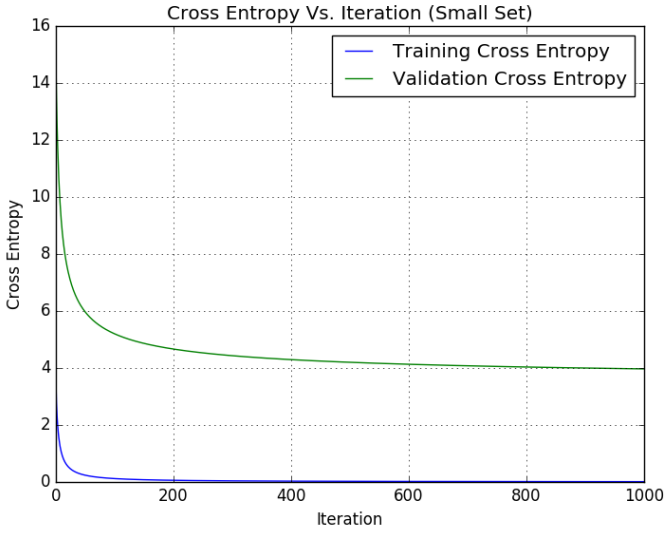
I am ininitializing my weights with a Gaussian distribution with mean 0 and variance 0.0001. I chose these parameters as they resulted in high accuracy and with smaller learning_rate I could track the convergence over a larger spread of iterations.

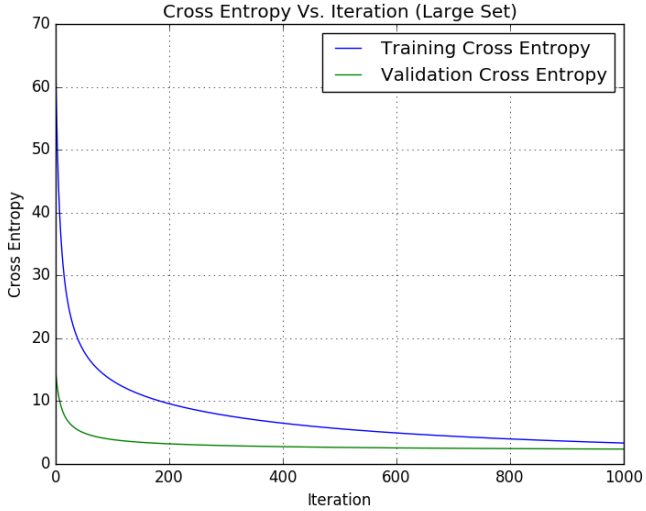
These are the results for the large and small training sets:

Large Training Set:
ITERATION:1000
TRAIN NLOGL:0.04
TRAIN CE:3.362690
TRAIN FRAC:100.00
VALID CE:2.384855
VALID FRAC:92.00
TEST CE:1.591762
TEST FRAC:96.00

Small Training Set:
ITERATION:1000
TRAIN NLOGL:0.00
TRAIN CE:0.011808
TRAIN FRAC:100.00
VALID CE:3.816212
VALID FRAC:80.00
TEST CE:4.420310
TEST FRAC:74.00

b) Below are plots for the training and validation cross entropies as a function of iteration.





The results in my plots do not change even if I run my code several times. I suspect it is due to my small variance Gaussian distribution for the weight initialization. This consistency is nice as it allowed me to tune my hyper parameters to a level that I was comfortable with. It produced great results for both the small and large training set, but as expected the data seems to overfit the smaller training set by achieving 100 % accuracy and near 0 cross entropy within a hundred iterations. The larger training set's validation cross entropy eventually converges to a value lower than that of the validation set in the smaller training set case, again implying that the larger training data helps solve the overfitting issue.

