Ahmad Saad

CIS 4516-001

Prof. Kai Zhang

The assignment required me to implement a k-nearest neighbour algorithm and a pocket algorithm to train and test the Letter Recognition Data Set.

kNN ALGORITHM

The following was my algorithm for kNN:

```
def test_knn(train_x, train_y, test_x, num_nn):
    For each value in testing:
        For each value in training:
            Find and store the distance between test value and training value
        Find the k smallest distances and their respective index
        Predict the label of the test value by counting the occurrences of the labels in the
        k smallest distances. The label that is the nearest neighbour the greatest number
        of times is taken as the predicted label
        Add the label to a Predicted Label array
    Return the prediction array
```

This algorithm was run over multiple sizes of training data and multiple values of k as per the assignment's requirements. For each value of test_x, the algorithm calculated the distance of that point with every value in train_x. Then, it sorts the distance array to find the k smallest distances and returns their indices. The label of the test point is predicted by finding the class with the highest occurrence in the k closest points.

The distance between two n-dimensional points was calculated by finding the 2-norm of (test Point – train Point) using the numpy.linalg.norm() function

The sorting of the distance array was done using numpy's argsort() method which returns the indices of the elements of the sorted array (in the ascending order).

The classification accuracy reported was as follows:

```
Training data size:  100
Number of nearest neighbor's used:  1
Testing Accuracy:  34.52%
Number of nearest neighbor's used:  3
Testing Accuracy:  27.22%
Number of nearest neighbor's used:  5
Testing Accuracy:  25.26%
Number of nearest neighbor's used:  7
Testing Accuracy:  24.52%
Number of nearest neighbor's used:  9
Testing Accuracy:  22.74%
```

```
Training data size:  1000
Number of nearest neighbor’s used:  1
Testing Accuracy:  75.72%
Number of nearest neighbor’s used:  3
Testing Accuracy:  69.92%
Number of nearest neighbor’s used:  5
Testing Accuracy:  68.02%
Number of nearest neighbor’s used:  7
Testing Accuracy:  64.78%
Number of nearest neighbor’s used:  9
Testing Accuracy:  63.0%

Training data size:  2000
Number of nearest neighbor’s used:  1
Testing Accuracy:  83.06%
Number of nearest neighbor’s used:  3
Testing Accuracy:  79.64%
Number of nearest neighbor’s used:  5
Testing Accuracy:  78.06%
Number of nearest neighbor’s used:  7
Testing Accuracy:  76.18%
Number of nearest neighbor’s used:  9
Testing Accuracy:  74.88%

Training data size:  5000
Number of nearest neighbor’s used:  1
Testing Accuracy:  90.86%
Number of nearest neighbor’s used:  3
Testing Accuracy:  89.18%
Number of nearest neighbor’s used:  5
Testing Accuracy:  88.94%
Number of nearest neighbor’s used:  7
Testing Accuracy:  87.96%
Number of nearest neighbor’s used:  9
Testing Accuracy:  86.82%

Training data size:  10000
Number of nearest neighbor’s used:  1
Testing Accuracy:  93.84%
Number of nearest neighbor’s used:  3
Testing Accuracy:  93.32%
Number of nearest neighbor’s used:  5
Testing Accuracy:  92.96%
Number of nearest neighbor’s used:  7
Testing Accuracy:  92.96%
Number of nearest neighbor’s used:  9
Testing Accuracy:  92.48%

Training data size:  15000
Number of nearest neighbor’s used:  1
Testing Accuracy:  95.44%
Number of nearest neighbor’s used:  3
Testing Accuracy:  94.86%
Number of nearest neighbor’s used:  5
Testing Accuracy:  94.64%
Number of nearest neighbor’s used:  7
Testing Accuracy:  94.56%
```

Number of nearest neighbor’s used: 9
Testing Accuracy: 94.46%

The increase in number of neighbours and the training sample size increased the run time of the algorithm greatly. A general trend observed was that the Accuracy decreased when the number of nearest neighbours considered increased. The highest accuracy of 95.44% was achieved when the training sample size was 15000 and the number of nearest neighbours considered was 1. Another observation noted was the accuracy was the lowest when the training data set was 100. This is understandable since the data could have been sparse and random and as such the representation of each letter might not have been adequate. This might have resulted in incorrect predictions.

POCKET ALGORITHM

For Pocket algorithm, I performed the following algorithm:

```
def train_pocket(train_x, train_y, num_iters):
    Initialize a 26x16 weight matrix that has 26 weight vectors corresponding to the 26
    labels
    For I from 0 to num_iters:
        For each value in train_x:
            Initialize the dot product array to 0
            For each row vector in weight matrix:
                Find and store the dot product between train value and every
                weight vector
            Find the largest dot product and predict the label based on it
            If prediction is not correct, update the weights: add x to the weight
            vector corresponding to the correct label; and subtract x from the weight
            vector correspond to the incorrect label
    return the weight matrix


def test_pocket(weight_matrix, test_x):
    For each value in test_x:
        Initialize the dot product array to 0
        For each row vector in weight matrix:
            Find and store the dot product between test value and every weight
            vector
        Find the largest dot product and predict the label based on it
        Store all the predictions in a PredictionArray
    return PredictionArray
```

The algorithm initialized a weight matrix of dimensions 26 x 16 in which each row corresponded to a weight vector for the label row_number. So, weight vector in row 0 was the weight vector for label 0 (or A) and so on. Then for each value in the training data set, we find the dot product of that value with every weight vector in the matrix. The weight vector that generates the largest dot product is the predicted label of that point. If the predicted label does not match the actual label of the point, we update the weight vectors by adding the point to the actual label and subtracting the point from the incorrect label. This results in the vectors to rotate/ move towards the correct values. This whole procedure is repeated num_iters time where num_iters is the number of iterations of the training algorithm. Once the training is done, the weight matrix is returned and the pocket algorithm is ready for testing.

