Assignment-I

Sri Hari Malla - CS19BTECH11039

September 24, 2021

1 k - Nearest Neighbours

Given n points separated into two classes of data each of n/2 points, which are overlapped to some extent in a 2-dimensional space.

1.1 Variation in Training error while k changes from n to 1:

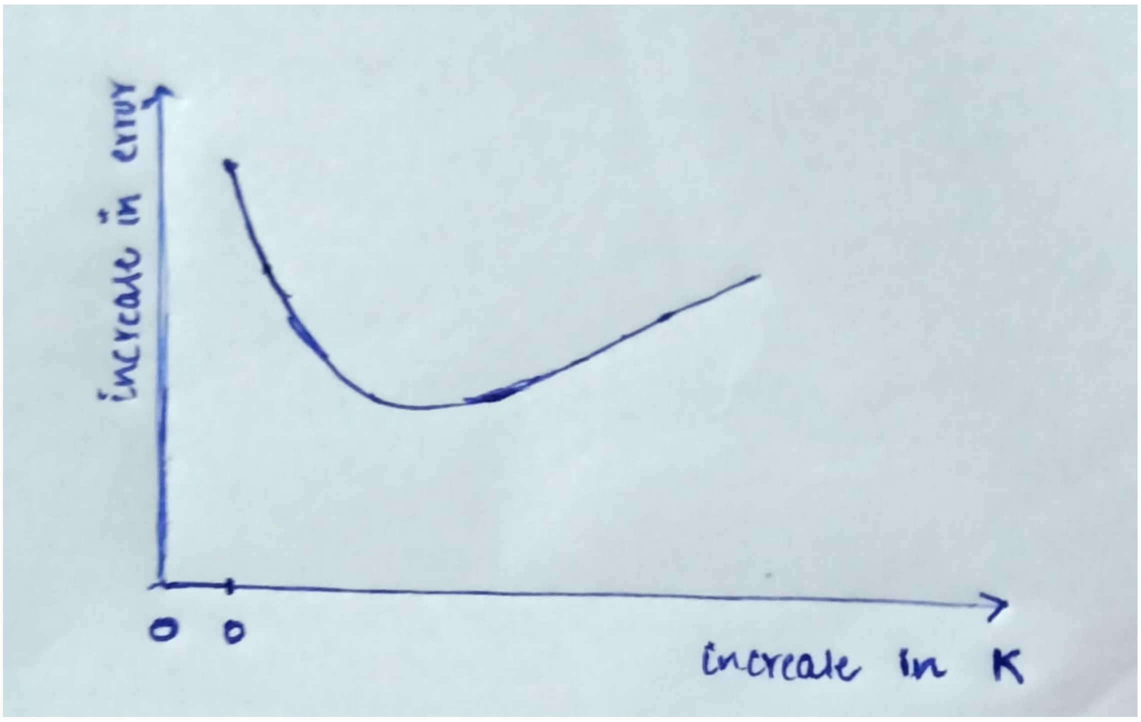
The training error is defined as the average error that results from using a method to predict the responses on the training set, the very same set which is used to train the method.

- k = n: As the model trains on all the points in the data set, we get the mean error and now we calculate the training error on all the data sets again. Which is the sum of squares of differences of each point with the mean point, which is variance.
- k = 1: As the model trains on only one neighbour, and is tested on the same neighbour again, there will be no mistake made. So the training error will be 0.
- $1 < k < n$: We can't really have an idea on this case unless the points are given. It depends on the location of the point. One can say that the training error decreases while k runs from n to 1.

1.2 Generalisation Error:

- When k =1, the generalisation (or test) error is very high because only one closer example is taken and this means that the rest of data is essentially discarded.
- As k increases, up to a point, more data in the training set is utilised to make prediction and hence the error reduces. This is where bias reduces essentially.
- But after a point, more than necessary training points are considered, so the model fits more on training set and poorly generalises. So the generalising error again raises. This amounts to increase in variance.

- Finally when k reaches to n, the error will not be as high as in the case of k = 1. The reason being that model will train with some k > 1, model gets trained with many points and definitely the model will have less error prone to the initial case.



1.3 k-NN and High dimensions:

- Consider a data set in multi dimensions of very high order. Now the euclidean distance is almost helpless in this kind of situations because, all the other vectors are equidistant from the point containing vector. It mainly hinges on the point that data points being closer together which becomes a limitation when we deal with high dimensions. As dimensions increases, the closest distance between two points approaches the average distance between points, which makes the k-NN algorithm to work in an inefficient way and no valuable predictions are made.

- Storage concerns and computationally expensive, One needs to keep all the memory to get the model running. Basically it doesn't learn much, but stores all the training instances and does comparisons at the training or testing time. So while the dimensions increase, the data points increases exponentially, and as a result the method becomes slower and slower.

