Project Report

AI 520

Vicky Prakash(vp407), Ananya Jana(aj611)

Question 2:

Feature Models:

1. Counting hash[f1]: We are counting the number of hash in each data point.
2. Counting plus[f2]: We are counting the number of plus in each data set.
3. Counting White Space[f3]: We are counting the number of white spaces (blank spaces) in the modified data points with reduced numbers of white spaces.
4. Active pixels in top half[f4]: We are counting the density of active pixels (plus and hash) in the top half of each data set.
5. Active pixels in bottom half[f5]: We are counting the density of active pixels in the bottom half of each data set.
6. Active pixels in right half[f6]: We are counting the density of active pixels in the right half of each data set.
7. Active pixels in left half[f7]: We are counting the density of active pixels in the left half of each data set.
8. Aspect Ratio[f8]: We are counting the ratio of the length and width of active pixels for each data set.
9. Connected Components[f9]: We are finding the number of connected components in each data point. (e.g.: 8 has 3 connected components.)

We have experimented with all possible subsets of the feature set for each of the algorithms. For this purpose, we have used validation data provided. After training the models with training data, we used the validation data for feature selection. In this way, we identified features for each of the algorithms and the final accuracy is computed on the testing data which has not been used in any part for training.

The features used in each of the algorithm are:

Algorithm/Program	Features Used
Naïve Bayes Digit	f1, f2, f3, f4, f7, f8
Perceptron Digit	f1, f2, f4, f8
K-Nearest Neighbor Digit	f1, f2, f3, f4
Naïve Bayes Face	f1, f4, f7, f8
Perceptron Face	f1
K-Nearest Neighbor Face	f1, f5, f8

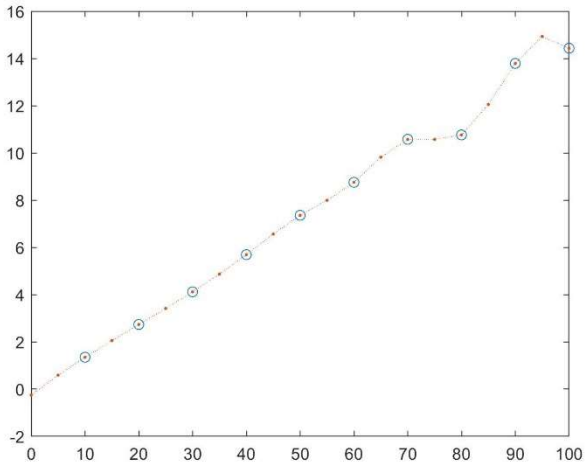
Question 4:

Digit Prediction:

Naïve Bayes:

Data Percent	*Data Size*	*Accuracy(%)*	*Prediction Error*	*Time Taken*
10	500	30.2	69.8	1.3482
20	1000	32.7	67.3	2.7355
30	1500	32.6	67.4	4.1187
40	2000	34.2	65.8	5.6926
50	2500	34.7	65.3	7.36
60	3000	34	66	8.7568
70	3500	35	65	10.5767
80	4000	34.7	65.3	10.7643
90	4500	34.2	65.8	13.7917
100	5000	35.1	64.9	14.4338

For Time taken as a function of number of data points, I have used Matlab to find the interpolated graph and the interpolated function. The graph is:

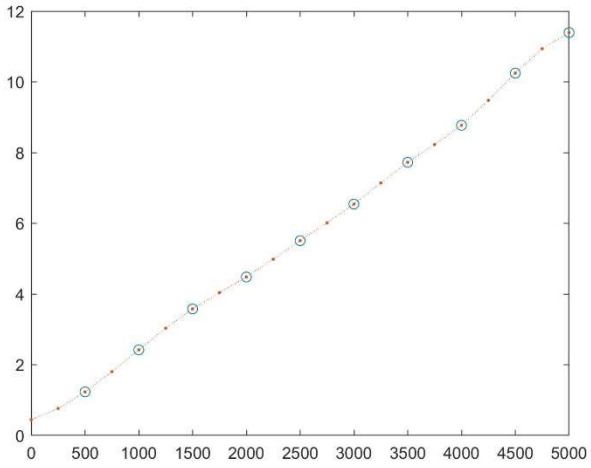


The function is y = 0.0030x -0.2245, where y is the time taken by the algorithm and x is the number of data points. (The co-efficient of larger degree powers were very small, e.g. the co-efficient of x^2 is (-0.000000006237879), so, we have neglected them)