3.3

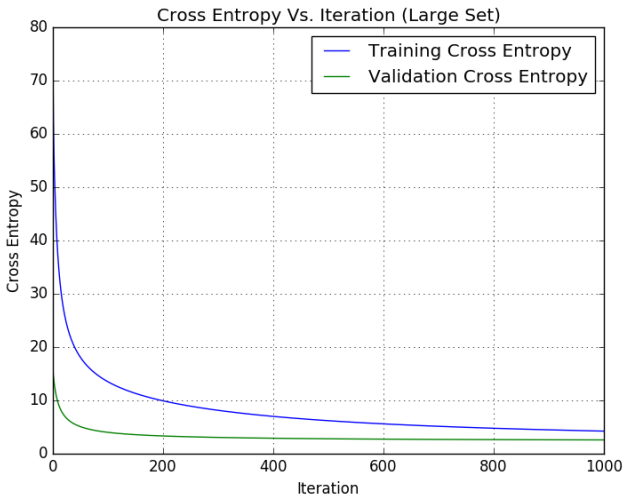
a) The best hyperparameters I found for the penalized logistic regression were learning_rate = 0.06, num_iterations = 1000, weight_decay = 1.0 with weights initialized with mean 0 and variance 0.0001 in a Gaussian distribution. Similar to before, by having a smaller learning rate, I could run for more iterations before convergence. This allowed me to track performance better.

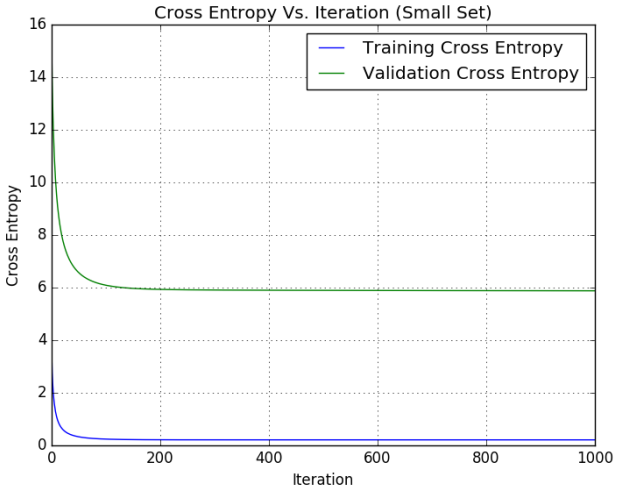
These are the results for the large and small training sets:

Large Training Set:
ITERATION:1000
TRAIN NLOGL:0.04
TRAIN CE:3.365190
TRAIN FRAC:100.00
VALID CE:2.347047

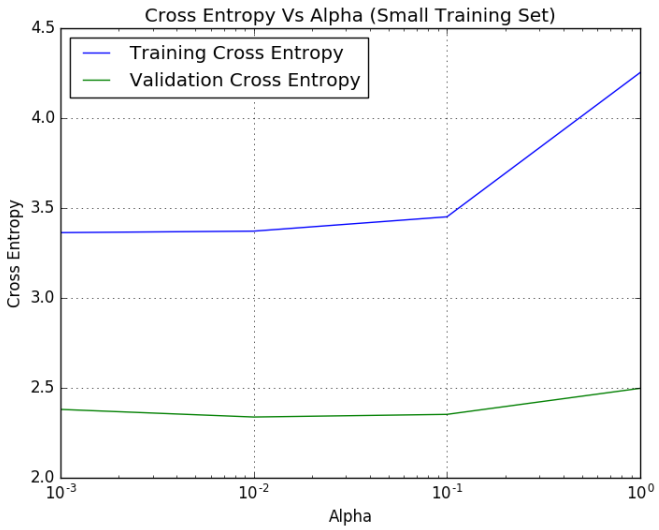
VALID FRAC:92.00
TEST CE:1.589346
TEST FRAC:96.00

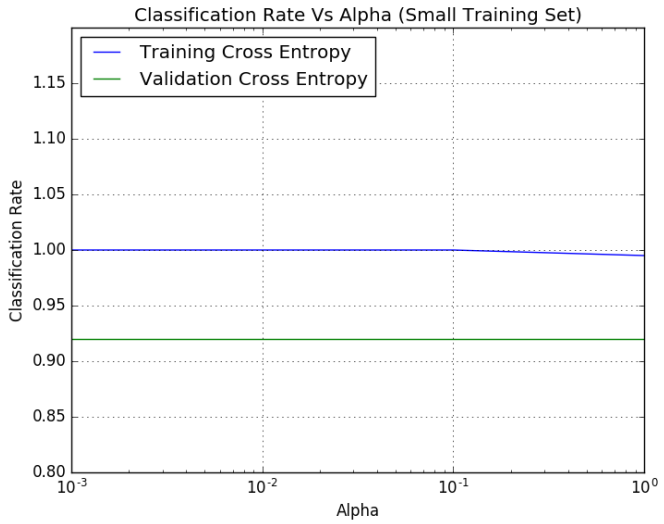
Small Training Set:
ITERATION:1000
TRAIN NLOGL:0.10
TRAIN CE:0.210423
TRAIN FRAC:100.00
VALID CE:5.878254
VALID FRAC:80.00
TEST CE:6.126948
TEST FRAC:72.00