1.4 Univariate Decision tree?

We can't build a uni variate decision tree based on the given constraint.

The decision Boundaries for 1-NN corresponds to each point's cell boundary. They need not be parallel to the co-ordinate axes as defined in the question and can be in any orientation. The decision tree boundaries were always parallel to the co-ordinate axes, so does the uni variate decision tree. In order to approximate, it can take any number of decisions, so we can't use the decision tree here.

2 Bayes Classifier:

2.1

According to the Bayes theorem on classification probability, we have the formula:

$$p\left(c_j \mid x\right)=\frac{p\left(x \mid c_j\right) p\left(c_j\right)}{p(x)}$$

The above can be written as:

$$=\frac{p\left(x \mid c_j\right) p\left(c_j\right)}{\sum_{k=1}^{K} p\left(x \mid c_k\right) p\left(c_k\right)}$$

In this case, we have the Gaussian likelihood distribution formula:

$$p\left(x \mid c_j\right)=\frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{1}{2}\left(\frac{x-\mu_j}{\sigma_j}\right)^2}$$

Given values, in the question:

$$\hat{\sigma_1^2}=0.0149, \Longrightarrow \hat{\sigma_1}=0.122$$

$$\hat{\sigma_2^2}=0.0092, \Longrightarrow \hat{\sigma_2}=0.095$$

We have ML parameter estimators as:

$$\hat{\mu}_j=\frac{1}{N_j} \sum_{i=1}^{N_j} x_i$$

$$\hat{p}\left(c_j\right)=\frac{N_j}{\sum_{k=1}^{N} N_k}$$

On computing from the above formulas, we get;

$$\hat{\mu}_1=\frac{\sum_{i=1}^{10} n_{1i}}{10}=\frac{2.6}{10}=0.26,$$

$$\hat{\mu}_2=\frac{\sum_{i=1}^{4} n_{2i}}{4}=\frac{3.54}{4}=0.8625$$

$$\hat{p_1} = \frac{10}{14} = 0.714,$$
$$\hat{p_2} = \frac{4}{14} = 0.286,$$

$$\text{Distribution of class1 } = 3.27e^{\frac{-1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2}$$

$$\text{Distribution of class1 } = 4.16e^{\frac{-1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2}$$

$$p\left(c_1 \mid x=0.6\right) = \frac{p(x=0.6|c_1)p(c_1)}{p(x=0.6|c_1)p(c_1)+p(x=0.6|c_2)p(c_2)}$$

$$p\left(x=0.6 \mid c_1\right)p\left(c_1\right) = 3.27e^{\frac{-1}{2}\left(\frac{0.6-0.26}{0.122}\right)^2} \times 0.714 = 0.048$$

$$p\left(x=0.6 \mid c_2\right)p\left(c_2\right) = 4.16e^{\frac{-1}{2}\left(\frac{0.6-0.8625}{0.0959}\right)^2} \times 0.286 = 0.028$$

$$\implies p\left(c_1 \mid x=0.6\right) = 0.6309$$

2.2 Maximum likelihood Naive Bayes classifier:

We know that,

$$p\left(c_p \mid \vec{x}\right) = \frac{p(\vec{x}|c_p)p(c_p)}{p(\vec{x}|c_p)p(c_p)+p(\vec{x}|c_s)p(c_s)}$$

and also,

$$p\left(\vec{x} \mid c_s\right) = p\left(x_1 \mid c_s\right)\ldots p\left(x_8 \mid c_s\right)$$

$$\text{but } p\left(x_7 = 1 \mid c_s\right) = 0$$

Which means,

$$\implies p\left(\vec{x} \mid c_s\right) = 0$$

Now going to the above mentioned formula from present observation,

$$p\left(c_p \mid \vec{x}\right) = \frac{p(\vec{x}|c_p)p(c_p)}{p(\vec{x}|c_p)p(c_p)+0}$$

$$p\left(c_p \mid \vec{x}\right) = \frac{p(\vec{x}|c_p)p(c_p)}{p(\vec{x}|c_p)p(c_p)} = 1$$

$$\implies p\left(c_p \mid \vec{x}\right) = 1$$

3 Decision Trees

3.1 Implementation:

The flow goes like this:

- Reading and storing the data
- Splitting the data into training and test sets.
- Creating a decision tree instance.
- Learning phase of the decision tree.
  - Assign the depth of the tree(for performance)
  - Building the decision tree
    - If the depth is above the max depth, return a leaf
    - Get the best split of the data
      - Assign initial information gain to negative maximum
      - Iterate over each index in the training set
      - Separate the left tree data and right tree data
      - Get the information gain from the data obtained.(Gini/Entropy)
      - Pick out the split with max information gain and return it.
    - create a decision node with the data obtained from best split.
    - Recursively call build tree on left and right data as splitted in best split.
- Evaluating the result on the test set using classify method.
  - If the node is leaf, return the value of the leaf.
  - Choose the tree(left/right) based on the data present in the decision node.
  - Recursively go through the choosen tree until we hit a leaf node.