Perceptron:

Data Percent	*Data Size*	*Accuracy*	*Prediction Error*	*Time Taken*
10	500	13.4	86.6	1.2321
20	1000	13.8	86.2	2.4209
30	1500	22.4	77.6	3.5791
40	2000	26	74	4.4817
50	2500	26.9	73.1	5.5094
60	3000	28.3	71.7	6.544
70	3500	27.2	72.8	7.7257
80	4000	27.1	72.9	8.7720
90	4500	28.2	71.8	10.2488
100	5000	31.0	69	11.3959

For Time taken as a function of number of data points, I have used Matlab to find the interpolated graph and the interpolated function. The graph is:



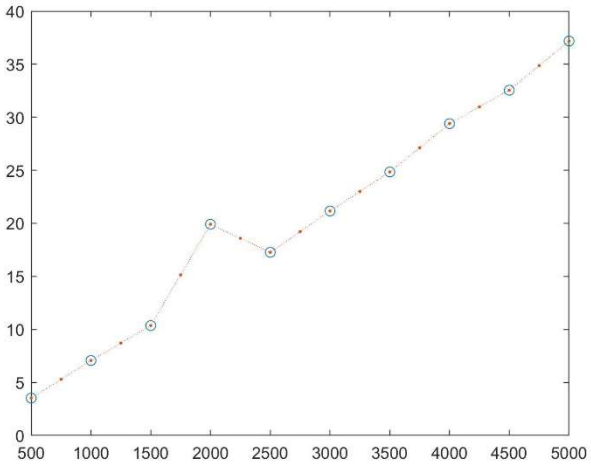
The function is y = 0.00189x + 0.4176, where y is the time taken by the algorithm and x is the number of data points. (The co-efficient of larger degree powers were very small, e.g. the co-efficient of x^2 is (-0.000000059401515), so, we have neglected them)

K-Nearest Neighbor:

Data Percent	*Data Size*	*Accuracy*	*Prediction Error*	*Time Taken*
10	500	28.2	71.8	3.5193
20	1000	29.8	70.2	7.0645
30	1500	27.1	72.9	10.3565
40	2000	29.7	70.3	19.9089
50	2500	29.9	70.1	17.2659
60	3000	31.1	68.9	21.1596
70	3500	31.2	68.8	24.8437
80	4000	30.7	69.3	29.4003
90	4500	30.5	69.5	32.552

100	5000	31.5	68.5	37.1842

The interpolated graph of Time Taken by the algorithm vs. number of datapoints is:



The function is y = 0.0075x + 0.1265, where y is the time taken by the algorithm and x is the number of data points. (The co-efficient of larger degree powers were very small, e.g. the co-efficient ofx^2 is (-0.000000064584848), so, we have neglected them)

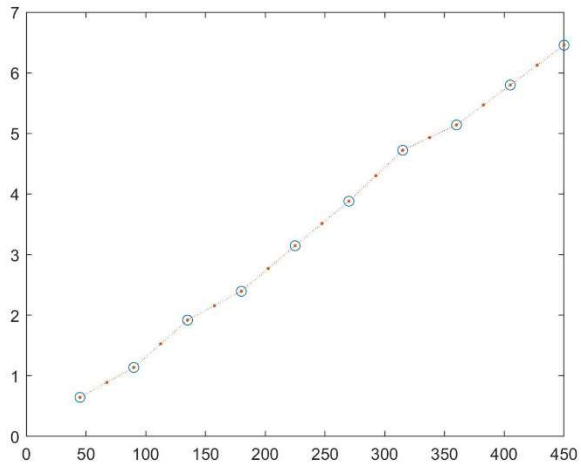
Face Prediction:

Naïve Bayes:

Data Percent	*Data Size*	*Accuracy*	*Prediction Error*	*Time Taken*
10	45	74	26	0.6447
20	90	76	24	1.1381
30	135	76	24	1.9194
40	180	76	24	2.3952
50	225	74	26	3.1469

60	270	73.33	26.67	3.8811
70	315	73.33	26.7	4.7219
80	360	74	26	5.1404
90	405	74	26	5.7979
100	451	74	26	6.456

The interpolated graph of Time Taken by the algorithm vs. number of datapoints is:



