1 Euclidean vs Cosine Distance

In my experiments I found that the euclidean distance had a mean of: 87.531%, and a standard deviation of: 2.399%. My cosine distance had a much better performance with a mean of: 96.094%, and a standard deviation of: 0.972%. I believe this is because cosine distance does not consider the magnitude of a vector when calculating similarity. I have demonstrated this in figure 1, where A,B,C are articles. We can see that $A \rightarrow B$ is more similar than $A \rightarrow C$ when considering euclidean distance (d1,d2), but have the same similarity when using cosine distance (θ). This is useful for our article classification model, especially when we have varying lengths of articles. Let's say the word "telephone" appears higher in article B than C, we might assume that B is more related than C to a given class. However, if B is longer in length than C, there is a high probability of an identifying word occurring more in B than C due to length, and not because of similarity to a class. Euclidean distance cannot identify this error, but cosine distance corrects for this.

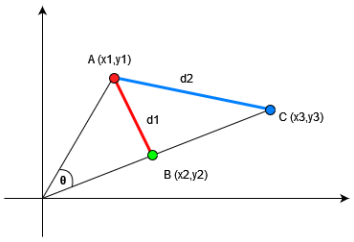
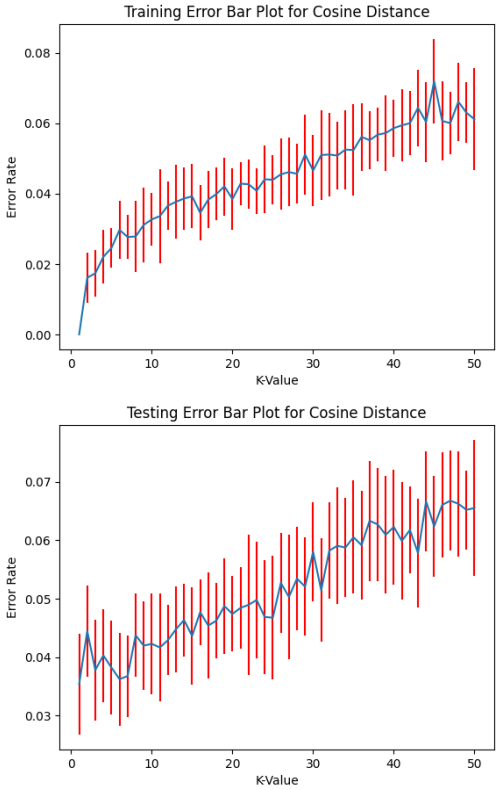


Figure 1: Euclidean vs Cosine

2 Error Rate of Cosine Distance



Comparing the averaged accuracies (blue line) of the training and testing data we can see that the training data starts off very low around 1-2% at $k = 1$ before logarithmically increasing to around 6-7% for $k = 50$. On the other hand, the testing data starts with an error rate of around 4% at $k = 1$ and increases linearly towards around 6.5% at $k = 50$. I believe that the training data behaves in this fashion as at a k-value of 1 it will most likely match with itself in similarity, but as the k value increases the specific article must match with articles of a different which exponentially increases the chances of classifying as the wrong article. The testing data behaves in a more linear fashion than the training data, as it cannot possibly match with itself as it is not present in the training data, so it has a higher error rate when $k = 1$. As the k-value increases it must match with more articles, which increases the chance of classifying as the wrong article, especially if the article is an edge case (an article that borders on 2 classes in similarity).

As k increases, we see an increase in error rate, which is natural. There is a higher chance of being matched with articles of a different class, and therefore an increased chance of misclassification. A small k value also causes high variance, caused by small fluctations in the training data. A small k also has low bias, as we consider only a few similar articles, but as we increase to larger values of k the bias increases as the threshold for similar articles lowers and we have a less accurate guess. An increasing k value also causes the variance in our tests to reduce from large at $k = 1$ to small at $k = 50$. I think a reasonable value of k would be between 5 to 7 as they has the lowest testing error rate for a k-value more than 1, and has a good balance between bias and variance.

3 New Data Classification

Judging the content of each of these articles, I believe: $1 = Earn, 2 = Crude, 3 = Trade, 4 = Interest, 5 = Earn$. When running my classifier it identifies each class as a different class almost each time I run it. There are 2 main reasons behind this: the training data is randomly selected each time and so therefore the new articles are classified differently each run, and the size of training data I am using. If I increase the training data from a size of 320 to 800 however we get a more static result: $1 = Crude, 2 = Crude, 3 = Interest, 4 = Trade, 5 = Earn$.

4 Confusion Matrix & Classifier Performance

Confusion Matrix and Accuracy						
	Earn	Crude	Trade	Interest	Sport	***Accuracy***
Earn	585	10	9	3	1	97.500%
Crude	2	582	13	7	3	97.000%
Trade	3	4	570	14	3	95.000%
Interest	10	4	8	576	0	96.000%
Sport	0	0	0	0	5	41.667%

The classifier performs very well on the already established classes with lots of training data. These classes very rarely get classified incorrectly and even fewer get classified as "sport". The Sport class on the other hand has limited training data which causes large inaccuracies but performs better than having no training data, which would never classify a sport article as a sport article.

5 Zero-Shot or Few-Shot Learning?

Zero shot learning is where we train a model on already existing classes, and test them on articles of a class or classes that were not observed during training, which the model then needs to classify. Few shot learning is where our training set contains limited data on a given class or classes, but is trained anyways in order to improve accuracy.
I believe that my model is a few-shot learning model as the model is still fed data for the sport class, despite being limited data, which performs better than zero-shot learning.

6 1-NN True Error

To calculate the interval where the true error of a 1-NN classifier lies within 90%, I first ran 20 iterations of the 1-NN test, and calculated a mean error of the testing data which was 3.542%. To calulate the true error, we must calculate

$$a = zp\sqrt{\frac{error(1-error)}{n}}$$

Where $zp = 1.64$ (as we are calculating the probability at 90%), n is our number of training samples (480), and error is our mean error. We calculate a as 0.0138356, and then we calculate the true error interval's upper and lower bounds:

$$error\_interval \in [error - a, error + a]$$

The true error for $k = 1$ is between: 2.158% & 4.925%.

7 45-NN vs 1-NN True Error

The error rate for $k = 45$ for testing is: 3.635%.
The true error interval for $k = 45$ is between: 2.234% & 5.036%.
45-NN has a higher mean sample error, but to calculate the probability that it has a higher true error, we have to use a z-score test. We fist calculate a quantity zp as below:

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

$$d = |error(45\_NN) - error(1\_NN)|$$

$$\sigma = \sqrt{\frac{error(45\_NN)[1 - error(45\_NN)]}{n1} + \frac{error(1\_NN)[1 - error(1\_NN)]}{n2}}$$

Then we calculate the equivalent probability score using the provided "*Get_p_value*" function, and get the final probability:

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

Which I calculated as 52.500%.

8 Hyperparameter Selection

In my experiment, I split the 800 articles into a set of 600 for training, and 200 for validation. You can see this partitioning in the diagram below. I chose k-fold cross validation, with a k-fold value of 5. I iterate over all k-values between 1 and 50, and iterate over all potential k-fold partitions. In total I run 250 tests over my training & testing data, and record the mean error rates of all k values. The final error estimate over all k-values is 27.3%, and the best error rate was 25.542% at $k = 1$. I then ran 100 iterations on the validation set using $k = 1$ and returned a final error rate of 25.000%.