Foundations of Machine Learning
Assignment 1

Sakshi Badole, CS24MTECH11008

5 September 2024

1 k-NN

1. (a) In case of k-NN , when there are n points and n/2 and n/2 points are present in each of the two classes. Training error varies when k = n to 1 as:

- When k = 0, the nearest neighbor is itself hence the training error at k = 0 will be zero.
- As the value of k increases from k = 1 to some initial values of k the training error also increases but is low. This low training error is a sign of over-fitting as the testing error is high in this region.
- As the value of k increases from the initial values, the training error increases but does not vary much as the model now taking more neighbors for voting resulting in a better generalization.
- At later values of k , the training error becomes constant, this happens because the already included nearest neighbors also weigh in a lot not actually causing a change in the current state of the model. Increasing value of k after this point will add no benefit to the model.

1. (b) In case of k-NN , when there are n points and n/2 and n/2 points are present in each of the two classes. Testing error varies when k = n to 1 as:

- The testing error at k =1 is high as we are only considering only 1 neighbor which may not properly predict the correct class label (the nearest neighbor can also be a noise in data).
- The testing error in case of k-NN decreases as the value of k increases as now more neighbors are being included in the voting resulting in a better generalization of the dataset and better predictions.
- But this decrease in the testing error continues only till some optimal value of k where we see a dip in the testing error. This value is where the model has the best generalization.
- After this value of k, the testing error begins increasing again with an increase in k's value. The testing error becomes constant after some value of k as the model is now under-fitting, it does not learn any new pattern now as the local structure of the data remains same (due to already existing nearest neighbor that were also present at the smaller values of k) even though it is considering more and more neighbors.

Figure 1: Sketch displaying the graph of how the testing and training error varies for k-NN as the value of k varies from 0 to n

1. (c) For the determination of the possibility of building a univariate decision tree which exactly works like a 1-NN (k-nearest neighbor with k=1) by the Euclidean distance measure, firstly we should define and compare the working and limitations of these methods:

Comparison of the 1-NN Classification applying Euclidean Distance and Univariate Decision Trees:
The major difference lies between the decision boundaries with the 1-NN being able to adapt to complex shapes due to having a direct dependency on the data, whereas the Univariate Decision tree focuses on a single feature at each node, that provides an axis aligned piece-wise approximation of feature space.
Conclusion:

- **It is possible for a Univariate Decision Tree to work exactly like a 1-NN classifier ONLY if certain conditions are fulfilled.** The dataset classes are not overlapping and it is possible to create a linear decision boundary. Only in this case the decision boundary generated by Univariate Decision Tree is somewhat similar to that of 1-NN classifier for that particular dataset.

- Other than that, it is not possible for a Univariate Decision Tree to work exactly like a 1-NN classifier using Euclidean distance measure because of the following:
  Complex Boundaries: A univariate decision tree can only create parallel axes decision boundaries while the 1-NN can work on complex decision boundaries since it relies on the position of all data points in the feature space.
  Dimensional Limit: The 1-NN decision tree works multidimensionally providing vast results and capturing the interactions between various features that mimic the approach of a Univariate decision tree that provides results solely on one feature at a time.
  Therefore, the limitation of Univariate decision tree of approximating certain types of decision boundaries results in lacking flexibility due to which it cannot act like 1-NN classifier except for one case.

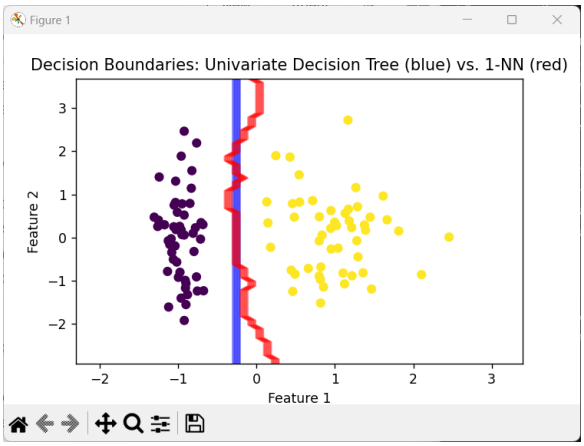

Figure 2: Graph showing Univariate Decision Tree and 1-NN decision boundaries for a dataset having linear decision boundary where the decision tree acts classifies similar to 1-NN.

