\

COMP9417 GROUP PROJECT

Submitted by

Team MLD

Kavya Musty (z5240707)
Vandhana Visakamurthy (z5222191)
Yiligong (z5199285)
Zhenzhang Wang (z5276314)

INTRODUCTION

Text classification is a common problem in the field of Machine Learning. A human can read and understand the subtle differences in the English language, but machines can’t read words as words. These words have to be encoded in such a way that machines can understand the same nuances that a human can identify. This is achieved by encoding or vectorizing words to make it machine readable and then training the machine using one of the many available algorithms to segregate text into classes. Common applications include sentiment analysis of tweets in twitter, classification of movie reviews, etc.

Recommendation systems are an important class of machine learning algorithms that offer "relevant" suggestions to users. Recommendation systems encompass a class of techniques and algorithms whose aim is to identify products that best fit user preferences. These techniques are advantageous to both users and vendors, as it enables the user to rapidly find what he needs and the vendors to promote their products and sales. As the industry became aware of the gains that could be accomplished by using such algorithms, the demand for recommendation systems also saw a rapid increase.

This project intends to observe the value of using a recommendation algorithm to find content to recommend to users. In a balanced probabilistic method, this project will show how news topics can be used to recommend news articles. The suggested articles are as relevant to the user as possible so that the user can engage with those articles. We are given a dataset that consists of an article, article text, the category it falls into, and a unique id to identify that article. We initially need to train our machine learning classifier using this data and then take in a new dataset for which we predict the category of topic it falls into. We then use this predicted data and recommend up to 10 articles to each user.

The major issue would be to use an optimum set of hyper-parameters for text vectorization and classification. There is also an issue in recommending articles as the commonly used coefficient metrics won’t be useful.

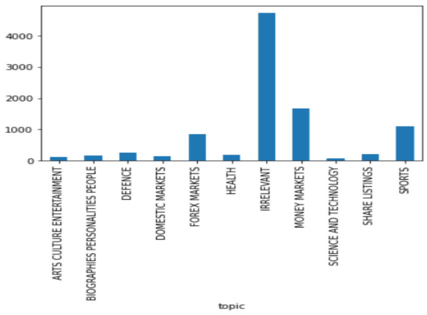
In this project, we used different machine learning methods to determine the topic of an article. To tackle this problem, supervised learning methods such as Multinomial Naïve Bayes, KNN Classifier, Random Forest Classifier, and Support Vector Machine (Linear SVC) are used. All the aforementioned models operate differently and therefore offer various solutions to the given problem.

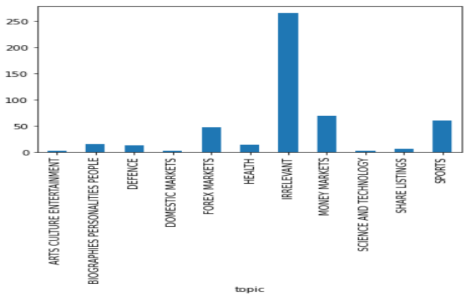
Furthermore, a number of experiments have been presented and discussed to identify the best classifier algorithm that fits best to the problem. Suitable metrics and visualization of such metrics are used to analyze the results.

EXPLORATORY DATA ANALYSIS

After plotting the number of articles per topic using group by method, the frequency distribution of the articles are obtained. A good percentage of the articles, almost 40% are irrelevant. This helps the classifier to differentiate articles as relevant or irrelevant easily.
There are a total of ten classes in terms of topics.

