**ECE521 Assignment 1**

Student: Winston Wong (1001614853)

1.1.1)

For every element in the training set, there must exists k ≥T ≥ 0 such that the i-th neighbor of such element is equal to the (i+T)-th neighbor. That is, the class of neighbor for every element as function of k must be periodic, where k is the kth closest element.

Justification:

Let the random variable = 0,1 be result of classification of training example i with k neighbors and N samples in total, where 1 is successful, 0 otherwise. Then, the classification accuracy can be represented as

If C is periodic with k, then:

Thus

One particular case when the above equality holds true is if . That is, the i-th neighbor falls into the same class of the (i+T)-th neighbor. Another example is the infinite sequence shown below:

Has a periodic classification accuracy:

|  |  |
| --- | --- |
| K=1 | 100% |
| K=3 | 0% |
| K=5 | 100% |
| K=7 | 0% |

And so on.

Q1.1.2)

We let, So,

And is given by

The expected value in the variance operator is just a constant thus can be brought out of the variance operator, as shown below

Using the property that

Therefore

It is given in the hints that is normally distributed with a variance of. That is . Therefore,

Combining all the results, we have:

As required.

1.2.1)

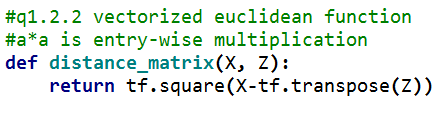
In this case, we have that:

Decomposing the terms in the summation, we have that

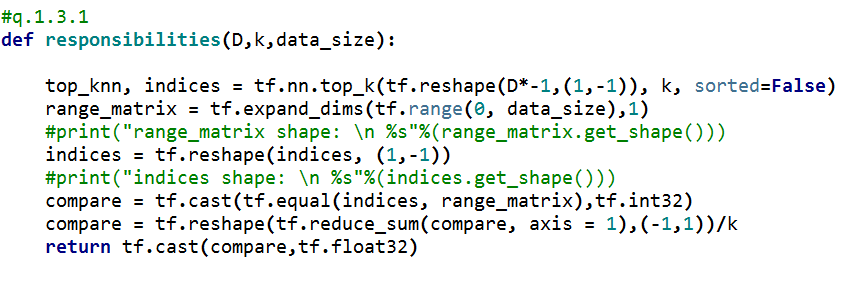
The nearest neighbor is given by the entry with the minimum value. In this case, the third term is the same for all dimensions as it is just the magnitude of the test point. The first term encompasses the magnitude of the corresponding input vectors. Since it is given that the magnitude for all input vectors are the same, it is also constant throughout all the dimensions. The only term that is left to analyse is the second term, which can be expressed as follows:

Where 2 is just a constant scaling factor. Thus by ranking the negative inner product, the nearest neighbors can be found.

1.2.2)



1.3.1)



1.3.2)

The loss for different k values are reported as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | K= 1 | K=3 | K=5 | K=50 |
| Training MSE | 0.0 | 0.1052421151 | 0.118541242468 | 1.24800941817 |
| Validation MSE | 0.0339437068357 | 0.040784791664 | 0.0388048138193 | 0.153587773515 |
| Test MSE | 0.0388755015538 | 0.0181364953241 | 0.0222908216956 | 0.0883668301976 |

Therefore, according to the validation MSE, the best k is 1.

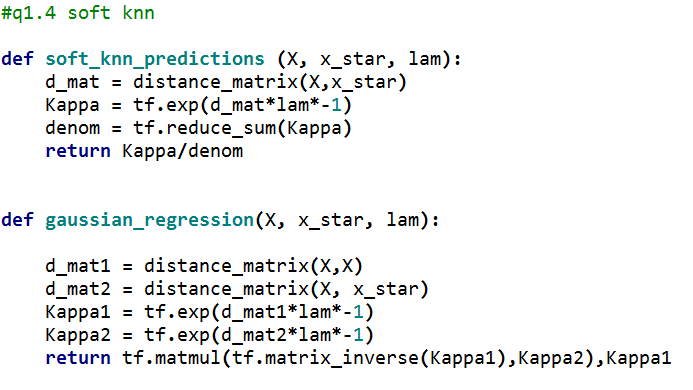
As shown in graphs below, the smaller the k value, the closer the prediction resembles the original data. We can observe a similar trend from the mean square error in the training set. However, the same could not be said when k-NN is performed on the validation or the test set.

|  |  |
| --- | --- |
|  |  |
| K=1 | K=3 |

|  |  |
| --- | --- |
|  |  |
| K = 5 | K =50 |

1.4)

The python code for soft knn and Gaussian regression is as shown below.