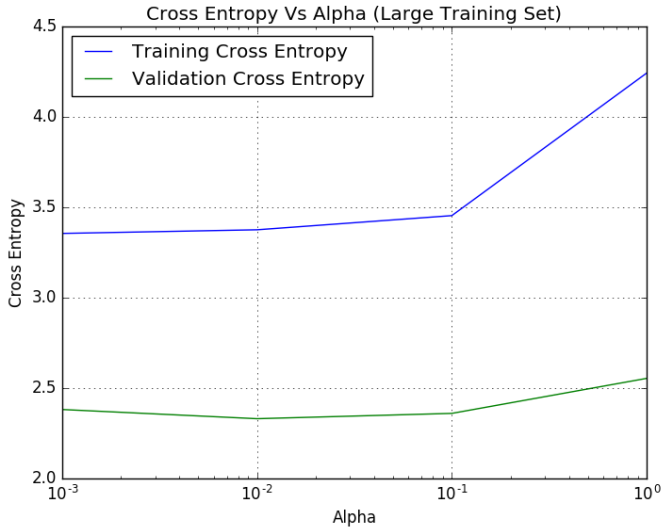


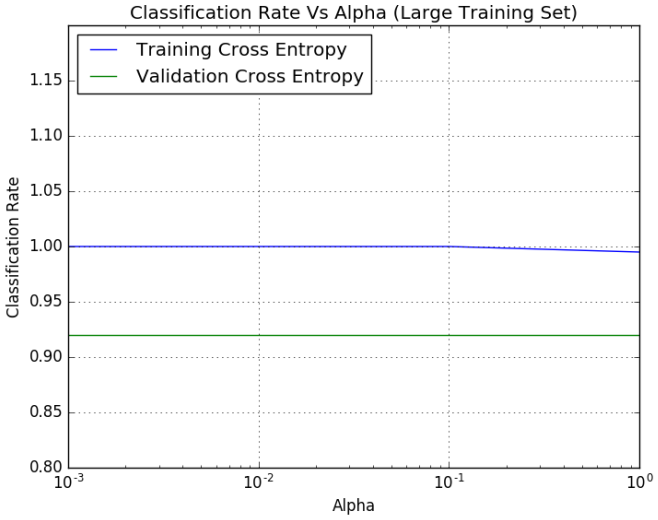


b) Here are my plots of final Cross Entropy and Classification Rates for the Small Training and Large Training Sets as a function of Alpha:









I found that the cross entropy increased as alpha increased on both the large and small training set. I'm assuming this occurs as alpha governs the regularization strength of the model. With a higher value of alpha, the regularization strength increases, which can help to combat overfitting. It also leads to a higher total loss value in the training set and hence an increased cross-entropy value. For both sets the validation cross entropy at first decreases and then increases for alpha =1.0. I suspect that with lower regularization strength, the model may not be generalizing quite as well for the validation set, hence increasing alpha to a point actually improves the cross entropy loss for this set marginally as it helps to generalize the model. Once alpha reaches 1.0 however, the model begins to underfit the validation set however.

Conversely, the classification rate hardly changes with alpha. As alpha increases, it does decrease slightly as the model likely generalizes a bit better as overfitting is slightly reduced, but overall it is quite consistent. I'm assuming this is due to gradient descent and my weight initialization already keeping the weights small while predicting the training set. Hence regularization affects cross entropy values more than classification rate in my model.