Initially, we analyse the training data set and see the topic distribution which would look like:

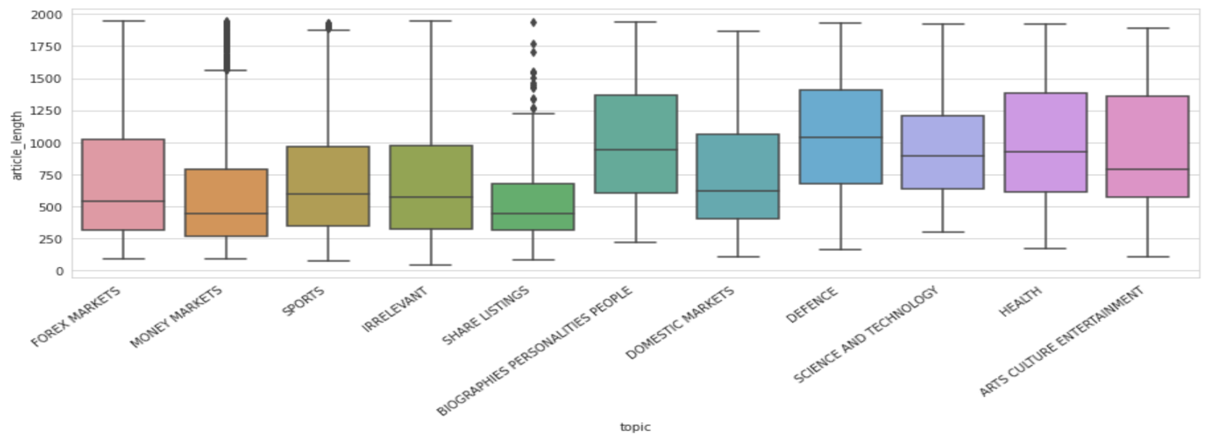




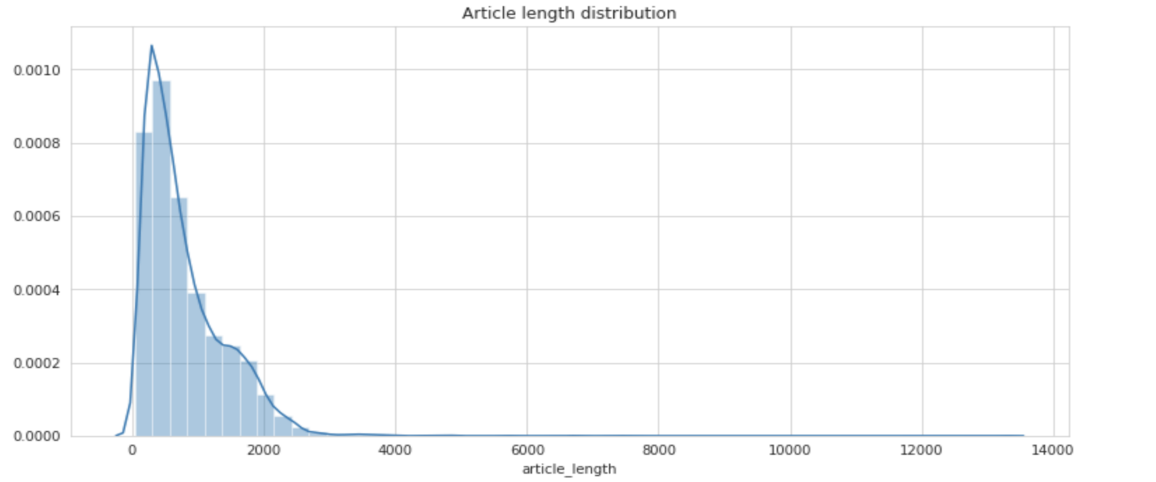
The 'irrelevant' topic has the most articles in the dataset which is then followed by 'money markets'. The least amount of articles belong to the topic 'science and technology'.

In the test dataset, we can observe that the 'irrelevant' topic has the greatest number of articles and the topics 'arts culture and entertainment', 'domestic markets' and 'science and technology' has the least number of article distribution over the test dataset.