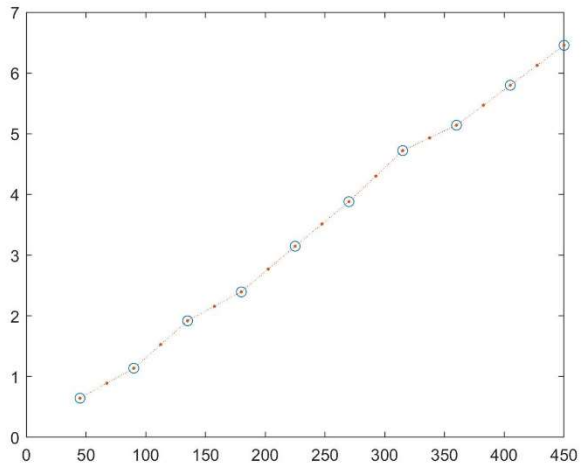
The function is y = 0.014688x - 0.10440, where y is the time taken by the algorithm and x is the number of data points. (The co-efficient of larger degree powers were very small, e.g. the co-efficient ofx^2 is (-0.000000087729143), so, we have neglected them)

Perceptron:

Data Percent	*Data Size*	*Accuracy*	*Prediction Error*	*Time Taken*
10	45	51.3	48.7	0.6563
20	90	51.3	48.7	1.2884
30	135	51.3	48.7	1.9522

40	180	51.3	48.7	2.5012
50	225	51.3	48.7	3.3190
60	270	51.3	48.7	3.7745
70	315	51.3	48.7	4.6025
80	360	51.3	48.7	5.137
90	405	51.3	48.7	5.8556
100	450	51.3	48.7	6.4020

The interpolated graph of Time Taken by the algorithm vs. number of datapoints is:



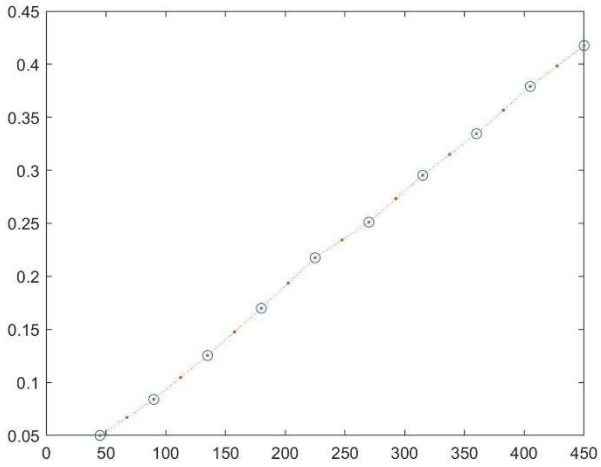
The function is y = 0.01445x -0.0080 where y is the time taken by the algorithm and x is the number of data points. (The co-efficient of larger degree powers were very small, e.g. the co-efficient of x^2 is (-0.000000255331089), so, we have neglected them)

K-Nearest Neighbor:

Data Percent	*Data Size*	*Accuracy*	*Prediction Error*	*Time Taken*
10	500	60.7	39.3	0.05
20	1000	64	36	0.084

30	1500	64.7	35.3	0.1254
40	2000	63.3	36.7	0.1699
50	2500	61.3	38.7	0.2175
60	3000	60.7	39.3	0.2511
70	3500	62	38	0.2953
80	4000	64	36	0.3345
90	4500	67.3	32.7	0.379
100	5000	66.7	33.3	0.4176

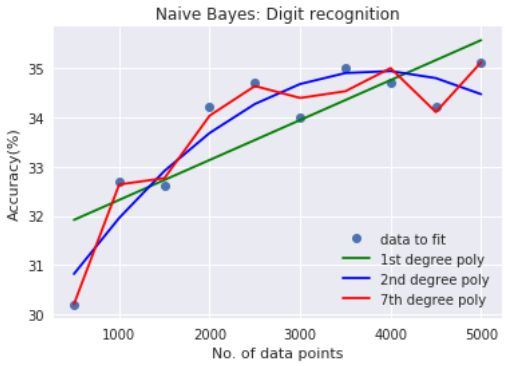
The interpolated graph of Time Taken by the algorithm vs. number of datapoints is:



The function is y = 0.000918x + 0.004948 where y is the time taken by the algorithm and x is the number of data points. (The co-efficient of larger degree powers were very small, e.g. the co-efficient ofx^2 is (0.000000003179948), so, we have neglected them)

From the above mentioned tables, graphs and interpolated equations for Naïve Bayes, Perceptron and K-Nearest Neighbor for digit and face detection, we can see that the Perceptron Algorithm is least time consuming (co-efficient of x is smallest) and hence more efficient for digit detection while K-Nearest Neighbor is most effective for Face Detection with smallest co-efficient for x(i.e., the dependency upon number of data points is lower as compared to other algorithms).

