Experiment 1:

kNN classification function has better accuracy when cosine distance is used instead of Euclidean distance for nearest-neighbor calculation. To be more precise, I achieved an accuracy rate of approximately 87% using Euclidean distance, but I obtained an accuracy rate of approximately 96% using Cosine distance. This is because Cosine distance is less affected by the differences in the magnitude of the feature values compared to Euclidean distance. To put it differently, if two data points have similar directions but different magnitudes, Cosine distance will produce a smaller value compared to Euclidean distance. When using the kNN algorithm with Cosine distance as the distance metric, we calculate the angle between the vectors that represent each article. This approach means that we are assessing the similarity between articles based on the direction of their vectors rather than their magnitudes.

Experiment 2:

In general, looking at both graphs, it can be seen that when the value of k increases, both: training and testing errors are increasing. However, in the beginning when k is equal to 1, train error is equal to 0 which is logical while the test error is around 4%. In the beginning our model suffers from overfitting as the train error is lower when compared to the test error, suggesting that the model is not generalizing well to new data. Therefore, in this case, low bias and high variance may be observed. When the value of k is high, both training and testing errors are high suggesting that our model suffers from underfitting meaning that our model is too simple and does not capture the underlying patterns in the data. Therefore, for high values of k we will obtain high bias and low variance. In order to come up with a reasonable value of k we should find the sweet spot where our model is not overfitting nor underfitting while trying to minimize the train error. Following this logic and looking at the graphs, I believe that reasonable k value should be around k = 1 and k =3,4. This where we have the lowest train errors and the testing errors are reasonable.

Experiment 3:

Regarding the first article, I believe it should be classified as "interest" because it centers around a professional tennis player's personal experiences and thoughts about her mother's health and its impact on her tennis career. As for the second article, it's difficult to categorize it using the given categories, but if I had to choose one, I would say "earn" is the most appropriate because it discusses the concept of scoring goals, winning games, and earning points and victories. The third article is also challenging to classify, but I think "interest" would be the best fit as it focuses on the plans of Manchester United's manager to build a culture and develop players over the long term. Moving on to the fourth article, I believe it should also be classified as "interest" because it talks about an upcoming football game between the Philadelphia Eagles and the Kansas City Chiefs and highlights the skills of the players involved, as well as the interest in the match. Finally, for the last article, I think "earn" is the appropriate category because it discusses winning, earning records, and victories.

However, the classifications provided by the classifier - "earn" for article 1, "crude" for article 2, "earn" for article 3, "trade" for article 4, and "trade" for article 5 - are not accurate in my opinion. The categories given are very specific and related more to the economic and business spheres, and they do not accurately capture the content of the articles.

As expected, the model has demonstrated strong performance on the prior four classes, achieving high levels of accuracy between 95-96%. However, the model has exhibited poor performance in classifying the newly added data, with an accuracy of only 33%. This highlights the crucial role that sufficient training data plays in the efficacy of the model. Notably, the model was only trained on a limited dataset of three articles from the 'sports' class, and was evaluated on just two additional examples. To improve the model's accuracy in this new class, it is recommended that additional training data be obtained and incorporated into the model's training process.

Zero-shot learning is a type of machine learning in which a model is trained to recognize objects or concepts that it has never seen before.

Few-shot learning, on the other hand, is a type of machine learning in which a model is trained to recognize objects or concepts with only a small amount of data.

In the context of our scenario, the limited availability of only three articles in the "sports" class indicates that the model has been trained on a small dataset, thereby suggesting a few-shot learning approach.

Result analysis:

In order to find estimate the interval where its true error lies with 90% probability, I have utilized the following formula:

In order to have 90% probability *const* should be equal to 1.64 as suggested in the table in our lecture materials. In the formula *n* is the number of samples tested which in our case is equal to 480(as 320 articles were used to train the model). The variable *error* is the test error.

45-NN has higher testing error than 1-NN. To compute the probability that 45-NN also has a high true error I firstly found z which is equal to d/sigma. Afterwards using z value I have found the p value and inserted it into C = 1 – ((1-p)/2) to find the final probability. To find z I have computed d which is equal to the absolute value difference between the two errors and sigma is equal to . Afterwards I have found value of p using the provided function Get\_p\_value. And then I plugged in the value of p into the formula to find value of C which is the final probability. I have obtained a value of around 98%.

6. Hyperparameter selection

In the present experiment, the dataset has been partitioned into two separate subsets, namely, the training set and the testing set, with a ratio of 80:20 respectively. Such a split is commonly used in machine learning to train a model on a subset of the data and test it on unseen data. Specifically, the training data is employed to fit the model and optimize its hyperparameters, while the testing data is utilized to evaluate the model's performance on previously unseen data. In order to conduct an effective training, I have chosen 160 articles for each class using the provided sample\\_indices function. This means that I will use 4 \* 160 = 640 i.e. 80% of my dataset for training.

To further fine-tune the model, the hyperparameter k (number of neighbors) is selected using k-fold cross-validation. The range of values of k to be tested is defined as 1 to 50, and the model's performance is evaluated using 10-fold cross-validation for each value of k. The use of cross-validation is important as it provides a more reliable estimate of the model's performance than a single train-test split. Moreover, by evaluating the model's performance across multiple folds, it ensures that the model's generalization is not dependent on the particular partitioning of the dataset. I have chosen 10-fold cv as my dataset size is moderate and my research has shown me that values from 5-10 are common choice for such kind of dataset size.

The results of the cross-validation process, namely the accuracy scores for each fold, are stored in an array named cv\\_scores. Subsequently, the k value with the highest cross-validation score is selected as the optimal hyperparameter for the model, which is printed as best\\_k. In this specific case, the best hyperparameter value was determined to be 18, as it produces the highest average accuracy amongst all the other k-values across every fold. Evaluating my model on the testing dataset I have obtained accuracy of 98.75\%.

It is important to emphasize that splitting the dataset into training, testing, and validation sets is a crucial step in machine learning experiments. It helps to avoid the problem of overfitting the model to the training data, which occurs when the model learns to memorize the training data instead of generalizing to new data. The validation set is used to fine-tune the model's hyperparameters, while the testing set is reserved for evaluating the model's performance on previously unseen data. This approach helps to ensure that the model generalizes well to new data and is not biased towards the training data. Without such a partitioning of the data, it would be difficult to know whether the model is overfitting or underfitting the data, which can lead to poor performance on new data.