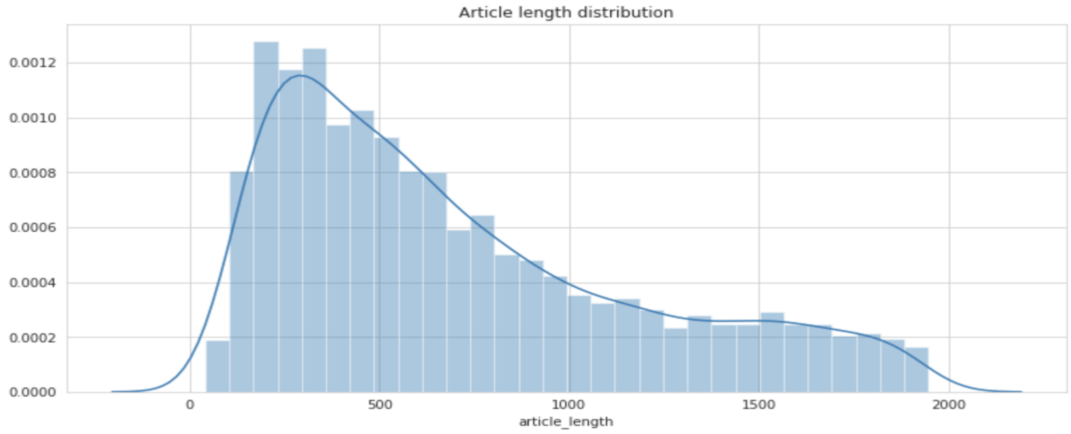
The same topic distribution over the training dataset with respect to the 'article_words' length can be represented using a box plot like:



We can also plot the column 'article_words' by finding out the article length by using a histogram like:



This graph indicates that the average length of each article is around 2000 characters.
A much closer look at the graph gives us a better understanding of how the length of the article doesn't necessarily hinder the classification of text.



METHODOLOGY

Pre-Processing Data:

The given problem statement of classifying articles into their respective topic classes using the keywords in the article is a very important text classification problem. Machines or computers are not advanced enough to read, interpret, and make sense of language like humans. But they can be trained to learn how to make sense of English words if these words can be represented using some form of quantitative representation. A number or a vector value is used to represent words so these values can be "machine read". This is achieved through count vectorization or TF-IDF vectorization. The basic idea of a vectorization method includes using a Bag of Words (BoW) model. Each word is encoded with a vector value and the number of occurrences of each word in a text provides the frequency distribution of every word in that text. This information is used to train the machine to differentiate articles based on the occurrence of certain terms and how many times they occur. The order or grammar of a sentence rarely matters in a BoW model.

CountVectorizer uses the number of unique words in a text to determine the length of the vector that it encodes the value to. The standard binary units of 0 and 1 are used to encode these values. The occurrence of zeroes also increases when the length of the vectors increases, which happens when the number of unique words in a text also increases. This is a fundamental reason that these vectors are called sparse vectors and the matrix used to represent the frequency of such vectors is known as a sparse matrix. This works efficiently for some dataset but there are better vectorizers like the TF-IDF vectorizer.

TF-IDF (Term Frequency- Inverse Document Frequency) vectorizer uses tokens or n-gram words to vectorize instead of vectorizing each word like countvectorizer. These terms are vectorized and weights are used to provide the vectorizer with a higher score based on the frequency of occurrence. A frequently occurring word is assigned the least score and the least occurring word is assigned the highest score. So, commonly occurring words like 'the', and', 'or' are scored less. And words or tuple pairs(n-grams) like 'good book' get a higher score.

Multinomial Naïve Bayes:

The Multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts.

1.Load data and divide

2.Use method to extract feature words

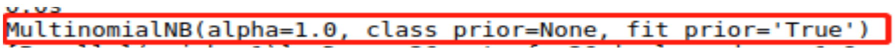
3.Multinomial Naive Bayes Estimator process for estimation

Feature selection:

In this model, I selected the CountVectorizer function to implement the BOW model to extract features. Bag-of-words model first appeared in the field of Natural Language Processing and Information Retrieval. The model ignores elements such as the grammar and word order of text, and treats it only as a collection of vocabularies. Each word in the document appears independently. The BoW model uses a disordered set of words to express a text or a document, so I chose this one.