Figure 1: q3_1: Digit Recognition Naive Bayes



Q.3 Accuracy Vs Number of data points

We have calculated the accuracy for digit and face recognition with 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100% o the data points. We have then calculated the interpolation polynomials by using python functions *polyfit* and *poly1d*. This interpolation polynomial gives the relationship between accuracy and the number of data points.

Accuracy Vs No. of data points(n)
Digit
Naive Bayes
Refer to figure **q3_1**. Coefficients of Interpolation polynomial of degree 2
-3.65151515e-07 2.81803030e-03 2.95050000e+01
$Accuracy = f(n) = -3.65151515e-07 * n^2 + 2.81803030e-03 * n + 2.95050000e+01$
which is approximately equal to
$f(n) = 2.81803030e - 03 * n^2 + 2.95050000e + 01$
where n is the number of data points.

We didn't take higher degree polynomials because the coefficients of the higher order terms are very close to 0.

Perceptron
Refer to figure **q3_2**. Coefficients of Interpolation polynomial of degree 2
-1.20454545e-06 1.01122727e-02 8.21500000e+00
$Accuracy = f(n) = -1.20454545e-06 * n^2 + 1.01122727e-02 * n + 8.21500000e+00$
which is approximately equal to
$f(n) = 1.01122727e - 02 * n + 8.21500000e + 00$
where n is the number of data points.

Figure 2: q3_2: Digit Recognition Perceptron

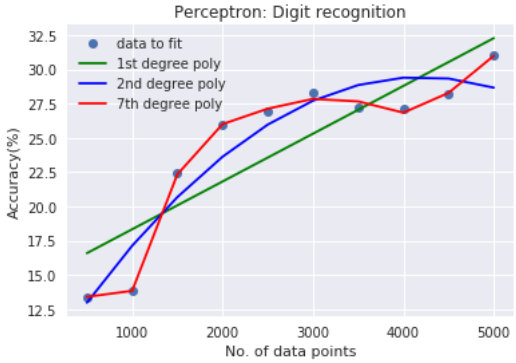
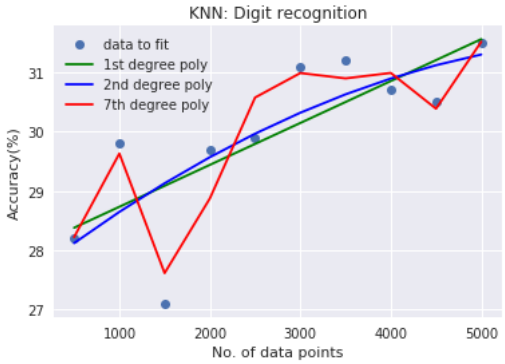


Figure 3: q3_3: Digit Recognition KNN



We didn't take higher degree polynomials because the coefficients of the higher order terms are very close to 0.

KNN

Refer to figure **q3_3**.

Coefficients of Interpolation polynomial of degree 2

-8.63636364e-08 1.18166667e-03 2.75516667e+01

$Accuracy = f(n) = -8.63636364e-08 * n^2 + 1.18166667e-03 * n + 2.75516667e+01$

which is approximately equal to

$f(n) = 1.18166667e-03 * n + 2.75516667e+01$

where n is the number of data points.

Perceptron

Figure 4: q3_4: Face Recognition Naive Bayes

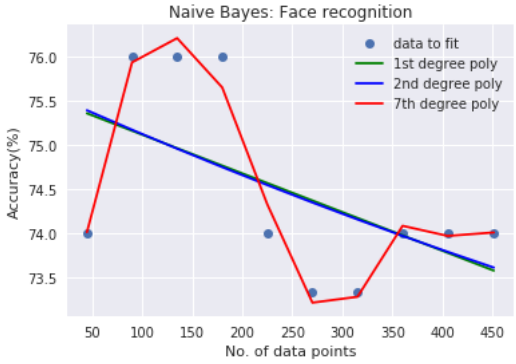
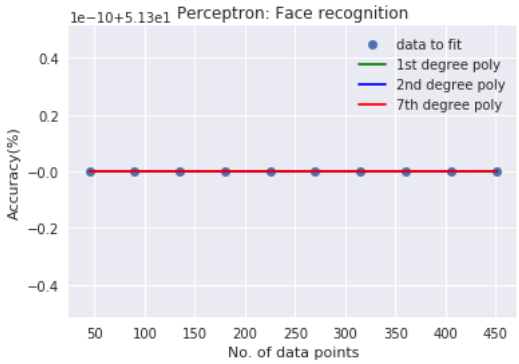