To test the pocket algorithm, we call the test_pocket function which receives the weight matrix and the test data set as inputs. To predict the label of each test point, we find the dot product of the test point with every weight vector. The weight vector that generates the biggest dot product helps predict the label of that point. This label is then stored in a predictionLabel array and returned.

The number of iterations for the pocket algorithm was chosen based on trial and error. I tried values like 10, 50, 100, 150, 200, 500, and 1000 but the following were my observations:

- The accuracy was very low for number of iterations less than 100.
- The accuracy for number of iterations between 150 and 200 resulted in almost similar values (about 65%).
- The run time was excessively long for number of iterations between 500 and 1000. Also, the accuracy did not improve either.

As such, based on these observations, I decided to go with 150 as the number of iterations.

Besides this, I chose to ignore the bias term as well when calculating the weight vectors as it added addition complexity to the algorithm.

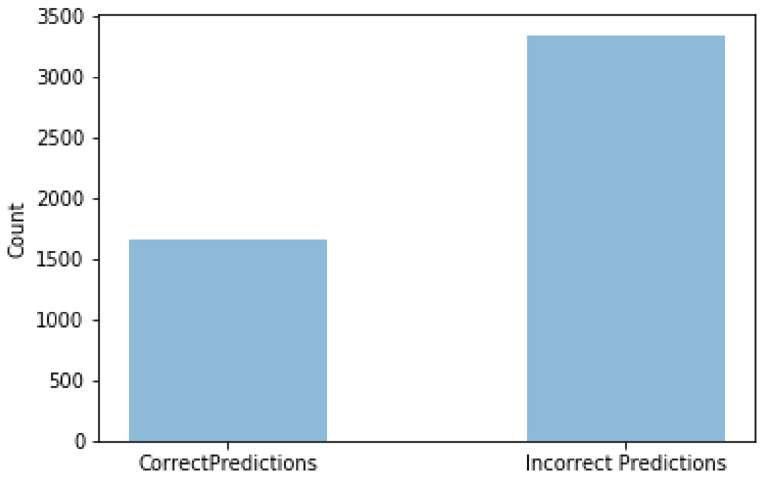
The following were the results of the pocket algorithm:

Number of iterations/ epochs for Pocket: 150

Size of Training data: 100
Accuracy: 33.12%
Confusion Matrix:

Predicted: on x-axis
Actual: on y-axis

```
[139.  0.  0.  4.  0.  0. 13.  2.  0.  0.  0.  5.  6.  3.  1.  6.  5.  4.  0.  0.  0.  6.  2.  2.  8.  0.]
 [  2. 42.  0. 24. 11.  1.  2. 33.  0.  0.  0.  0.  2.  0.  0. 17.  9.  0. 22.  0.  0.  7.  0.  0.  1.  0]
 [  0.  0. 44.  0. 52.  0. 27.  1.  0.  0.  0.  0.  5.  7.  1.  0.  0.  2.  0.  0. 29.  0.  3.  0.  0.  0.]
 [  1. 12.  0. 140.  4.  5.  0. 30.  0.  1.  0.  0.  1.  7.  1.  3.  4.  0.  0.  0.  0.  3.  0.  4.  0.  0.]
 [  2. 10.  4.  4. 139.  1.  9.  3.  1.  0.  0.  2.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0. 14.  1.  0.]
 [  0. 10. 12. 17.  2.  2.  2.  6.  0. 10.  0.  0.  1. 45.  0.  7.  0.  0.  0. 35. 26.  9.  0.  2.  8.  0]
 [  0.  1. 46.  2. 22.  5. 50. 16.  0.  0.  0.  3.  6.  0.  2.  9.  7.  2.  5.  0. 21.  9.  2.  0.  0.  0.]
 [  1.  0.  1. 16.  2.  0.  1. 104.  0.  1.  0.  0.  4. 18.  0.  0.  3.  0.  0.  0.  7. 10.  2.  3.  5.  0.]
 [  2.  8. 19. 12. 72.  3.  8.  0.  6. 13.  0.  0.  0.  6.  0.  3.  0.  0.  0. 12.  9.  4.  0. 28.  0.  0]
 [ 12.   7.   0.   5.   1.   6.   0.   2.   0. 135.   0.   0.   0.   0.   0.   5.   0.   0.   0.   2.   0.   8.   0.   0.   0.   0.]
 [  2.   7.   3.   6.  38.   1.  37.  17.   0.   0.   0.   5.   4.   4.   0.   0.   4.   7.   0.   0.  13.   2.   1.  23.   3.   0]
 [  4.  11.   3.  17.  13.   1.  16.   4.   0.   0.   0. 109.   0.   7.   0.   0.   1.   0.   0.   0.   0.   3.   0.  12.   4.   0]
 [  0.   1.   0.   0.   0.   0.   0.  93.   0.   0.   0.   0.  80.   2.   0.   0.   0.   1.   0.   0.   1.   0.   9.   0.   0.   0.]
 [  1.   0.   0.   8.   0.   0.   0.  70.   0.   0.   0.   0.   7.  83.   0.   0.   0.   3.   0.   0.   2.   9.  14.   0.   1.   0.]
 [  3.   0.   7.  36.   1.   2.   7.  85.   0.   1.   0.   0.   8.   2.   6.   9.   4.   0.   0.   0.  10.   0.   0.   0.   0.   0.]
 [  0.   4.   0.  26.   2.   0.   0.  36.   0.   1.   0.   0.   0.  26.   0.  68.   1.   0.   0.  13.   0.  29.   0.   0.   1.   0]
 [  7.   9.   8.  14.   7.   6.  77.  13.   0.   0.   0.   4.   2.   3.   0.   3.  43.   0.   0.   0.  11.   4.   3.   3.   0.   0.]
 [  1.   6.   1.  17.   1.   2.   1. 134.   0.   0.   0.   0.   7.   9.   0.   2.   1.  20.   0.   0.   0.   5.   0.   1.   0.   0.]
 [  3.  14.   0.  18.  43.   6.   0.   0.   0.   2.   0.   7.   0.   0.   1.   9.   4.   0.  58.   1.   2.   2.   0.  11.  17.   0]
 [  0.   2.   0.   1.  10.   0.   0.   2.   0.   0.   0.   0.   2.  57.   0.   1.   0.   0.   0.  62.   1.   0.   0.  12.  34.   0.]
 [  0.   1.   1.   3.   3.   0.   2.  21.   0.   0.   0.   0.   8.  29.   4.   0.   0.   0.   0.   0. 118.   1.  22.   0.   2.   0.]
 [  0.   4.   0.   0.   0.   0.   0.   5.   0.   0.   0.   0.   3.  38.   0.   2.   1.   0.   0.   3.  22.  12.  78.   0.   0.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   4.   0.   0.   0.   0.  16.  80.   0.   0.   0.   0.   0.   0.   0.   0.  67.   0.   0.   0.]
 [  0.  15.  19.  17.  57.   0.   7.   1.   0.   0.   0.   0.   0.   0.   0.   1.   1.   0.   0.   4.  10.   5.   0.  45.   4.   0.]
 [  0.   6.   0.   0.   3.   0.   0.   1.   0.   0.   0.   0.   2.   1.   0.   5.   1.   0.   0.   7.  14.  21.  38.   0.  84.   0.]
 [  4.   0.   0.   3. 134.   0.   0.   0.   0.   2.   0.   2.   0.   0.   0.   5.  17.   0.   2.   0.   0.   4.   0.  21.   0.   0.]
```