Hyper-parameter tuning:

The parameters that were used for MultinomialNB include **alpha(*default=1.0*), fit_prior(*default=True*) *and* class_prior(*default=None*).** Alpha is an additive (Laplace/Lidstone) smoothing parameter, fit_prior determines whether to learn class prior probabilities or not and class_prior represents prior probabilities of the classes. GridSearchCV is used to find the optimum parameters.
The parameters "alpha": [0.01, 0.1, 0.5, 1.0, 10, 100] and "fit_prior": ["True", "False"] is used. A total of two-fold cross validation was done and the data was refit as well.
The best parameter after performing GridSearchCv is as follows:



```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior='True')
```

The performance after using the above-mentioned parameters is as follows:

Accuracy Score for training data : 0.8232631578947368.

Accuracy Score for testing data : 0.728.

Evaluation metrics:

In the first case, the prediction is positive and the actual is also positive. We call it true positive (TP). In the second case, the prediction is positive and the actual is negative. We call it false positive (FP). The third case, the prediction is negative, the actual is positive. We call it false negative (FN). The last case, the prediction is negative, the actual is also negative, called true negative (TN). There is no other possibility.

Obviously, given a test set, we can get the following relationship:

$$N_{pre} = TP + TN$$

$$N_{total} = TP + TN + FP + FN$$

If we define a test set, the number of positive samples is P, and the number of negative samples is N, we can define recall, precision, F1-score as follows:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} = \frac{TP}{P}$$

$$F1 = \frac{2TP}{2TP + FN + FP} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

It can be seen that recall reflects the classification model's ability to identify positive samples. The higher the recall, the stronger the model's ability to identify positive samples. Precision reflects the model's ability to distinguish negative samples. The higher the precision, the better the model The stronger the discrimination ability of negative samples. F1-score is a combination of the two. The higher the F1-score, the more robust the classification model.

Therefore, F1-score is chosen here.

KNN Classifier:

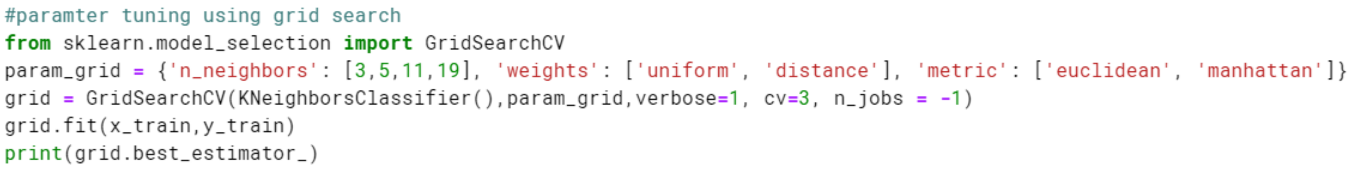
KNN algorithm is used to classify by finding the K nearest matches in training data and then using the label of closest matches to predict. After pre-processing the given dataset and extracting all the article words from both the datasets and fitting them into the bag of words, we have the frequency of occurrences of words of every article in the dataset. We have to now find the optimal parameters to train the model.

Feature Selection:

Initially, without using any parameters in the KNN classifier, we get an accuracy of 70.2% by passing the pre-processed data and bag of words using TDIF Vectorizer which yields a much more accurate data than the count vectorizer. In order to improve this accuracy, we can use a GridSearchCV() method to get the optimal parameters to train the model.

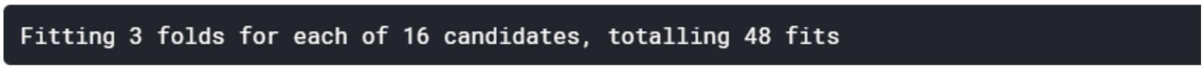
Hyperparameter Tuning:

The hyper parameters used for the KNeighborsClassifier() is ‘n_neighbors’, 'weights', 'metric'. The ‘weights’ parameter can be set to either ‘uniform’, where each neighbour within the boundary carries the same weight or ‘distance’ where closer points will be more heavily weighted toward the decision. The parameter ‘metric’ refers to how the distance of neighbouring points is chosen from the unknown point.The best parameter can be found by feeding this data to the GridSearchCV() as below:

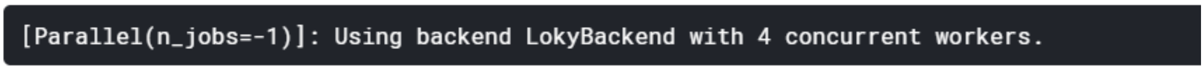


```
#paramter tuning using grid search
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': [3,5,11,19], 'weights': ['uniform', 'distance'], 'metric': ['euclidean', 'manhattan']}
grid = GridSearchCV(KNeighborsClassifier(),param_grid,verbose=1, cv=3, n_jobs = -1)
grid.fit(x_train,y_train)
print(grid.best_estimator_)
```

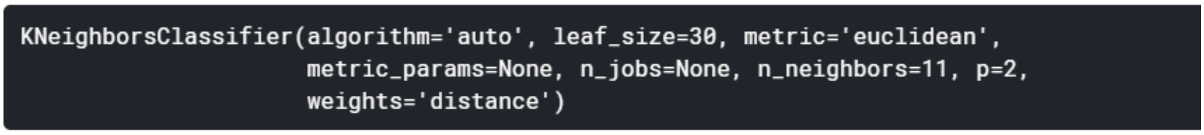
This performs a 3 folds cross validation and provides a result like:



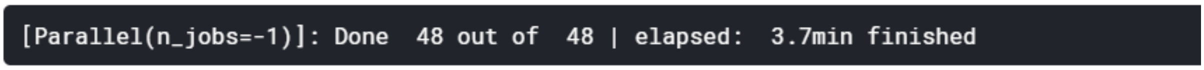
```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```



```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
```



```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                     weights='distance')
```



```
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:  3.7min finished
```

Evaluation metrics:

The above are the best parameters returned from grid search cross validation. Using the above model to predict the test set, the accuracy that was obtained is 70.8%.

Random Forest Classifier:

Random Forest Classifier is an ensemble method of multiple trees which will take random subsets of features instead of all the features. This method can reduce overfitting problems because it selects only the subset of features which is really useful when we have high dimensional data.

Feature Selection:

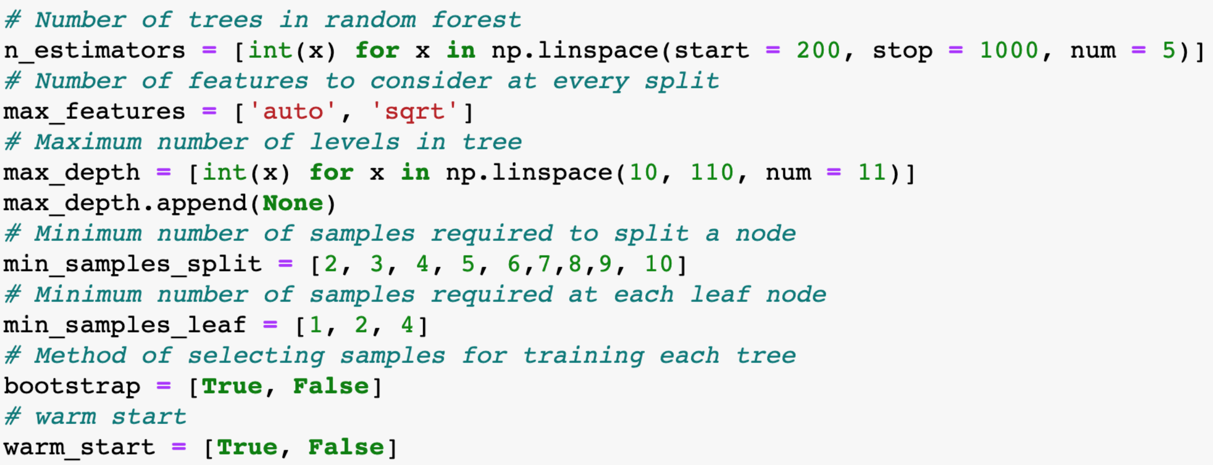
TF-IDF is selected as the vectorizer to represent the data. In the function TF-IDF, several parameters which can better represent a larger dimensional data are chosen.

The main parameters are ngram_range, max_df, min_df and max_features. Ngram_range is the range of the extracted n-grams. A range from 1 to 2 is chosen, which means the only values to be considered are unigram and bigram from the extracted text (reason). Max_df is the max frequency when building the vocabulary which means if some terms' frequency is strictly higher than the above value then those will be ignored. 1.0 is chosen here, which will not ignore any terms. Min_df is the opposite meaning of Max_df. 10 is chosen here which will ignore those terms which frequency is lower than this value. The next parameter is max_features which represent the top n features that with higher frequency across the corpus and the value chosen here is 1000. If we do not do a feature selection here, then the final vector matrix will have more than 30,000 features, which is a huge number and will cause some problems like curse of dimension in the later training process.

By using the above model to fit the text part in training data and then transform the training data and test data, we can get two vectors with same dimensions and one is our training data vector and another is test data vector. Then we use those two vectors to do the training process.

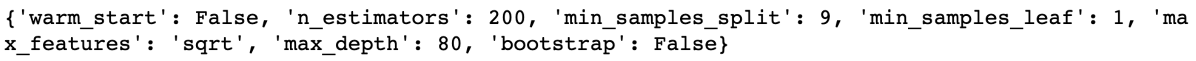
Hyper-Parameter Tuning:

In random forest classification, the parameters chosen are n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, max_features, max_leaf_nodes, bootstrap, and warm_start. When deciding which are the values that can maximise the performance of the models, gridsearchCV is used for the above parameters by doing a random parameters search on the values below.



