CS6510: Applied Machine Learning

Assignment 1

Saksham Mittal

03.02.2019

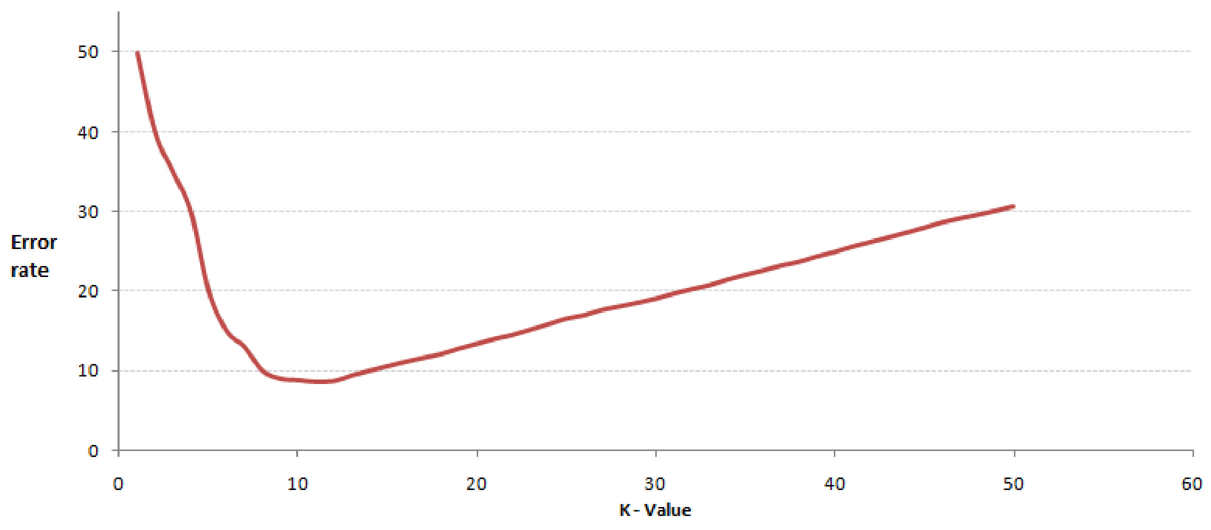
CS16BTECH11032

1. *k*-NN:

a) Training error is computed by running the trained model on train data itself. Hence, for k = 1 naturally, the error will be 0 as every point chooses itself as the nearest neighbor.

However, as k increases, we encounter other points also belonging to the other class, which increases the training error. It reaches its max at k = N, i.e. when all points are taken into consideration. Since N/2 points belong to each class, there is 50% error in guessing the correct class.

b) Generalization error decreases on increasing k to a limit(when k becomes optimum, usually k = sqrt(n)), and then increases on further increasing k. (and so does the computational cost)



Generalization error vs k

c) The two reasons are:
   - Curse of Dimensionality: As the dimension increases, the indegree distribution of the k-NN digraph becomes skewed with a peak on the right because of the emergence of a disproportionate number of hubs, that is, data-points that appear in many more k-NN lists of other data-points than the average. (Source: Wikipedia)
   - High computational power required to calculate distances in the higher dimensions.

d) We cant make a univariate decision tree as the decision tree boundaries would be parallel to coordinate axes which in turn would depend on comparisons made at each node of the decision tree. While for 1-NN the cell boundaries may not be parallel to the coordinate axes.

2. Bayes Classifier:

2) (a) First, fit a gaussian using maximum likelihood:

MLE for mean of each ~~generation~~ = gaussian = $\sum_i \frac{x_i}{n}$

Given, Data for class 1

$= \{0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25\}$

Data for class 2 = $\{0.9, 0.8, 0.75, ~~0.1~~ 1.0\}$

$$\hat{\mu}_1 = \frac{\Sigma x_i}{10} = \frac{0.5+0.1+0.2+0.4+0.3+0.2+0.2+0.1+0.35+0.25}{10}$$

$$= \frac{2.6}{10} = 0.26$$

and $\hat{\mu}_2 = \frac{0.9+0.8+0.75+1.0}{4} = 0.8625$

Class probabilities =,

$$P_1 = \frac{10}{14} = 0.7143$$

$$P_2 = 1 - P_1 = 0.2857$$

Given variances, $\sigma_1^2 = 0.0149$, $\sigma_2^2 = 0.0092$

To find probability that $x = 0.6$ belongs to class 1, we use Bayes theorem: $P(C_1|x)$

$$= \frac{P(x|C_1)\,P(C_1)}{P(x)} = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$\left\{ P(x|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left[-\frac{1}{2}\frac{(x-\mu_k)^2}{\sigma_k^2}\right] \right\}$$