2 Bayes Classifier

2. (a) class 1: {0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25} and class 2: {0.9, 0.8, 0.75, 1.0}. Class 1 variance : 0.0149, and class 2 variance 0.0092. Also estimate the class probabilities p1 and p2 using Maximum Likelihood. What is the probability that the test point x = 0.6 belongs to class 1?

Solution:

Using Gaussian Likelihood:

$$p(x \mid c_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)$$

Here,

Mean, $\widehat{\mu_i} = \frac{1}{n_i}\sum_{j=1}^{n_i} x_j$

Variance, $\widehat{\sigma_i^2} = \frac{1}{n_i}\sum_{j=1}^{n_i}(x_j - \widehat{\mu_i})^2$

- Fitting a Gaussian using Maximum likelihood to class 1:
  Variance for class 1 (given) = 0.0149
  Mean for class 1 : $\frac{(0.5+0.1+0.2+0.4+0.3+0.2+0.2+0.1+0.35+0.25)}{10}$
  Mean (μ_1) = 0.26
- Fitting a Gaussian using Maximum likelihood to class 2:
  Variance for class 2 (given) = 0.0092
  Mean for class 1 : $\frac{(0.9+0.8+0.75+1.0)}{4}$
  Mean (μ_2) = 0.8625

- Class probability using Gaussian likelihood : $\widehat{p}(c_i) = \dfrac{n_i}{\sum_{j=1}^{n}(n_j)}$

  Probability of class c_1 : $\widehat{p}(c_1) = \dfrac{10}{(10+4)} = 0.714$

  Probability of classc_2 : $\widehat{p}(c_2) = \dfrac{4}{(10+4)} = 0.2857$

- Bayes Theorem for classification :

$$p(c_i|x) = \frac{p(x|c_i)p(c_i)}{\sum_{j=1}^{n} p(x|c_j)p(c_j)}$$

Probality $p(c_1|0.6) = \dfrac{p(0.6|c_1)p(c_1)}{p(0.6|c_1)p(c_1) + p(0.6|c_2)p(c_2)} = \dfrac{0.068 \times 0.714}{(0.068 \times 0.714 + 0.098 \times 0.2857)}$

= **0.63499**

2. (b) Probability that the document x = (1, 0, 0, 1, 1, 1, 1, 0) is about politics?
To solve this problem using maximum likelihood Naïve Bayes Classifier,

p(politics) = $\dfrac{6}{12} = 0.5$

p(sports) = $\dfrac{6}{12} = 0.5$

$p(sports|x) \propto p(sports)p(goal|sports)p(football|sports)p(golf|sports)$
$p(defence|sports)p(offence|sports)p(wicket|sports)p(office|sports)p(strategy|sports)$

$p(sports|x) \propto 0.5 \times \dfrac{4}{6} \times \dfrac{2}{6} \times \dfrac{5}{6} \times \dfrac{4}{6} \times \dfrac{1}{6} \times \dfrac{1}{6} \times \dfrac{0}{6} \times \dfrac{5}{6}$

$p(sports|x) = 0$

$p(politics|x) \propto p(politics)p(goal|politics)p(football|politics)p(golf|politics)$
$p(defence|politics)p(offence|politics)p(wicket|politics)p(office|politics)p(strategy|politics)$

$p(politics|x) \propto 0.5 \times \dfrac{2}{6} \times \dfrac{5}{6} \times \dfrac{5}{6} \times \dfrac{5}{6} \times \dfrac{5}{6} \times \dfrac{1}{6} \times \dfrac{4}{6} \times \dfrac{1}{6}$

The probability of document x belonging to class politics is:-

$p(politics|x) = 0.5 \times 0.0029768 =$ **0.001488**

3 Decision Tree

3. (a)
The accuracy for the Decision tree implemented using binary univariate split, entropy ans information gain comes around 0.8237. In this implementation I have used information gain which is calculated using entropy $-\sum plog(p)$ where p is the probility of a class label in that node.