```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 1000, num = 5)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 3, 4, 5, 6,7,8,9, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# warm start
warm_start = [True, False]
```

By using RandomizedSearch cross validation function, I do a three fold cross validation and each fold has 50 parameters settings sampled. Then use this model to train our training data with training labels.



```
{'warm_start': False, 'n_estimators': 200, 'min_samples_split': 9, 'min_samples_leaf': 1, 'ma
x_features': 'sqrt', 'max_depth': 80, 'bootstrap': False}
```

The above are the best parameters returned from random search cross validation. Using the above model to predict the test set, I can get 75% accuracy on the test set.

Support Vector Machine:

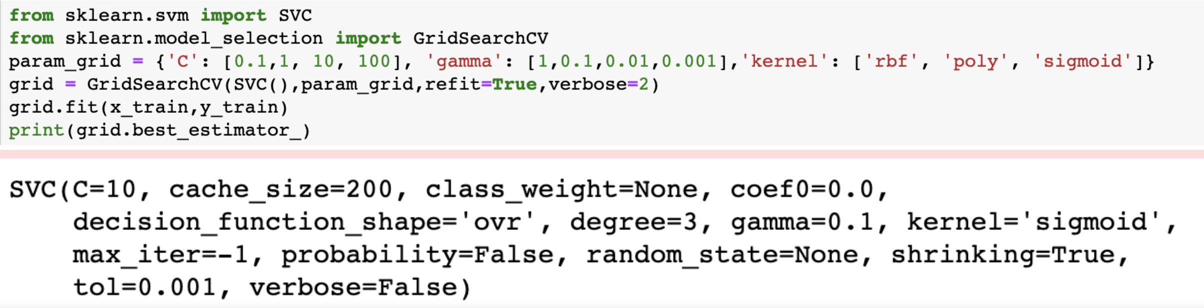
Feature Selection:

After pre-processing the article words, and fitting the words into the BoW model, we have the frequency of occurrences of words of every article in the dataset. Using the given data to train the classifiers to identify the class of topic can be done easily. The challenge arises in using the optimum parameters for the model. TF-IDF and Count Vectorizer was used. TF-IDF is more accurate from the results. Count Vectorizer provides an accuracy of 0.716 whereas TF-IDF boosts the accuracy to 0.768.

SVM works by classifying data points into classes using a separating line. This approximation is done by calculating all possible lines or hyperplanes and choosing the hyperplane that can separate most of the classes efficiently. For the given problem statement, the optimum parameters were found using GridSearchCV.

Hyper - Parameter Tuning:

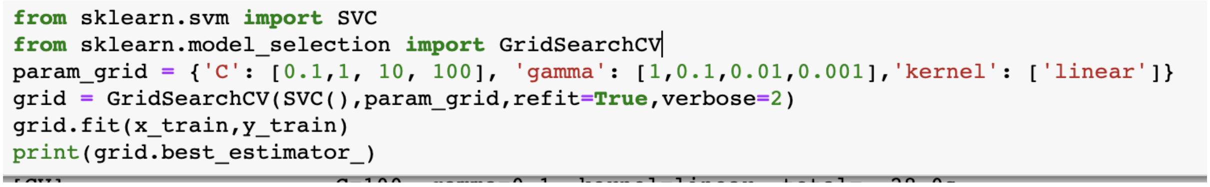
The parameters that were used for SVM include ‘C’, gamma, and kernel. ‘C’ is the regularization parameter and gamma is the kernel coefficient. 1 / (n_features * X.var()) is the default value of gamma. Auto uses 1/no of features instead. C values of [0.1,1,10,100] were used for gamma values [1,0.1,0.01,0.001]. The kernels used were linear, rbf, poly, and sigmoid. A total of three-fold cross validation was done and the data was refit as well.
The best parameter after performing GridSearchCv for rbf, poly and sigmoid kernels is as follows:



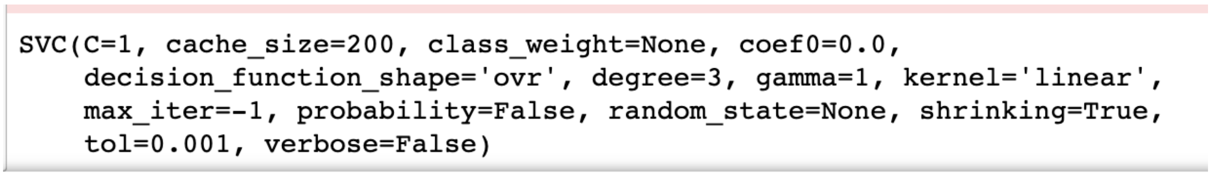
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['rbf', 'poly', 'sigmoid']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(x_train,y_train)
print(grid.best_estimator_)
```

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