Figure 5: q3_5: Face Recognition Perceptron



Face

Naive Bayes

Refer to figure **q3_4**.

In this case we take a polynomial of degree 1, because in the given interval, the 2nd degree polynomial and 1st degree polynomial almost overlap.

$Accuracy = f(n) = -4.39346169e-03 * n + 7.55538211e+01$

where n is the number of data points.

Perceptron

Refer to figure **q3_5**.

In this case,we can clearly see we have a polynomial of degree 0.

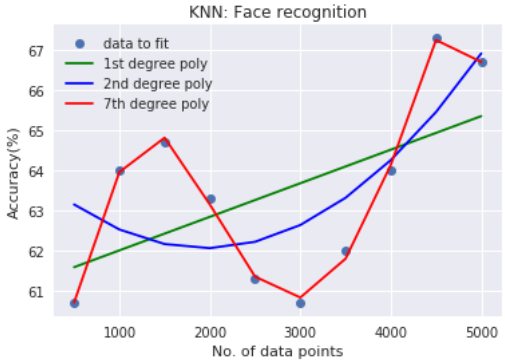
$Accuracy = f(n) = 5.13000000e+01$

where n is the number of data points.

KNN

Refer to **q3_6**

Figure 6: q3_6: Face Recognition KNN



$Accuracy = f(n) = -2.02909091e-03 * n + 6.40333333e+01$

Q.5 Algorithm Discussion

Perceptron

Digit Recognition(Multiclass Perceptron)

Perceptron classifier is a very basic linear classifier i.e. linear classification algorithm which makes its predictions based on a linear predictor function which combines the feature vectors with a set of weights. The main idea is to refine weights when the prediction goes wrong. Since this is a linear classifier, it works well when the data points are linearly separable.

The basic version of Perceptron is nothing but a binary clasifier. But, we have total 10 digits to be recognized. So, we have used the multiclass Perceptron classifier. Multiclass Perceptron classifier is an extension of the linear classifier. Here we use multiple separating hyperplanes(and hence weights) instead of just 1 in binary classifier. We have as many hyperplanes as the number of classees i.e. 10

Preprocessing

We preprocess the raw data to extract the set of feature vectors. Those features from the training data set are in *X_TrainFea*. The process of extracting features is described in detail in **Q.2**.

Training

In our program we have initially generated random weights in the range $[0, 1]$ using the function *numpy.random.rand*. Then we take one by one the entries from *X_TrainFea* which contains the all the features corresponding to a particular data point and calculate scores for different classes. The scores are calculated using the following formula $\sum_i w_i * f_i(x)$ where w_is are weights for the particular class and $f_i(x)$s are the features of the training data point x. The scores corresponding to the 10 classes are stored in an array calle *Scores*. Next, we

iteratein this array to find out which class gives the maximum score. The class with the maximum score is predicted to be the class of the data point.
We also have the label of the training data point x available. If it matches the predicted label, then we do not modify the weights. But if the prediction is wrong, we adjust the weights of these concerned two classes in the following manner:
i) Lower the score of the class of the predicted label by $w_i = w_i - \eta * f_i(x)$ where η is the *learning rate*.
ii) Raise the score of the class of the predicted label by $w_i = w_i + \eta * f_i(x)$.

We initialized η to an arbitrary value 0.2 at the beginning and then reduced it with evey iteration. The idea is that, as we iterate over the training data set, the weights stabilize and hence the change in weights are very small. After changing the weights we normally don't re-check the score of the training data point and hence we don't know whether the cange corrected the prediction or not. As a remedy for this we make a number of iterations called *epochs* over the entire training data set. This is to make sure the weights we generate are really fine-tuned enough.