3. (b)
The Accuracy of Decision Tree after applying K-Fold Cross Validation with k=10 is 0.8172. Cross Validation helps us validate that our model is really working as it should, we can tweak the hyperparameters after running k-fold validation to get a better model.

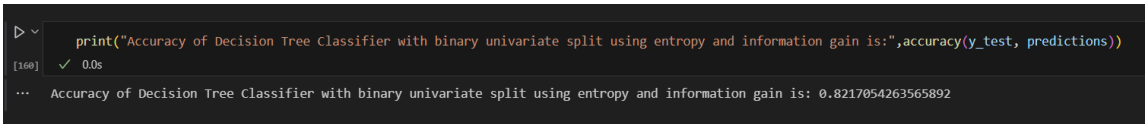

Figure 3: Snapshot of Accuracy in this case.

Figure 4: Snapshot of accuracies in different iterations and the average accuracy in case of k-fold

3. (c)

For improvement strategies I have used :

- Gini index: Here the accuracy increased from 0.8237 to 0.84911. This has made the model faster as the calculation of Gini Index is less complicated than that of Entropy.
  Gini Index $= 1 - \sum p^2$, here p is the probability of a particular class label being in that node.

Figure 5: Snapshot displaying the accuracy of Decision Tree when Gini Index is used.

- Pruning the decision tree after splitting: Here the accuracy is increased from 0.8260 to 0.82909. Pruning the tree helps in better generalization as it doesn't let it go to really large depths which may lead to overfitting as the tree will completely fit the training data but its efficiency will reduce in case of testing data. Pruning the tree reduced the unnecessary splits to maintain the generalization of the decision tree.

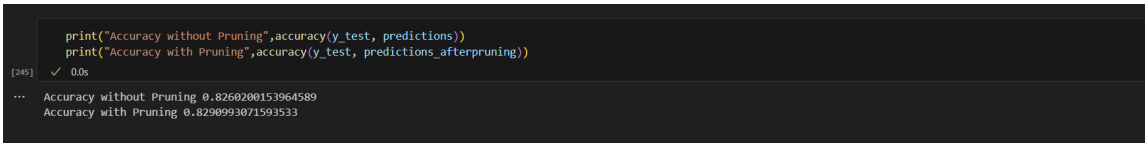

Figure 6: Snapshot displaying the accuracy of Decision Tree and the Pruned Decision Tree

- Accuracy by using both pruning and gini index increases from 0.8237 to 0.8337.

Figure 7: Accuracy of decision tree using both Pruning and Gini index

4 Method Comparison

4. (a) Computational Complexity of KNN, decision trees and Naïve Bayes.
The computational complexity of **KNN** during:

- Training Phase Time Complexity: **O(1)**
  The model need not be trained hence it is a constant time process. But the space complexity is O(nd) to store the data.
- Inference Phase: **O(knd)**, here k = number of neighbors selected, n = data samples, d = dimensions of data (or number of features)
  As an inference is made, the distance from all existing k values need to be calculated, with an increase in the value of k and the dimensions of data points the calculation complexity increases leading to an increase in the computational complexity.

The computational complexity of **decision trees**:

- Training Phase: **O(nlog(n)d)**, here n = data samples, d = dimensions of data (or number of features)
  A decision tree is created for which we have to evaluate best feature out of d features over n data samples, hence n*d. The depth of the decision tree possible for a properly balanced decision tree i.e. log(n).When the decision tree is not properly balanced then the max depth may go till O(n) in the worst case. Hence, the total time complexity for a properly implemented decision tree is O(dnlog(n)).
- Inference Phase:**O(log(n))**, here n = data samples
  For testing a data point, we need to traverse the decision tree traversed which takes O(log(n)) time to traverse till its depth when implemented properly. The max depth of the tree is possible to be O(n) in an unbalanced tree in the worst case.