The performance after using the above-mentioned parameters is as follows:
Accuracy Score for training data: 0.8970526315789473.
Accuracy Score for testing data: 0.768.
The best parameter after performing GridSearchCv for linear kernel is as follows:



```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['linear']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(x_train,y_train)
print(grid.best_estimator_)
```



```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

The performance after using the above-mentioned parameters is as follows:
Accuracy Score for training data: 0.8978947368421053.
Accuracy Score for testing data : 0.768.
The Linear Kernel with the parameters below is the best

SVC(C=1, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False) provides the best accuracy with no problem of overfitting.

Recommendation System

The key point of developing this recommendation system is using cosine similarity. Cosine similarity is a method to measure the similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

A and B are two vectors. In our implementation, A and B would be the sparse matrix return from TF-IDF vectorizing. $||A||$ and $||B||$ are the Euclidean norm of vector A and B. If two vectors are really similar to each other, then the cosine similarity will be really close to 1 which means the angle will be close to 0 degree, otherwise it will be close to -1.

Implementation details:

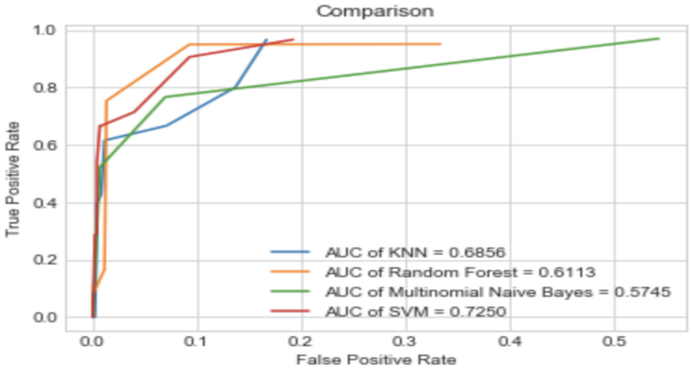
We use the predicted value to replace the original topic of the testing data set. Then we use the tf-idf vector to fit the text in training data. We have a list with all the topic names and we do a vectorizing on both training data and testing data on each topic. Then we do a cosine similarity on those two sparse matrices and the returned value would be a matrix with shape (N,M). N is the number of how many articles on that topic in testing data and M is the number of how many articles on that topic in training data. By sorting the matrix, we can select the best 10 out of those similarity values with highest scores. At same time, we also keep recording the article numbers each time we do the sorting on each topic. After we get all the article numbers for all topics, we get the true labels and the predicted labels according to those article numbers on the test set, which will be helpful for calculating precision, recall and f1.

And the final result is shown below. From the result below, the topic Science and Technology does have any recommendations. This is because we didn't predict any articles which should belong to Science and Technology correctly.