Testing
Here we calculate the scores for a particular data point by using its feature points and the refined set of weights from training phase. Similar to training procedure, we check which class gives the maximum score and predict that class to be the class of the data point.

Face Recognition(Binary Perceptron)
We use binary Perceptron in this case because our classifier only needs to tell us whetherthe test data point is a face or is not a face.

Preprocessing
Same as above.

Training
In this case, instead of having a set of weights for each class like above, we have just one set of weights and this is sufficient to tell us whether a test data point corresponds to a face or not.
Here we calculate scores for individual training data point and see whether the score is +ve or -ve. If the score is +ve, that means, the data point corresponds to a face, else if the score is -ve, that means, the data point corresponds to a non-face.
The weight update procedure is little bit different in this case. When the prediction is correct, the weights are not changed just like in multiclass perceptron. But, if there is a wrong prediction then two types of weight update can happen:
i) A positive label is predicted to be negative label. Then, weights are increased using $w = w + \eta * f(x)$ where η is the *learning rate*.
ii)A negative label is predicted to be positive label. Then, weights are decreased

using $w = w - \eta * f(x)$.
η is refined just like the multiclass classifier.

Testing
Similar to training phase. We find whether a test data point gives score > 0 or < 0 and predict it to be a face or non-face accordingly.

Naive Bayes

The basic assumption behind Naive Bayes algorithm is that features are conditionally independent. Suppose we have a data point with feature vector as $X = (x_1, x_2, ...x_n)$ and there are total k classes as $C_1, ..., C_k$. Now, we want to calculate the probabilities, given the feature vector, what is the probability that the data point belongs to the class C_1, C_2,...C_k i.e. $P(C_k|x_1, x_2, ...x_n)$. The class for which this probability value is maximum is the predicted class for the data point. Since, the features are conditionally independent, according to Naive Bayes assumption, we have

$$P(C_k|x_1, x_2, ...x_n) = \frac{P(C_k) * \prod_{i=1}^{n} P(x_i|C_k)}{P(x_1, x_2, ...x_n)}$$
$$\implies P(C_k|x_1, x_2, ...x_n) \propto \mathrm{P}\left(\left(\right) C_k\right) * \prod_{i=1}^{n} P(x_i|C_k) P(x_1, x_2, ...x_n)$$

This is because $P(x_1, x_2, ...x_n)$ remains the same for all classes.

In our project, we have used Naive Bayes classifier for both digit and face recognition in similar fashion. However, there is one difference. in case of digit recognition, we have 10 classes and we compute 10 $P(C_k|x_1, x_2, ...x_n)$s for each test data point. In case of face recognition we have just two classes and hence we compute 2 $P(C_k|x_1, x_2, ...x_n)$s for each test data point.

Preprocessing
Same as above.

Training
We calculate the value of each prior i.e. $P(C_k)$ from the training data set.
$P(C_k) = \frac{Number\ of\ data\ points\ from\ the\ class\ C_k}{Total\ number\ of\ data\ points}$.

For calculating $P(x_i|C_k)$, we assume that the data points follow **Gaussian distribution**. We calculate the mean and standard deviation of each feature for each class using the functions *numpy.mean()* and *numpy.std()*.

Testing
When we have a new data point, we extract features from the data point and then calculate$P(x_i|C_k)$ for each feature x_i of this new data point. We used *scipy.stats.norm().pdf()* for this purpose. Once we have all the probabilities corresponding to all the classes, we check which class is giving maximum prob-

ability by using *numpy.argmax()* function. That class is to be predicted to be the class of the data point.

K Nearest Neighbor(Algorithm of our choice

In this algorithm, we treat the feature set from a data point as a point in space. We calculate the distance of this point from all the other points corresponding to the feature vector of the training points. Then we sort the distances and take the k nearest neighbor points. We count the number of time a label appears in this set. The label with the maximum count in this k nearest neighbor set is predicted to be the class of the test data point.

Preprocessing

Same as above.

Training

There is no such phase as training in k Nearest Neighbor algorithm. Every time a new data point comes, we calculate the Euclidean distance of its feature vector with every other feature vector in the training data set using the function *EuclideanDistane*. So, this method is more like an online method.

Testing

These distances calculated are stored in an array of tuples, *dist*. Then we sort the*dist*array in ascending order using the function *sorted* and pick up the top k elements. These are the nearest neighbors for our test data point.We count the number of neighbors from the same set and take the maximum. If the neighbor which has the maximum count belongs to the class C, the this is the predicted class of the test data point.

