1. K-Nearest Neighbor Classifier
   1. Lazy Classifier
2. When a new training example becomes available, among SVM, Naive Bayes and KNN, which classifier(s) have to be re-trained from scratch?

SVM has to be re-trained from scratch. For KNN, just add the new data to the training set and then it will be available for prediction. Nothing else has to be done. For Naïve Bayes also, just the count of data points will vary and accordingly probability values can be adjusted and hence it can be updated easily. But for SVM, the new data might change the support vectors entirely and hence has to re-trained from scratch.

1. When a new test example becomes available, among SVM, Naive Bayes and KNN, which

classifier needs the most computation to infer the class label for this example, and what is the time complexity for this inference, assuming that we have n training examples, and the number of features is significantly smaller than n?

1. Naïve Bayes

Consider number of features as d. Assuming binary classifier, (2d+1) independent probabilities are required. Each probability at the most can take all n examples for calculation. Hence it complexity is O((2d+1)n). Also, d is negligible compared to n and hence its O(n).

1. KNN

In KNN, firstly we need to calculate distance from the new test sample to all of the n training samples. Since number of features are negligible, it will take O(1) time to calculate distance from the test sample to 1 training sample. Therefore O(n) time is required to calculate distance from test sample to n training samples.

Now to select k closest points from the sample, need O(n Log k), assuming max-heap is used to select k closest points.

Selecting label based on majority vote will take O(k).

Hence complexity will be O(n)+O(n Log k)+O(k). If k is negligible compared to n, then O(n).

1. SVM

Since w and b will already be estimated, when the new sample just calculate wX+b and based on that take decision. Hence just O(1) for SVM.

* 1. Implementation of KNN Classifier

1. Pseudocode
2. Download 'mnist\_train.csv' and 'mnist\_test.csv' files from the site mentioned.
3. Load the first 6000 samples from training set to X\_train (Samples) and y\_train(Labels).
4. Load the last 1000 samples from test set to X\_test (Samples) and y\_test(Labels).
5. Calculate Euclidean Distance from each of the test sample to all the training samples and store it in a matrix of 1000\*6000 dimension.
6. Predict label for the test set using the distance matrix using KNN Classifier algorithm with different values of k and calculate error.
7. Plot the graph of error vs the value of k.

Function calculate\_distance\_matrix

For i=0 to NumberOfTestSamples-1

difference = X\_testi – X\_train

squared = difference^2

summed = ∑j (squaredj)

squareRooted = √ summed

distance\_matrix[i] = squareRooted

return distance\_matrix

Function predict

For i=0 to NumberOfTestSamples-1

distance\_from\_i = distance\_matrix[i]

sort(distance\_from\_i)

select k closest points from distance\_from\_i

obtain classes of those k points from y\_train

y\_pred[i] = majority label among k values

accuracy = (# y\_pred == y\_test) / (# y\_pred)

error = 1-accuracy

return error

1. Curve of Error vs Value of K