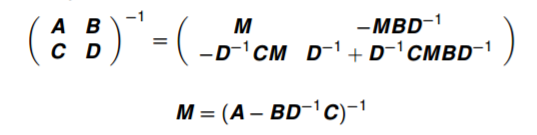
One difference between soft KNN and Gaussian regression is that the individual responsibilities of soft-kNN sums to 1, but the individual responsibilities of Gaussian regression does not sum to 1. Soft KNN responsibilities in fact is a constant multiple of the distance matrix, but the same cannot be said for Gaussian regression. Soft KNN works well with , but not true for Gaussian regression, where

|  |  |
| --- | --- |
|  |  |
| Soft KNN with | Gaussian Regression |

1.4.2)

First, we define the following

Using the given matrix formula,



We have that

With The joint distribution is given by the following:

**Let**

(\*)

We are asked to find . Thus and henceare given. (Thus constant in this case).

We want to cast equation (\*) (i.e. into a sum of 2 quadratic forms in and respectively.

Again in the case that is constant, Q() is constant. Let be in the form

Expanding we have

We observe that equation (\*) could be massaged into the following

Comparing the terms highlighted in (a) and (b) we have that

And

As required.

2.1.1) The Jensen Inequality says that a function is convex iff

The Loss function is given by:

So

By Triangle Inequality, for,

And

Thus

The function is also convex in b.

2.1.2)

We have a new input matrix such that

Thus, in terms of the original vector , the transformed input vector

Can be written as:

Where

The loss function for the transformed input is given by

The above expression is identical to the original loss function with weight being and the bias being . Therefore, Let the optimized weight and bias for the original dataset be and respectively, then the optimum weight and bias will be

At minimum, since and , the minimum loss function will be the same as the original dataset.

2.1.3)

Now, if the loss function is regularized and the input is the transformed dataset,

The analytical solution is the ridge regression given as follows:

Where Y is the output matrix, and is the new input matrix M (number of training examples) by N(input vector dimensions). can be expressed in terms original input matrix as follows

Again, is the matrix mentioned above, and is a similar matrix to but with a dimension M by N, as follows

So the ridge regression equation becomes

The minimum loss can be expressed as follows

2.1.4)

To solve a multi-class classification using only binary classifiers, we run a series of binary classification to each class. For example, there are 4 classes (i.e. A, B, C, D). First, we train a prediction function to classify whether the input belongs to class A or does not belong to class A. We repeat the same steps for class B,C and D. Now that we have 4 prediction functions predicting whether the input belong/does not belong to a particular class, we then compare the output of each of the prediction function to see which is the maximum. The input is likely to fall into the class which gives maximum value.

2.2.1) (See a1.2.py for python script)

With a mini-batch size of 50, the SGD algorithm converges the fastest with a learning rate on the scale of (i.e. 0.125). Further increasing the learning rate (to 0.25) will result in a blow up of the loss function.