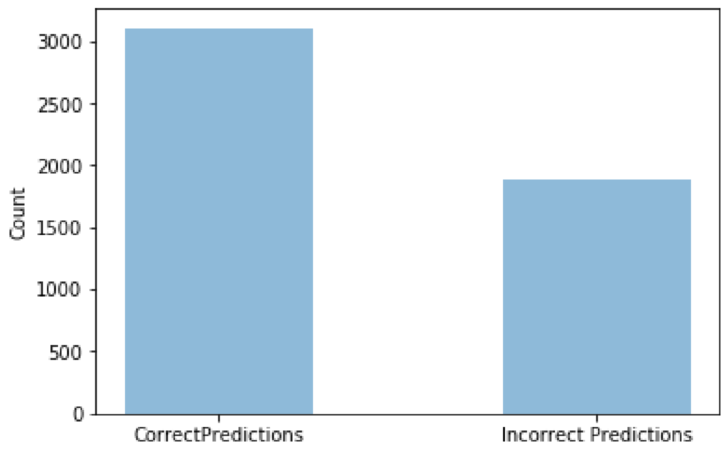


Bar plot of number of Correct and Incorrect Predictions

Size of Training data: 1000
Accuracy: 62.14%
Confusion Matrix:

Predicted: on x-axis
Actual: on y-axis

```
[144.   0.   0.   2.   0.   0.   1.   0.   0.   7.   0.   2.   6.   3.   2.   0.  11.  16.   5.   4.   1.   1.   0.   1.   0.   0.]
[  0.  90.   0.  24.   0.   1.   0.   0.   2.   0.   0.   0.   0.   0.   4.   2.  11.  26.  13.   0.   0.   0.   0.   0.   0.   0]
[  0.   0. 124.   0.   2.   0.   0.   0.   0.   0.   5.   2.   3.   0.   9.   0.   5.  11.   0.   0.   9.   1.   0.   0.   0.   0.]
[  0.   2.   0. 194.   0.   0.   0.   0.   0.   0.   0.   0.   0.   5.   9.   0.   0.   3.   0.   2.   0.   0.   0.   1.   0.   0.]
[  0.   8.  62.   2.  14.   0.   0.   0.  11.   0.   0.   0.   0.   0.   0.   7.   7.  27.   3.   4.   0.   0.   0.  46.   0.   0.]
[  0.   5.   0.   5.   0. 111.   0.   0.   0.   1.   0.   0.   0.   3.   2.  23.   0.   7.  10.  24.   0.   0.   1.   2.   0.   0]
[  0.   1.  60.   2.   0.   0.   6.   0.   0.   0.   0.   2.   2.   0.  12.   3.  81.  25.   6.   0.   2.   5.   1.   0.   0.   0.]
[  0.   0.   0.  16.   0.   3.   1.   8.   1.   2.   0.   0.   3.   8.  66.   0.   6.  47.   1.   2.   5.   3.   0.   4.   2.   0.]
[ 1.  0.  0.  8.  0.  4.  0.  0. 154.  4.  0.  1.  0.  0.   0.  0.  13.  0.  18.  0.  0.  0.  0.  2.  0.  0.]
[ 0.  0.  0.  5.  0.  1.  0.  0.  8. 143.  0.  0.  0.  0.   2.  0.  12.  0.  10.  1.  0.  0.  0.  1.  0.  0.]
[ 0.  0.  31.  1.  0.  0.  1.  0.  1.  0.  15.  5.  0.  0.   2.  0.  7.  75.  0.  0.  15.  0.  0.  24.  0.  0]
[ 0.  1.  7.  5.  1.  0.  5.  0.  0.  0.  0. 141.  0.  2.   2.  0.  12.  4.  0.  6.  11.  0.  0.  8.  0.  0.]