Result Discussion

We are attaching the output snippets from our code displaying the results:

Perceptron

Digit Recognition

———Digit Recognition with Perceptron———-

Number of test data points: 1000
Percentage of training data used: 100%
Number of training data points: 5000
Accuracy: 31.0%
Prediction Error: 69.0%
Training time: 11.153861045837402 seconds

Face Recognition

———Face Recognition with Perceptron———-

Number of test data points: 150
Percentage of training data used: 100%

Number of training data points: 451
Accuracy: 51.3333333333%
Prediction Error: 48.6666666667%
Training time: 6.793347358703613 seconds

Naive Bayes

Digit Recognition
Number of test data points: 1000
Percentage of training data used: 100%
Number of training data points: 5000
Accuracy: 35.1%
Prediction Error: 64.9%
Training time: 15.419532299041748 seconds

Face Recognition
———Face Recognition with Naive Bayes————-

Number of test data points: 150
Percentage of training data used: 100%
Number of training data points: 451
Accuracy: 72.6666666667%
Prediction Error: 27.3333333333%
Training time: 6.6637091636657715 seconds

K Nearest Neighbor

Digit Recognition
———Digit Recognition with K Nearest Neighbors————-

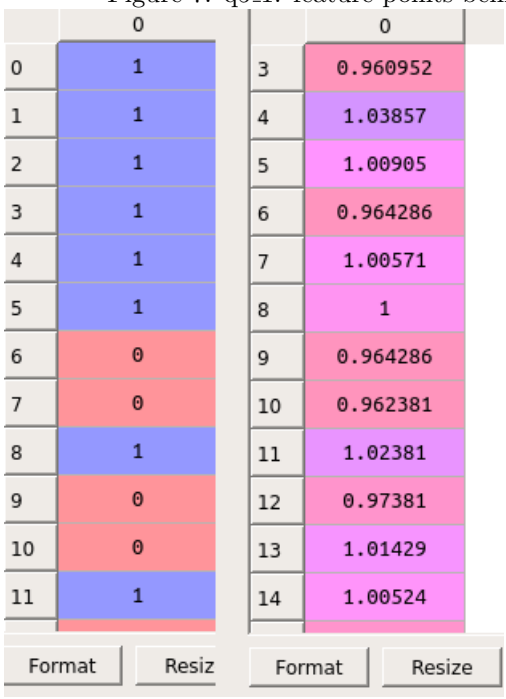
Number of test data points: 1000
Percentage of training data used: 100%
Number of training data points: 5000
Accuracy: 31.5%
Prediction Error: 68.5%
Execution time: 54.60056734085083 seconds

Face Recognition
———Face Recognition with K Nearest Neighbors————-

Number of test data points: 150
Percentage of training data used: 100%
Number of training data points: 451
Accuracy: 66.0%
Prediction Error: 34.0%
Execution time: 0.564697265625 seconds

As we can see, the Naive Bayes algorithm gives best performance fo both Digit

Figure 7: q5_1: feature points being shown in Anaconda IDE



Recognition and Face recognition.

Lessons Learnt
Our key learnings are:
1. The accuracy depends on how good the features are. We initially started with very primitive features. The first features we considered for digit recognition are x_1 = count of '#' symbols and x_2 = count of '+' symbols. With this we had accuracy $\approx$ 28%. But later when we stared considering other features like boundary, symmetry etc, we were able to improve our accuracy to 35.1%.

2. The accuracy of Perceptron model using our feature is $\approx$ 51% which is almost equivalent randomly saying *face* or *not a face*. We analyzed the data and found out the reason. When we had only one feature and the the classifier would always pass through the origin, but the feature points for both +ve and -ve samples were evenly distributed along the x-axis. This can be seen from the figure *q5_1* where labels an te feature values are shown. Hence, no matter which line we get as classifier, it was not able to separate those sets of +ve and -ve samples. Those samples were not linearly separable. We tried adding more features as well, but still we didn't get linearly separable sets.

3. We tried adding the count of blank space symbols in the feature set, but

that didnt improve our frequency. This was because of the fact that the count of blank space was not conditionally independent with the count of '+' and '#'. We can simply get the count of blanksby subtracting the sum of counts of '+' and '#' from the total number of pixels.

4. Data processing and parameter selection part is the most important part of this project. Almost 75% of the effort went to these two parts.