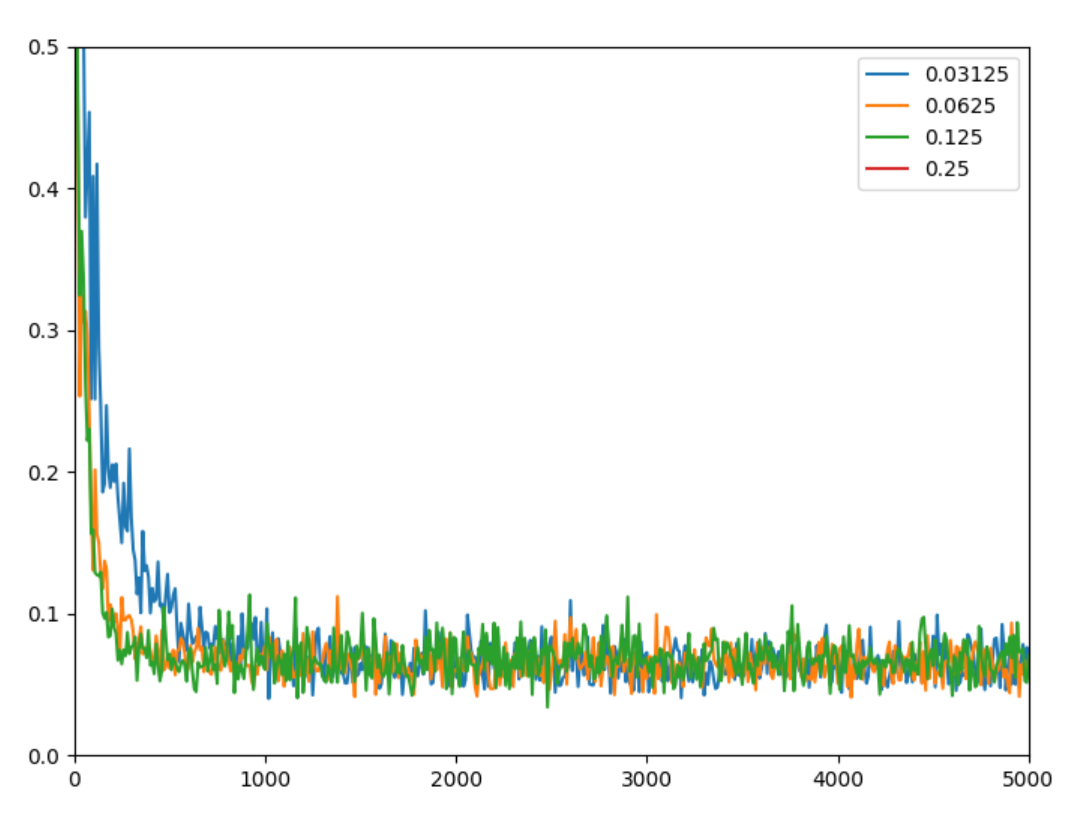


Figure Loss vs Number of steps for different learning rate

As the learning rate increases, the loss function converges faster. However, the loss function will blow up if learning rate is too large.

2.2.2) (See a1.2.py for python script)

Running the same test for mini-batch size of 1, 50, 100 and 700, it was found that the optimum learning are (on the scale of 0.125) for all sizes.

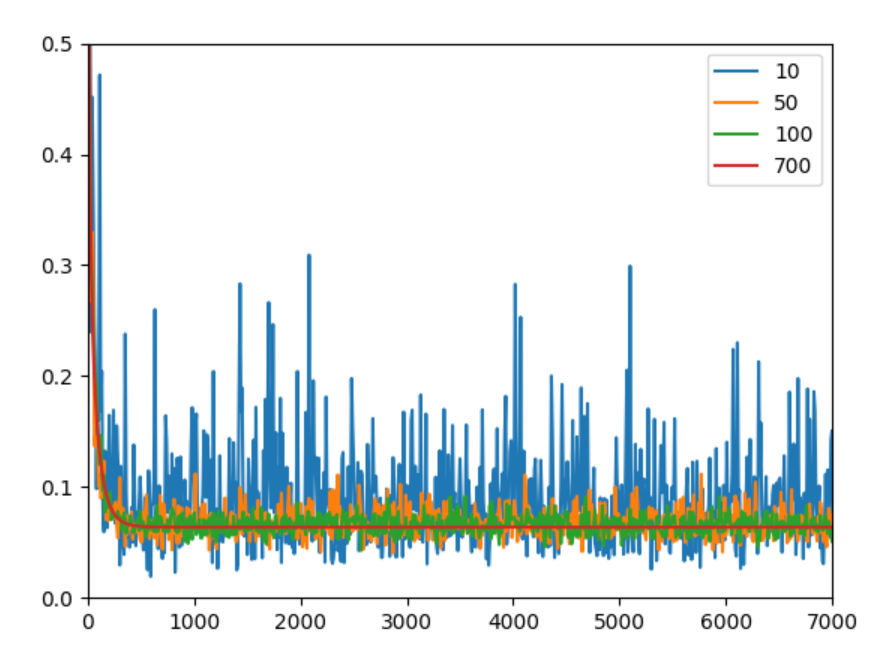


Figure Loss vs Number of Steps for different batch size

From the above figure, we observe that as the batch size decreases the noise increases. All batch sizes give a similar convergence speed. However, the smaller the batch size, the faster each iteration is. Thus, using a minimum mini-batch size of 10 gives the best training time.