The computational complexity of **Naïve Bayes**:

- Training Phase: **O(nd)** While training we have to calculate the prior i.e. class probabilities of each class and likelihood p(x | c) for the all the n data points of the training data where any data point , say 'x', contains d features or dimensions.

  $$p(x|c_i) \times (c_i) = p(d_1|c_i) \times p(d_2|c_i) \times p(d_3|c_i) \times \ldots \times p(d_d|c_i) \times p(c_i) \quad (1)$$

  Total of O(ncd) time required but here I neglected c(number of classes) as it is much smaller value.
- Inference: **O(d)**, here d is the number of features of the dataset
  For inference, we just need to get the value of likelihoods of different features and multiply them for each class to get the maximum likelihood which in turn help us classify the test sample. Total of O(dc) operations needed , I neglected 'c' here as it is much smaller in comparison.

4. (b)
Answer

1. k-NN
    - Large Dataset – Large Number of Features:
      The time complexity of k-NN for testing is O(nkd), clearly as the dataset size ('n') and the number of features ('d') increases the computational complexity increases and the performance reduces. As the dimensions increases the nearest neighbors' distance from the data sample increases and the model becomes less accurate, also called as the curse of dimensionality. This case is the **weakness of k-NN**.
    - Large Dataset – Small Number of Features:
      The large dataset increases the computational complexity as 'n' value increases while the small number of features reduces the 'd' value thus improving efficiency as compared to the large number of features.
    - Small Dataset-Large Number of Features:
      Small dataset is better for k-NN as the computational complexity is reduced as the 'n' size is reduced. The large number of features still causes the curse of dimensionality which in turn reduces the efficiency of the model.
    - Small Dataset - Small Number of Features:
      This case is the **strength of kNN** as the computational complexity is less when dataset size 'n' value is small as well as the number of features/dimensions of the data 'd' is small.
2. Naïve Bayes:
    - Large Dataset – Large Number of Features:
      Large dataset is good for Naïve Bayes as there may be sufficient observations for each class, while large number of features may cause a problem if they don't follow the 'Naïve' assumption that each feature is independent of the other one.
    - Large Dataset – Small Number of Features:
      Large dataset with small number of features is **great(strength) case for Naïve Bayes** as large datasets may have sufficient observations, while small number of features will have little affect over the classification efficiency even if the independence assumption is not true. Evaluating a result for a test case will also be faster as the time complexity O(d) is really small in this case.
    - Small Dataset-Large Number of Features:
      This case can be called as the **weakness of Naïve Bayes**. Small dataset will/may have limited observations of for each class which may not be sufficient to properly reach the correct classification label. Large number of features not following the independence assumption will hinder the accuracy of the classifier.
    - Small Dataset- Small Number of Features:
      Small number of features won't affect the accuracy of the classifier that much even if the independence assumption for the features does not hold. While small dataset may not provide enough observation about all the class label reducing the efficiency of the classifier.
3. Decision Tree
    - Large Dataset – Large Number of Features:
      Large datasets can be handled by decision trees but it does increase the computational complexity as the time complexity for this is O(dnlog(n)), and the large number of features may cause the tree to split again and again creating a complex decision tree. It is important to scale the features properly for better efficiency of the tree.

- Large Dataset – Small Number of Features:
  Large datasets increase the computational complexity but the small number of features is still a better case for the decision tree, as that of large number of features. Smaller number of features reduces the computational complexity as the time complexity is O (n*d*log(n)).
- Small Dataset - Large Number of Features:
  Large number of features will cause more branching which leads to complex tree and overfitting. This can be solved by pruning the tree.
- Small Dataset - Small Number of Features:
  Limited dataset may not help properly to generalize the solution, which may lead to overfitting as the model will adjust to the training data and fail to generalize.