[ 0.  0.  0.  2.  0.  0.  0.  0.  0.  0.  0.  0. 167.  1.   0.  1.  0.  6.  0.  0.  1.  0.  9.  0.  0.  0.]
[ 0.  0.  0.  9.  0.  1.  0.  0.  0.  0.  0.  24. 127.  11.  0.  0.  8.  0.  4.  6.  4.  4.  0.  0.  0.]
[ 0.  0.  0.  12.  0.  0.  0.  0.  0.  1.  0.  0.  2.  0.  137.  1.  11.  4.  0.  1.  2.  0.  10.  0.  0.  0]
[ 0.  3.  0.  12.  0.  5.  0.  0.  3.  1.  0.  0.  0.  0.   3. 151.  5.  8.  2.  6.  0.  5.  1.  0.  2.  0.]
[ 2.  4.  1.  3.  7.  0.  0.  0.  0.  1.  0.  5.  0.  0.  27.  2. 149.  2.  4.  0.  0.  5.  3.  1.  1.  0.]
[ 0.  2.  1.  22.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.   3.  0.  5. 171.  0.  1.  0.  0.  0.  3.  0.  0.]
[ 0.  10.  0.  6.  0.  1.  0.  0.  10.  1.  0.  10.  0.  0.   8.  0.  11.  2. 128.  1.  1.  0.  0.  6.  2.  1]
[ 0.  0.  1.  3.  0.  1.  0.  0.  1.  0.  0.  0.  0.  0.   7.  4.  0.  4.  7. 150.  0.  0.  0.  4.  0.  2.]
[ 0.  0.  0.  3.  0.  0.  0.  0.  0.  0.  0.  0.  5.  7.  14.  0.  1.  0.  0.  0. 179.  0.  6.  0.  0.  0.]
[ 0.  0.  0.  2.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.   1.  0.  0.  7.  0.  5.  7. 136.  9.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  8.  4.   0.  0.  0.  5.  0.  0.  2.  5. 143.  0.  0.  0.]
[ 0.  0.  3.  8.  0.  0.  0.  0.  1.  1.  0.  0.  0.  0.   7.  1.  18.  1.  10.  0.  3.  0.  0. 133.  0.  0.]
[ 0.  1.  0.  2.  0.  11.  0.  0.  3.  0.  0.  0.  2.  0.   2.  4.  6.  0.  3.  44.  3.  20.  0.  7.  75.  0.]
[ 0.  5.  0.  5.  1.  0.  0.  0.  5.  15.  0.  2.  0.  0.   0.  0.  4.  0.  26.  2.  0.  0.  0.  12.  0. 117.]
```

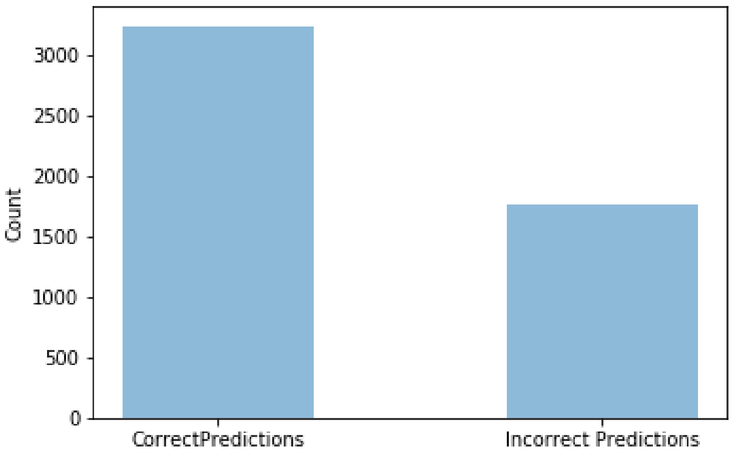


Bar plot of number of Correct and Incorrect Predictions

Size of Training data: 2000
Accuracy: 64.68%
Confusion Matrix:

Predicted: on x-axis
Actual: on y-axis

```
[166.  0.  0.  1.  0.  0.  1.  0.  0.  4.  2.  0.  3.  0.  2.  0.  1.  6.  11.  1.  0.  6.  1.  0.  1.  0.]
[ 0. 119.  0.  3.  0.  0.  0.  0.  8.  0.  0.  0.  0.  0.  0.  14.  0.  16.  11.  0.  0.  2.  0.  0.  0.  0.]
[ 0.  1. 107.  0.  28.  0.  7.  0.  0.  0.  8.  5.  1.  0.  2.  1.  0.  1.  0.  1.  5.  1.  3.  0.  0.  0.]
[ 0.  19.  0. 158.  9.  0.  1.  0.  2.  2.  0.  1.  2.  0.   0.  2.  0.  2.  1.  6.  2.  7.  0.  1.  0.  1.]
[ 0.  1.  2.  0. 155.  0.  3.  0.  4.  0.  2.  1.  0.  0.   0.  5.  0.  12.  4.  1.  0.  0.  0.  1.  0.  0.]
[ 0.  5.  0.  4.  4.  42.  0.  0.  0.  1.  0.  0.  0.  0.   0.  92.  0.  2.  5.  29.  0.  5.  3.  0.  2.  0.]
[ 0.  12.  51.  0.  16.  0.  69.  0.  5.  0.  3.  7.  2.  0.  2.  2.  11.  11.  7.  0.  0.  10.  0.  0.  0.  0.]
[ 1.  1.  1.  15.  3.  0.  3.  54.  2.  0.  8.  0.  2.  1.  6.  0.  2.  44.  3.  2.  6.  19.  0.  2.  3.  0.]
[  1.   0.   0.   3.   4.   2.   0.   0. 164.   2.   0.   3.   0.   0.   0.   3.   0.   0.  18.   1.   0.   0.   0.   2.   0.   2.]
[  1.   2.   0.   2.   2.   1.   0.   1.  18. 140.   0.   0.   0.   0.   1.   0.   0.   0.  15.   0.   0.   0.   0.   0.   0.   0.]
[  0.   4.   7.   0.  16.   0.   1.   0.   0.   0. 104.   3.   0.   0.   0.   0.   0.  32.   2.   0.   3.   2.   0.   3.   0.   0.]
[  0.   1.   1.   5.  10.   0.  12.   0.   5.   0.   0. 154.   0.   0.   0.   0.   0.   3.   7.   5.   0.   0.   0.   2.   0.   0.]
[  0.   1.   0.   2.   0.   0.   0.   0.   0.   0.   2.   0. 140.   0.   0.   1.   0.   4.   0.   0.   3.   1.  33.   0.   0.   0.]
[  2.   0.   0.   8.   0.   0.   0.  16.   0.   0.   9.   0.   9.  56.   3.   2.   0.   9.   0.   3.   9.  29.  43.   0.   0.   0.]
[  1.   0.   9.  13.   0.   0.   3.  15.   0.   1.   0.   1.   0.   0.  92.  11.   1.   8.   4.   4.   1.   5.  12.   0.   0.   0.]
[  0.   3.   0.   2.   0.   0.   1.   0.   1.   0.   1.   0.   0.   0.   0. 175.   1.   5.   0.   1.   1.  10.   2.   0.   4.   0.]
[  3.   6.   0.   1.   9.   0.  16.   1.  10.   4.   0.   5.   0.   0.  22.   3. 103.   3.  10.   0.   0.  15.   2.   1.   3.   0]
[  0.  26.   0.   3.   3.   0.   0.   0.   1.   0.   2.   0.   0.   0.   0.   4.   0. 164.   0.   2.   0.   1.   0.   2.   0.   0.]
[  1.   9.   0.   0.   8.   0.   1.   0.   9.   0.   0.  13.   0.   0.   2.   5.   0.   0. 127.   1.   0.   1.   0.   1.   8.  12.]
[  0.   0.   0.   0.   3.   0.   1.   0.   0.   0.   0.   1.   0.   0.   0.   5.   0.   6.   9. 131.   0.  24.   0.   1.   1.   2.]
[  0.   0.   6.   2.   0.   0.   0.   2.   0.   0.   0.   0.   2.   0.  12.   0.   0.   0.   0.   2. 156.  19.  14.   0.   0.   0.]
[  0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   3.   0.   5.   0.   0.   0. 149.  10.   0.   0.   0.]
[  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   5.   0.   0.   0.   0.   3.   0.   0.   0.   9. 150.   0.   0.   0.]
[  0.   0.   0.   6.  20.   0.   1.   0.  11.   1.   4.   0.   0.   0.   1.   1.   2.   0.  19.   1.   1.   1.   0. 113.   4.   0.]
[  0.   3.   0.   1.   0.   0.   0.   0.   0.   1.   0.   1.   0.   0.   0.   2.   5.   0.   0.  14.   0.  41.   2.   0. 113.   0.]
[  0.   1.   0.   0.  25.   0.   0.   0.   1.  14.   0.   2.   0.   0. 0.   0.   0.   0.  15.   3.   0.   0.   0.   0.   0. 133]
```

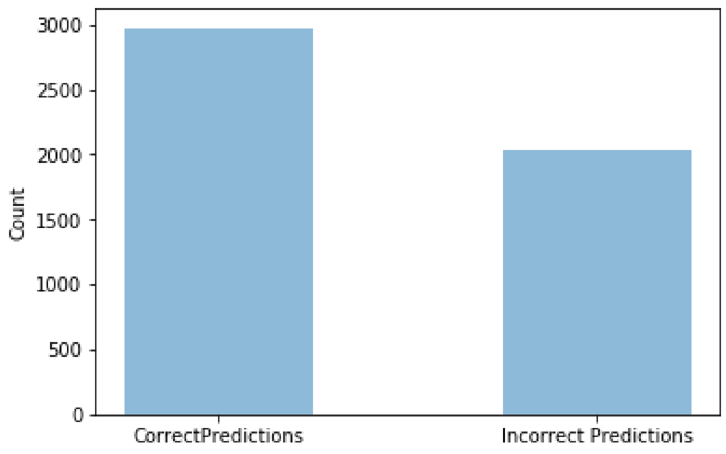


Bar plot of number of Correct and Incorrect Predictions

Size of Training data: 5000
Accuracy: 59.44%
Confusion Matrix:

Predicted: on x-axis
Actual: on y-axis

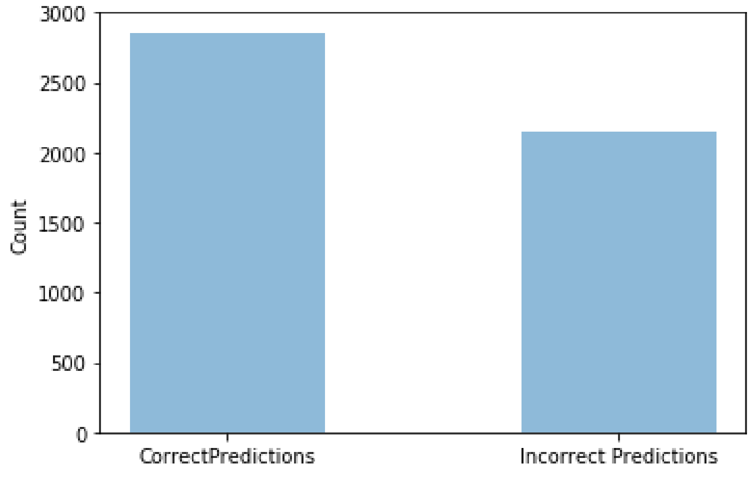
```
[184.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   2.   0.   0.   0.  10.   0.   5.   0.   2.   0.   0.   0.   3.   0.]
[  2.  57.   2.   1.   0.   3.   0.   4.   1.   0.   0.   0.   0.   0.   0.   1.  70.   0.  28.   1.   0.   0.   0.   3.   0.   0.]
[  2.   0. 143.   0.   0.   0.   2.   0.   0.   0.   2.   0.   2.   0.   0.   0.  12.   0.   0.   1.   7.   0.   0.   0.   0.   0.]
[  1.   9.   0. 132.   0.   0.   0.  13.   0.   1.   0.   0.   4.   1.   9.   2.  13.   0.   1.  18.   3.   0.   0.   6.   0.   3]
[  0.   1.  86.   0.  11.   2.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  40.   0.   9.   3.   0.   0.   0.  35.   0.   4.]
[  7.   0.   9.   1.   0. 112.   1.   7.   0.   0.   0.   0.   1.   2.   0.  16.  13.   2.   2.  19.   0.   0.   0.   1.   1.   0]
[  2.   0.  85.   0.   0.   1.   0.   1.   0.   0.   0.   0.   1.   0.   1.   0. 106.   0.   4.   0.   0.   7.   0.   0.   0.   0.]
[ 15.   0.   5.   2.   0.   1.   0.  91.   0.   0.   0.   0.   5.   7.   3.   0.  22.   2.   0.   1.  12.   7.   0.   3.   2.   0.]
[  2.   0.  13.   0.   0.   8.   0.   0. 122.   1.   0.   0.   0.   0.   0.   1.  29.   0.  10.   2.   0.   0.   0.  15.   0.   2]
[ 21.   0.   1.   1.   0.   4.   0.   1.   1. 117.   0.   0.   0.   0.   0.   1.  30.   0.   3.   0.   0.   0.   0.   3.   0.   0.]
[ 10.   1.  84.   0.   0.   0.   0.   1.   0.   0.   9.   1.   1.   1.   0.   0.  23.  10.   0.   0.   9.   0.   0.  27.   0.   0.]
[ 12.   0.  17.   3.   0.   0.   0.   0.   0.   0.   0. 126.   1.   0.   0.   0.  26.   0.   5.   5.   5.   0.   0.   5.   0.   0.]
[  1.   0.   0.   1.   0.   0.   0.   2.   0.   0.   0.   0. 171.   7.   0.   0.   3.   1.   0.   0.   0.   1.   0.   0.   0.   0.]
[  8.   0.   0.   1.   0.   0.   0.  12.   0.   0.   0.   0.  33. 130.   1.   0.   2.   0.   0.   1.   7.   3.   0.   0.   0.   0.]
[  0.   0.   9.   2.   0.   0.   0.  37.   0.   0.   0.   0.   8.   0.  35.   1.  81.   0.   0.   4.   1.   3.   0.   0.   0.   0.]
[  0.   0.   1.   1.   0.  16.   0.   4.   0.   1.   0.   0.   0.   0.   3. 157.  13.   0.   0.   2.   0.   2.   0.   0.   7.   0.]
[  4.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0. 198.   0.  13.   1.   0.   0.   0.   0.   0.   0.]
[ 10.   4.   5.   2.   0.   0.   0.  19.   0.   1.   1.   0.   2.   1.   3.   0.  74.  79.   1.   4.   0.   0.   0.   2.   0.   0.]
[  1.   3.   2.   0.   0.   4.   0.   1.   0.   0.   0.   2.   0.   0.   0.   1.  40.   0. 112.   2.   0.   0.   0.   8.   9.  13.]
[  0.   0.   2.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   2.   6.   0.   4. 155.   6.   0.   0.   3.   2.   3.]
[  1.   0.   5.   0.   0.   0.   0.   1.   0.   0.   0.   0.  11.   2.   0.   0.  10.   0.   0.   0. 182.   3.   0.   0.   0.   0.]
[  8.   0.   0.   0.   0.   1.   0.   2.   0.   0.   0.   0.   3.   0.   0.   0.   7.   0.   0.   0.   1. 142.   1.   0.   3.   0.]
[  1.   0.   0.   0.   0.   0.   0.   2.   0.   0.   0.   0.  23.  26.   0.   0.   0.   0.   0.   0.   1.  22.  92.   0.   0.   0.]
[  1.   0.   5.   3.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.  32.   0.   3.   2.   3.   0.   0. 133.   2.   1.]
[  4.   0.   0.   0.   0.   4.   0.   1.   0.   0.   0.   0.   1.   0.   0.   1.  15.   0.   0.  14.   1.   7.   0.   4. 131.   0.]
[  0.   0.   0.   0.   0.   1.   0.   0.   0.   1.   0.   2.   0.   0.   0.   0.  11.   0.  16.   4.   0.   0.   0.   8.   0. 151.]
```