(using Gaussian likelihood)

$$\therefore \quad P(x|C_1) = \frac{1}{\sqrt{2\pi \times 0.0149}} e^{-\left\{\frac{1}{2}\frac{(0.6-0.26)^2}{0.0149}\right\}}$$

$$= 0.06756$$

$$\& \ P(x|C_2) = 0.09834$$

$$\therefore \quad \boxed{P(C_1|x) = 0.6305}$$

(b) To compute : $Pr[\, x=(1,0,0,1,1,1,1,0) \text{ is politics}]$

In x_{sport} table 7th attribute all values are 0

Using naive bayes on data for x, we get.

$$P(x_{pol}|x) = \frac{P(x|\text{politics})\, P(\text{politics})}{P(x|\text{politics})\, P(\text{politics}) + P(x|\text{sports})\, P(\text{sports})}$$

If we take $P(x|\text{sports}) = 0$, we get ~~P(x)~~

$$\boxed{P(\text{politics}|x) = 1}$$

This is known as <u>zero frequency problem</u>.

To solve this we add 1 to the count of every attribute value class pair.

$$\Rightarrow P(\text{politics}|x) = \frac{\frac{3}{8}\times\frac{6}{8}\times\frac{6}{8}\times\frac{6}{8}\times\frac{6}{8}\times\frac{2}{8}\times\frac{5}{8}\times\frac{2}{8} \times \frac{1}{2}}{\left(\frac{3}{8}\times\frac{6}{8}\times\frac{6}{8}\times\frac{6}{8}\times\frac{6}{8}\times\frac{2}{8}\times\frac{5}{8}\times\frac{2}{8}\right)\frac{1}{2} + \left(\frac{5}{8}\times\frac{3}{8}\times\frac{6}{8}\times\frac{5}{8}\times\frac{2}{8}\times\frac{2}{8}\times\frac{1}{8}\times\frac{6}{8}\right)\frac{1}{2}}$$

$$= \underline{0.878}$$

This gives a more realistic probability for x to belong to politics.

3. Decision Trees:

Design and implementation:

The decision tree implemented is using binary univariate split and entropy is used for calculating the homogeneity of the sample. An attribute with highest information gain is tested/split first.

We iterate over all the attributes and find the attribute which has highest information gain. For finding the split value, i.e. on which value to split the attribute, I tried various methods.

First was taking the mean of the attribute values as the split value. But since this doesn't capture the best way to split, the accuracy suffers. Then, I finally divided the attribute into 15 equal parts and took the value representing each part as the split value.

This increased the computational power used(and hence, the time taken to train) but improved the accuracy of the model.

After finding the split value, the data is split into two parts(binary split) according to the split value, and that attribute is removed. This process repeats until we encounter a leaf node. For leaf node either the entropy is 0, or no attributes are left to split.

For part (b), I used 10-fold cross-validation. First, I divided the data into 10 parts. Then, used the first 9 parts for training, and last one for testing. Similarly, each part is used for testing and other nines for training. The final accuracy is average of the 10 fold accuracies.

Accuracy:

(i) Accuracy of initial implementation: **81.58**

(ii) Accuracy of improved implementation: **82.36**

For the improved implementation, I used Gini index instead of entropy for purity measurement. Gini index computes purity as follows:

$$\text{Gini Index} = 1 - \sum_{j} P_j^2$$

Both are methods used in classification problems, but Gini index is preferred, as it is easy to compute compared to entropy which requires to computes logarithms(hence computationally expensive).

Also, Gini is intended for continuous attributes and Entropy for attributes that occur in classes. Hence, we get improved accuracy.

I also used the pre-pruning technique for better generalizations and improving the accuracy of the model.

Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

With these changes, I was able to improve the accuracy of the model by ~1%.

3. Kaggle - What's cooking:

I used scikitlearn library's implementation of kNN, Decision tree, and Naive Bayes algorithms for training the model.

1) k-Nearest Neighbours: This model scored the least score on Kaggle. The highest score was obtained on k = 40 (Number of distinct classes was 20). On further increasing k, the score was decreased. I tried k from 40 to 100 with an increment of 10. The reason for the decrease in score for larger k can be due to overfitting. Also, the poor performance of kNN can be attributed to the **Curse of Dimensionality**.

   As the dimension increases, the indegree distribution of the k-NN digraph becomes skewed with a peak on the right because of the emergence of a disproportionate number of hubs, that is, data-points that appear in many more k-NN lists of other data-points than the average. This phenomenon can have a considerable impact on various techniques for classification. *(Source: Wikipedia)*

   Score obtained(k = 40): **0.56908**

2) Decision Tree: This model scored the 2nd highest score on Kaggle. I used the

default options of the classifier, i.e. gini index for purity and best features for splitting instead of all features.

Decision tree works better as the splits at each level can be done for the elements belonging to a particular class.

Score obtained: **0.61967**

3) Bernoulli Naive Bayes: This model scored the highest score on Kaggle. Since the attributes are independent and binary. This makes the data fit as input for the multivariate Bernoulli model, as the features there are independent boolean variables.

   Score obtained: **0.71148**