2.2.3)

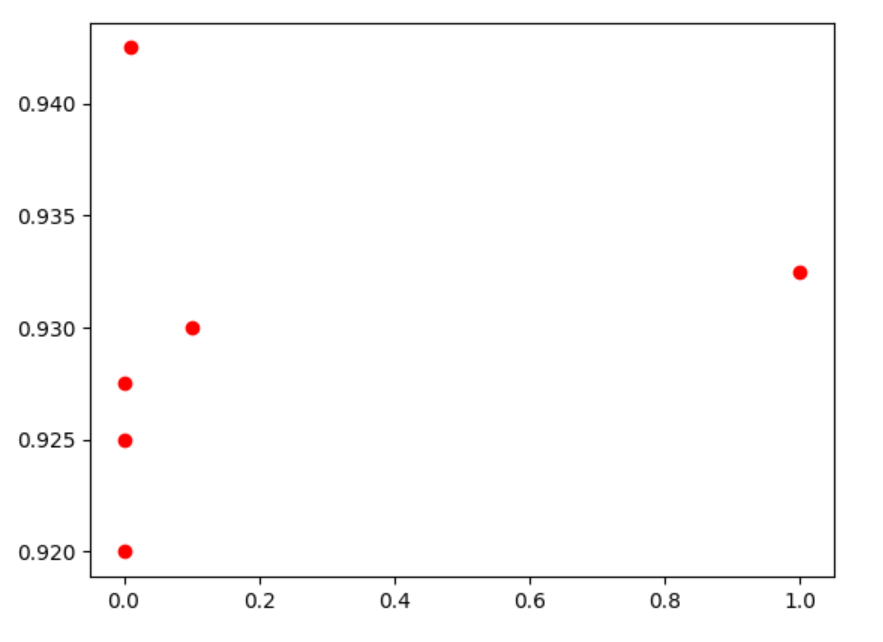


Figure Test Accuracy vs Regularization factor

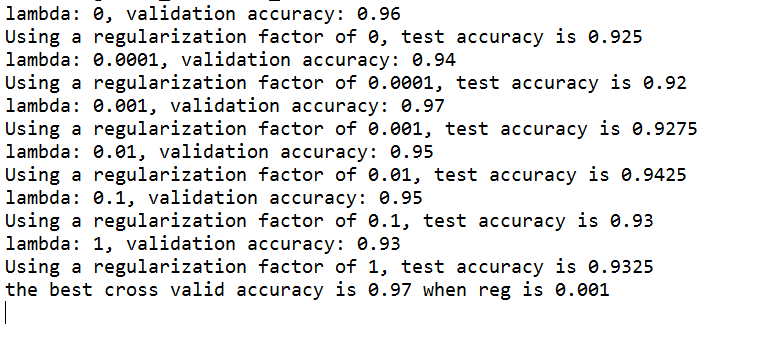


Figure Test Result

The test results are shown above. From the plot, it can be observed that optimal regularization lies in the middle. Too much or too little weight penalty will have a negative effect on test accuracy. This can be further illustrated by the graph below that shows the relationship between test accuracy and a list of regularization factors ranges from1 to .

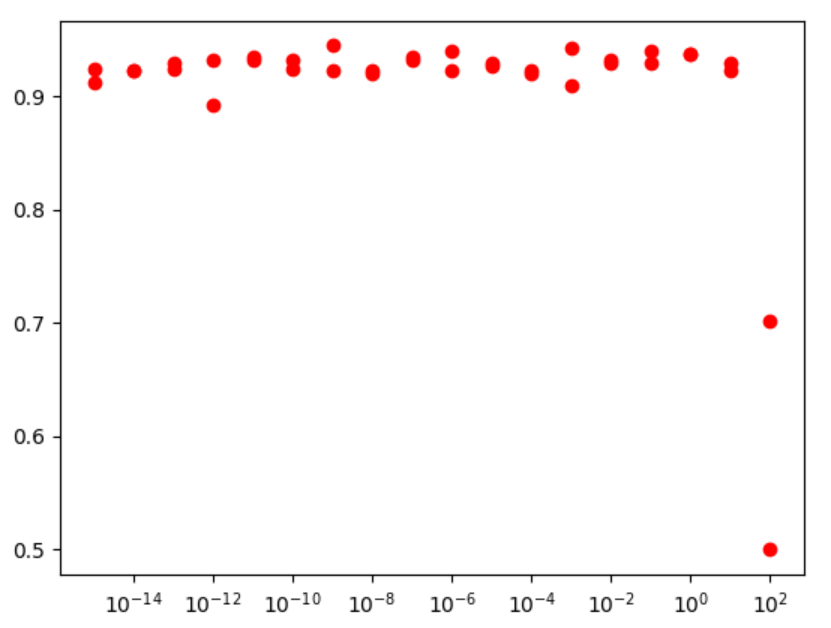


Figure Test Accuracy vs Regularization Factor (log scale) [E-14 to E2]

We use the cross-validation set to tune our hyperparameter instead of the training set because by tuning the hyperparameter we are essentially conducting an estimation of performance, which requires an independent set of data. Simultaneously tuning the hyperparameter and optimizing the parameters will lead to high bias.