The best value of alpha based on my experiments is 1.0. It may increase the cross entropy loss in my model slightly, but more importantly this value increases the generalization of my model. The improvement is not major in my model however as I suspect that my model's small initial weights already combat overfitting well without regularization, hence its contribution to the success of my model seems to be minimal.

c)

First let's look at the data for the penalized and non-penalized logistic regression again:

Penalized Logistic Regression (Large Set)	Non-Penalized Logistic Regression (Large Set)
ITERATION:1000 TRAIN NLOGL:0.04 TRAIN CE:3.365190 TRAIN FRAC:100.00 VALID CE:2.347047 VALID FRAC:92.00 TEST CE:1.589346 TEST FRAC:96.00	ITERATION:1000 TRAIN NLOGL:0.04 TRAIN CE:3.362690 TRAIN FRAC:100.00 VALID CE:2.384855 VALID FRAC:92.00 TEST CE:1.591762 TEST FRAC:96.00

For the large set, the performance is roughly the same. The only gain seems to be in that the test set's cross entropy actually decreases with regularization here (so regularization does help slightly). In any case, it appears that the model already generalizes quite well and overfitting is not an issue that needs to be addressed very much here. Regularization's impact on performance is thus minimal.

Penalized Logistic Regression (Small Set)	Non-Penalized Logistic Regression (Small Set)
ITERATION:1000 TRAIN NLOGL:0.10 TRAIN CE:0.210423 TRAIN FRAC:100.00 VALID CE:5.878254 VALID FRAC:80.00 TEST CE:6.126948 TEST FRAC:72.00	ITERATION:1000 TRAIN NLOGL:0.00 TRAIN CE:0.011808 TRAIN FRAC:100.00 VALID CE:3.816212 VALID FRAC:80.00 TEST CE:4.420310 TEST FRAC:74.00

For the small test set, performance actually decreases with regularization. Specifically, the test set classification rate decreases by 2%, which implies the model is actually worse. I suspect that the model may be underfitting the data with regularization here. I also presume again that due to my small weight

initialization coupled with gradient descent keeping weights small, that regularization isn't really needed in my model.

So in conclusion regularization induces a very minor improvement for the large data set, but actually decreases performance in the small data set. Penalization should therefore be used for the large data set, but not for the small data set.

d)
Comparing KNN results with the penalized and non-penalized logistic regression results we have the following tables:

Validation (Small Training Set)

Best Knn Result (K =1)	Penalized Logistic Regression	Non-Penalized Logistic Regression
80 %	80 %	80 %

Test (Small Training Set)

Best Knn Result (K =5)	Penalized Logistic Regression	Non-Penalized Logistic Regression
72 %	72 %	74 %

Validation (Large Training Set)

Best Knn Result (K = 3,5,7)	Penalized Logistic Regression	Non-Penalized Logistic Regression
98 %	92 %	92 %

Test (Large Training Set)

Best Knn Result (K = 3, 5)	Penalized Logistic Regression	Non-Penalized Logistic Regression
98%	96 %	96 %

Looking at these summaries, I find that generally KNN actually outperforms logistic regression on this data set, especially for the large training set. In the small training set, non-penalized logistic regression initialized with small weights in a Gaussian distribution seems to be the way to go as it offers a minor improvement over the other methods.

Choosing one method over another in general really depends on the distribution of data as well as the size of a data set. KNN seems to work best with lots of samples, while logistic regression works better if there is more class noise. I feel that in general running an experiment similar to this assignment would be the best way to decide between the two methods, although KNN is likely more robust as it can form more complex decision boundaries than logistic regression.

Turning to advantages and disadvantages of methods, KNN seems to be more powerful than linear regression in general. Specifically, it handles data sets that contain many samples of non-linear data better. Another advantage of KNN is that training is simple. To train simply involves storing a copy of data in memory with a distance metric between points. Provided there isn't too much data (and memory is cheap), this is not a huge drawback. Logistic regression on the other hand works best as a linear classifier. If data is linearly separable, this method will likely work best. It also handles noise better than KNN, which can fall into pitfalls with noisy data. Training is more difficult here than in KNN, but storing the training data isn't required (so if memory is a concern this approach is likely better too). In both cases hyperparameters must be tuned, so neither can be adapted immediately to any data set without considering and adapting to its distribution and properties.