The Entire code is in the file named **cs19btech11039.py** All the initial code and improved code is in the same file. Refer to note.txt for the improved code line numbers.

3.2 K-fold cross validation:

3.2.1 Splitting data Initial:

- Add into training set if index of row in set is not a multiple of 10
- Add into testing set if index of row in set is a multiple of 10
- Train and test on the data.

3.2.2 Splitting data Final:

- Loop over 0 to K(Given, K = 10) and label as rem(remainder)
- In each iteration, create a tree instance, and split the data
- Add into the training set if index of row in set doesn't leave the remainder as rem
- Add into the testing set if index of row in set leaves a remainder as rem
- Essentially, we get the required split.
- Train the model using decision tree algorithm using training set
- Test the model using test set.
- Record the accuracy and decide. Add it to the global accuracy for getting average of K accuracies.
- Average the global accuracy to get an estimate accuracy.

Below is the implementation:

```
accuracy = 0
K = 10
for rem in range(0,K):
    training_set = [x for i, x in enumerate(data) if i % K != rem]
    test_set = [x for i, x in enumerate(data) if i % K == rem]

    tree = DecisionTree()
    # Construct a tree using training set
    tree.learn(training_set)

    # Classify the test set using the tree we just constructed
    results = []
    for instance in test_set:
        result = tree.classify(instance[:-1])
        results.append(result == instance[-1])

    # Accuracy
    kaccuracy = float(results.count(True))/float(len(results))
    print(rem,"completed with accuracy ",kaccuracy)
    accuracy += kaccuracy

accuracy /= K
print("accuracy: %.4f" % accuracy)
```

The same can be found in the python files attached.

3.3 Improvements:

3.3.1 Initial Implementation Detials:

- Depth = No limit, Time taken is too much as it is over fitting.
- Average Accuracy: 0.79 (using Entropy)
- Average Accuracy: 0.82 (using Gini)
- Time taken for 1 fold: Nearly 1.5 mins
- Time taken on average for all sets: 13-14 mins

This is too much cost on a data with 5000 data points. So the improvements done are:

- Limiting the depth of the tree
- Using Gini Index

3.3.2 Depth of the tree(observed on entropy):

Initially, there is no limit on the depth of the tree.
So, as a result the model became overfitting with the data set.

So, added a condition in build tree which directs the program to build sub trees if and only if the present depth of the tree is less than the maximum depth assigned.

Now, the model does only certain number of computations and builds a minimal version of the previously built over fitting tree.

Observations:

- Depth: Assigned as number of features.
  - Average Accuracy: 0.79 (same as that of the over fitting tree) with minimal changes.
  - Time taken for 1 fold: 1 min
  - Average time taken: 8-9 Mins

  Better than the over fitting tree as there is no much difference in the final accuracy and time decreased by a considerable extent.

- Depth: Assigned as half the number of features.
  - Average Accuracy: 0.78(same as that of the over fitting tree) with very minimal changes on a scale of 0.000001.
  - Time taken for 1 fold: 42 sec
  - Average Time taken: 4-5 mins

  Better than the improved tree as there is no much difference in the final accuracy and time decreased by a considerable extent.

- Depth: Assigned as a quarter of number of features.
  - Average Accuracy: 0.78(same as that of the improved fitting tree) with very minimal changes on a scale of 0.000001.
  - Time taken for 1 fold: 25-30 sec
  - Average Time taken: 2 mins

  Better than the improved tree as there is no much difference in the final accuracy and time decreased by a considerable extent.

- Depth: Less than 1/4 of number of features
  - Time is improving.
  - But the problem is the fall in accuracy.

So finally, the over fitting model is improved by limiting the depth of the tree. Finally the best depth is recorded for the value of depth as **(numFeatures)/4**

3.3.3 Gini Index:

Using Gini Index and impurity concept increased the accuracy slightly on a scale of 0.1 which is highly preferred minimum reduce of change in cost of computation.

- Accuracy: 0.8 and above
- Time taken: Very slightly better than Entropy.

Reasons:

- **cost:**
  - The computation cost of Gini index is slightly better than that of entropy.
  - Because, the calculation of entropy involves complex logartihmic in built function but on the other hand, gini index just uses squaring of probability.
- **Feature Selection:**
  - The entropy of node can vary from 0 to 1
  - The Gini Index of node can vary from 0 to 0.5
  - The entropy has a wider range of values than gini index.
  - Given, that entropy has a wider range, the values are more discrete for Entropy than the gini index.
  - Therefore, it is better to have gini index as a criterion for information gain.

So both the reasons played part to make Gini index criterion better than the entropy.

These are the two improvements made for the initially built decision tree.

LaTeX generated document

THE END