Bar plot of number of Correct and Incorrect Predictions

```
Size of Training data:  10000
Accuracy:  57.14%
Confusion Matrix:

Predicted: on x-axis
Actual: on y-axis
[179.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   3.   0.   0.   0.   0.   1.  13.   1.   1.   0.   1.   3.   3.   0.]
[  1. 136.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   0.   4.  29.   0.   0.   0.   0.   2.   0.   0.]
[  0.   1. 132.   0.   0.   0.   2.   2.   0.   0.   8.   4.   9.   0.   0.   1.   1.   0.   6.   1.   1.   0.   1.   2.   0.   0.]
[  1. 149.   0.  23.   0.   0.   0.   4.   0.   0.   0.   0.   7.   1.   0.   3.   0.   0.  16.   8.   0.   0.   0.   4.   0.   0.]
[  2.  19.  60.   0.   0.   1.   1.   0.   0.   0.   3.   2.   0.   0.   0.   0.   1.   9.  35.   1.   0.   0.   0.  57.   0.   0.]
[  1.  11.  11.   0.   0.  64.   0.   0.   0.   0.   0.   0.   0.   0.   0.  58.   0.   2.  17.  25.   0.   0.   3.   2.   0.   0.]
[  4.  11.  84.   0.   0.   0.   9.   1.   0.   0.   3.   6.   2.   0.   0.   9.  28.   6.  39.   0.   0.   0.   2.   4.   0.   0.]
[  4.  15.   4.   0.   0.   0.   1.  79.   0.   0.   2.   0.  19.   5.   0.   1.   1.   7.   7.   6.   1.   3.   1.  22.   0.   0.]
[ 0.  5.  2.  0.  0.  4.  0.  0. 147.  0.  0.  0.  0.  0.  0.  0.  0.  0. 37.  1.  0.  0.  0.  9.  0.  0.]
[22.  2.  1.  0.  0.  1.  0.  0. 39. 65.  0.  0.  0.  0.  0.  0.  7.  0. 44.  0.  0.  0.  0.  2.  0.  0.]
[ 2. 11. 14.  0.  0.  0.  1.  0.  0.  0. 72.  4.  3.  0.  0.  0.  0. 12.  3.  0.  3.  0.  0. 52.  0.  0.]
[ 0.  1.  2.  3.  0.  0.  3.  1.  0.  0.  0. 150.  1.  0.  0.  0.  0.  0. 28.  2.  3.  0.  0. 11.  0.  0.]
[ 0.  5.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0. 178.  0.  0.  0.  0.  2.  0.  0.  0.  0.  1.  0.  0.  0.]
[ 1.  0.  1.  0.  0.  1.  0. 12.  0.  0.  0.  0. 84. 80.  0.  2.  0.  2.  0.  5.  0.  0.  9.  1.  0.  0.]
[ 1. 18. 13.  1.  0.  0.  2. 88.  0.  0.  0.  0.  4.  0.  1.  7. 10.  2. 18.  5.  1.  0.  7.  3.  0.  0.]
[ 0.  8.  0.  0.  0.  5.  0.  1.  1.  0.  1.  0.  1.  0.  0. 175.  0.  1.  2.  4.  0.  0.  1.  0.  7.  0.]
[ 6.  7.  1.  0.  0.  0.  0.  1.  1.  0.  0.  2.  0.  0.  0.  4. 145.  1. 36.  1.  0.  0.  2. 10.  0.  0.]
[ 3. 59.  1.  0.  0.  0.  0.  1.  0.  0.  5.  1.  1.  1.  0.  0.  0. 122.  2.  1.  0.  0.  0. 11.  0.  0.]
[ 1. 10.  1.  0.  0.  0.  0.  0.  0.  0.  0.  4.  0.  0.  0.  3.  1.  0. 173.  0.  0.  0.  0.  4.  1.  0.]
[ 0.  3.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  2.  0.  0.  4.  0.  1. 12. 153.  1.  0.  0.  5.  0.  2.]
[ 1.  1. 11.  0.  0.  0.  0. 10.  0.  0.  1.  0. 12.  0.  0.  0.  0.  0.  0.  1. 168.  0.  8.  2.  0.  0.]
[ 1.  8.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  8.  0.  0.  1.  1.  0. 129. 15.  4.  1.  0.]
[ 1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 14.  0.  0.  0.  0.  0.  0.  0.  0.  0. 151.  0.  0.  0.]
[ 1.  2.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. 23.  0.  1.  0.  0. 159.  0.  0.]
[ 2.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  2.  0.  0. 10.  6.  0. 11. 20.  0.  4.  0. 23. 105.  0.]
[ 0.  2.  0.  0.  0.  0.  0.  0.  0.  1.  0.  2.  0.  0.  0.  1.  0.  0. 116.  1.  0.  0.  0.  9.  0. 62.]
```

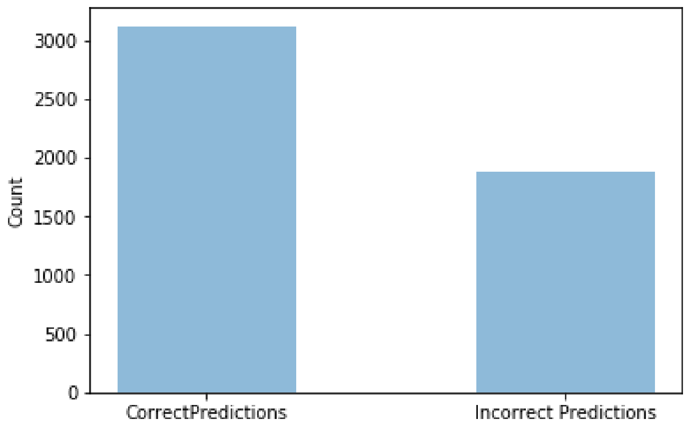


Bar plot of number of Correct and Incorrect Predictions

Size of Training data: 15000
Accuracy: 62.38%
Confusion Matrix:

Predicted: on x-axis
Actual: on y-axis

```
[172.  1.  0.  0.  0.  0.  0.  0.  0.  1.  1.  0.  1.  1.  9.  0.  4.  1.  4.  2.  1.  0.  0.  3.  3.  2.]
[ 0. 142.  0.  0.  0.  3.  0.  0.  2.  0.  0.  0.  0.  0. 19.  0.  0.  6.  0.  1.  0.  0.  0.  0.  0.  0.]
[ 0.  0. 111.  0. 15.  0.  4.  0.  0.  0.  5.  2.  2.  0. 24.  0.  2.  1.  0.  2.  3.  0.  0.  0.  0.  0.]
[ 0. 91.  0. 27.  1.  0.  0.  0.  1.  0.  0.  0.  0.  5. 71.  0.  0.  0.  0. 16.  1.  0.  0.  2.  0.  1.]
[ 0.  5.  1.  0. 155.  0.  4.  0.  0.  0.  0.  0.  0.  0.  2.  0.  3.  7.  1.  5.  0.  0.  0.  8.  0.  0.]
[ 0.  4.  0.  0.  4. 133.  3.  0.  0.  0.  0.  0.  0.  2.  2.  0.  0.  2.  0. 42.  0.  0.  1.  1.  0.  0.]
[ 0.  6. 35.  0.  7.  3. 49.  0.  1.  0.  0.  5.  1.  0. 67.  0. 24.  6.  3.  0.  0.  0.  0.  1.  0.  0.]
[  0.   2.   0.   0.   0.   4.   1.  40.   0.   0.   0.   0.   3.  16.   82.   0.   4.   9.   0.   4.   3.   3.   0.   5.   2.   0.]
[  0.   5.   0.   0.   3.   6.   0.   0. 175.   0.   0.   1.   0.   0.    2.   0.   0.   0.   1.   2.   0.   0.   0.   9.   0.   1.]
[  5.   2.   0.   0.   1.   8.   0.   0.  47.  92.   0.   0.   0.   0.   13.   0.   6.   0.   0.   3.   0.   0.   0.   5.   0.   1.]
[  0.   4.  16.   0.  40.   0.   7.   0.   0.   0.  37.   0.   1.   1.  13.   0.   2.  23.   0.   3.   3.   0.   0.  27.   0.   0]
[  0.   2.   4.   3.  11.   0.   9.   0.   1.   0.   0. 136.   0.   0.   7.   0.   4.   0.   5.   7.   2.   0.   0.  14.   0.   0.]
[  0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 169.   9.    5.   0.   0.   2.   0.   0.   0.   0.   1.   0.   0.   0.]
[  0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   2. 165.   18.   0.   0.   3.   0.   7.   2.   0.   0.   0.   0.   0.]
[  0.   0.   0.   0.   0.   1.   0.   1.   0.   0.   0.   0.   2.   5.  167.   0.   2.   0.   0.   0.   0.   0.   1.   1.   0.   1.]
[  0.  10.   0.   0.   0.  58.   0.   0.   2.   0.   0.   0.   0.   1.   26.  93.   2.   1.   0.   5.   0.   0.   1.   0.   8.   0.]
[  4.  17.   0.   0.   9.   1.   0.   1.   0.   0.   0.   4.   0.   0.   69.   0. 109.   0.   1.   1.   0.   0.   0.   1.   0.   0.]
[  0.  26.   0.   0.   2.   0.   0.   1.   0.   0.   1.   0.   1.   1.   31.   0.   0. 140.   0.   3.   0.   0.   0.   2.   0.   0.]
[  1.  37.   0.   0.  14.   0.   0.   0.  25.   0.   0.   7.   0.   0.   7.   0.   3.   2.  45.   9.   0.   0.   0.  17.   5.  26.]
[  0.   0.   0.   0.   2.   0.   0.   0.   2.   0.   0.   0.   0.   0.   11.   0.   0.   0.   0. 166.   0.   0.   0.   1.   0.   2.]
[  1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   7.   5.   40.   0.   0.   0.   0.   3. 159.   0.   0.   0.   0.   0.]
[  0.   0.   0.   0.   0.   3.   0.   0.   0.   0.   0.   0.   0.   2.   17.   1.   0.   1.   0.   9.   1. 105.   3.   1.  25.   0.]
[  0.   0.   0.   0.   0.   0.   0.   2.   0.   0.   0.   0.  11.  33.    4.   0.   0.   0.   0.   0.   0.   0. 117.   0.   0.   0.]
[  0.   3.   0.   0.  10.   0.   0.   0.   7.   0.   0.   0.   0.   0.    7.   0.   2.   1.   0.   8.   0.   0.   0. 144.   4.   0.]
[  0.   0.   0.   0.   0.   2.   0.   0.   1.   0.   0.   0.   0.   0.    4.   0.   9.   0.   0.  28.   1.   0.   0.   4. 134.   0.]
[  0.   6.   0.   0.  17.   2.   0.   0.   7.   1.   0.   2.   0.   0.    0.   0.   1.   1.   7.   7.   0.   0.   0.   6.   0. 137.]
```



Bar plot of number of Correct and Incorrect Predictions

A trend we noticed in the pocket algorithm was that the number of iterations did matter. The ideal number of iterations based on trial and error were found to be between 150 and 200.

Besides, the accuracy did not go over 64% for the algorithm.

Some of the key labels that algorithm misclassified were corresponding to alphabets D, G, H, J, K, P and S. D was confused for B and O. G was mistaken for C, O and Q. H was misclassified as O. J was confused with I. K was confused with E, R and W. P was wrongly classified as F. And finally, S was misclassified as B, I and Z.

With the exception of H, it was understandable why the algorithm mis-labelled the others.

Also, another noticeable thing about the pocket algorithm was that it was noticeably way faster than the kNN algorithm. This was because the kNN is a lazy algorithm and does not do any actual learning. It iterates over the entire training data for each test point thereby increasing the runtime considerable.

Since the pocket algorithm is a learning algorithm, once it is trained, it does not need to refer to the training data again and again. Thus, the runtime is significantly smaller.