COMP24112 Lab 2: News Article Classification by k–NN

MARCH 2023

Experiment 1:

The cosine distance method performs better because the mean of the testing accuracies with Euclidean distance is 86.4%, while the cosine distance method is 96.8%.
According to my investigation, This can be attributed to the nature of the data and the way the two measures calculate distance.

The Euclidean distance is a way to calculate the straight-line separation between two points in a multidimensional space. The distance is calculated using the absolute differences between the values of the features, which may give greater-value features a higher weight. When the datasets are continuous and have similar ranges, this distance metric performs well.

Rather than comparing the magnitude of two vectors, the cosine similarity metric determines how similar they are by comparing their angles. This indicates that regardless of the scale of the feature values, the cosine similarity measure can represent the similarity between two vectors. Also, the cosine similarity measure is particularly helpful when the data is made up of sparse vectors or binary vectors since it can deal with conditions where many features have zero or nearly zero values.

Our experiment data is discrete and it is a very sparse data matrix, so it is more suitable for using the cosine distance method.

Experiment 2:

The training accuracies are higher than the test accuracies in most cases as the KNN classifier function uses the same data for both training and testing.

In my experiment, the gap between the two accuracies is generally small, which indicates that the model has achieved a good balance between fitting the training data and generalizing to new data.

A small value of k might lead to low bias and high variance, meaning the model may match the training data well but may be quite sensitive to data noise. A large value of k, on the other hand, can lead to high bias and low variance, which means that the model may not adequately capture the complexity of the data but may be more resistant to noise.

Initially, when k is small, the difference between the average testing error rate and the average training error rate is obvious, but the bias is small. When k becomes larger, the error rate rises, which means the bias increases. Also, the variance gradually becomes smaller, but not significantly.

3 is a reasonable value for k considering the bias and the variance.

Experiment 3:

(1)
My classifier does not make an appropriate class prediction for these 5 articles. These 5 articles are irrelevant to those initial 4 classes, and they might belong to the “sport” class based on my judgement. This is because if the new data represents a new kind of object that was not present in the training data or the new data has different features than the training data, then the classifier may not have learned to recognize it.

(3)
My classifier made a high accuracy generally, which is approximately 95% or above.
Having no training data means the model can not learn the pattern and relationships of features for the new class, and as a result, the model will not be able to classify any data points belonging to that class accurately. If there is a lack of training data for a new class, the model will have difficulty understanding its distinguishing features because of insufficient samples, leading to poor generalization performance.

(4)
Here are basic concepts about zero-shot learning and few-shot:

Zero-shot learning refers to the task of learning to recognize objects or classes that have never been seen before, without any training examples for those classes. In zero-shot learning, the goal is to use prior knowledge about the known classes to generalize to the unseen classes.
Few-shot learning, on the other hand, refers to the task of learning to recognize new objects or classes from only a few examples. In few-shot learning, the goal is to learn a model that can generalize to new classes with only a few examples.

My experiment in part(1) could be regarded as Zero-shot and it performs Few-shot learning for part(2)(3). In this experiment, the model is trained on only 5 new articles as examples of a new class, and it is expected to generalize to new examples of that class during testing.

Analysis 1:

There are several steps for computing the interval where its true error lies with 90% probability. Firstly we need to Obtain a random sample of size n from the population, here we choose 480. The second step is to Calculate the Error rate for k=1. Then we need to compute the alpha value, which is (Confidence level * Standard Error), the formula is

$$a = z_p \sqrt{\frac{\mathrm{error}_s\left(1 - \text{ error }_s\right)}{n}}$$

The Confidence level of 90%(critical value) is 1.64, and Standard Error = $\sqrt{(p(1-p)/n)}$, where p is the testing error rate, and n is the number of test samples. Finally, we could get the Confidence Interval:

$$\mathrm{error}_D \in \left[\text{ error }_S - a, \text{ error }_S + a\right]$$

Analysis 2:

45-NN has higher testing sample error. The probability that it also has a higher true error is **90.5%.**
Here it implements Z-test. Firstly we need to Compute a quantity zp:

$$zp = \frac{d}{\sigma},$$

where d is the absolute value of the difference between two test errors, here is 45-NN test error and 1-NN test error.

$$\left|\mathrm{error}_{s1}(A) - \mathrm{error}_{s2}(B)\right|$$

The next step is σ is the square root of the sum of two Standard Errors, here is the formula:

$$\sigma = \sqrt{\frac{errorA\left[1 - errorA\right]}{n_1} + \frac{errorB\left[1 - errorB\right]}{n_2}}$$

Once we get the zp, we can run the “Get_p_value” function to get the confidence value p. In the end we compute the final probability C(The probability that it also has a higher true error):

$$C = 1 - \frac{(1-p)}{2}$$

Hyperparameter Selection:

we will split the whole dataset into three parts: a training set, a validation set, and a test set. The training set will be used to train the k-NN model, the validation set will be used to select the best value of k, and the test set will be used to evaluate the performance of the final model.

I utilize K-fold cross-validation with k=10 for hyperparameter selection. In order to do this, the first thing is to shuffle the dataset randomly and split the whole data set into training data(80% of total data) and test data(20% of total data). Then the training data should be divided into k equal-sized subsets. The model needs to be trained and tested k times, with each validation set being a new subset and the remaining k-1 subsets serving as the training set. The value of k that gives the best performance can then be selected as the optimal value. As compared to a single train/test split, applying a K-fold CV for hyperparameter selection offers a more accurate estimation of the model's performance. It also makes better use of the data because every observation is used for both training and validation.

The key reason for splitting data is to evaluate the model's performance. The training set is used to train the model, whereas the test set is used to assess the model's performance on new, previously unknown data. The validation data can select the best set of hyper parameters for the model. We could evaluate how well the model will adapt to new data by comparing its predictions on the test set to the labeled data.

Another basic reason for separating the data is to avoid overfitting. Overfitting happens when the model is overly complex and closely matches the training data, including noise and random fluctuations. We may determine whether the model is overfitting by evaluating its performance on a different validation set and adjusting the model's parameters accordingly.

I implement “10-fold” here and the chosen value of k is 1, the accuracy is 97.47%