```
For topic ARTS CULTURE ENTERTAINMENT recommending article 9703,9830,9933,9952
For topic BIOGRAPHIES PERSONALITIES PEOPLE recommending article 9896,9526,9988,9940
For topic DEFENCE recommending article 9607,9770,9616,9670,9559,9759,9987,9576,9773
For topic DOMESTIC MARKETS recommending article 9640,9989
For topic FOREX MARKETS recommending article 9671,9565,9530,9529,9977,9551,9986,9682,9588,979
8
For topic HEALTH recommending article 9982,9947,9810,9661,9873,9621,9929,9807,9833,9926
For topic MONEY MARKETS recommending article 9961,9808,9678,9766,9725,9555,9761,9853,9634,958
6
For topic SHARE LISTINGS recommending article 9562,9666,9715,9601,9518
For topic SPORTS recommending article 9787,9992,9791,9754,9630,9752,9608,9513,9520,9800
```

EVALUATION

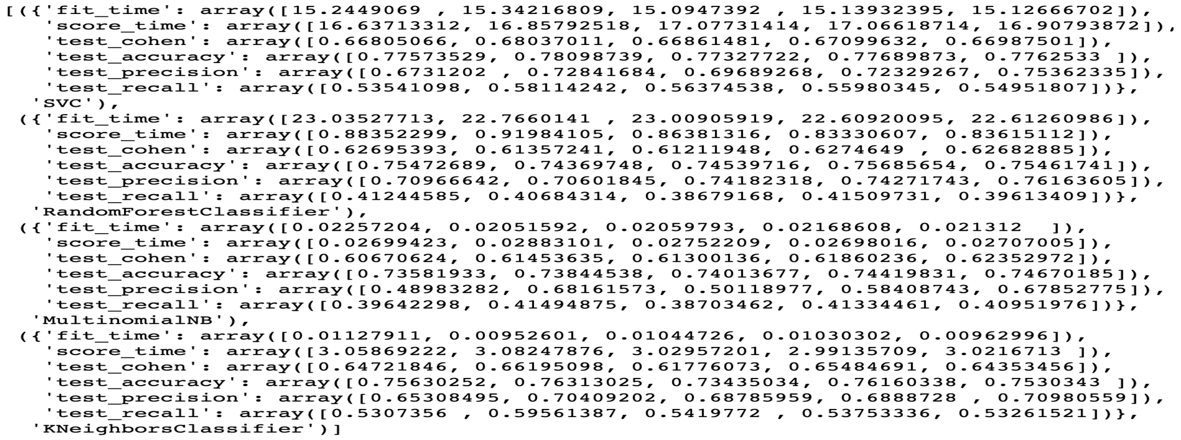
The true positive rate and the false positive rate of each of the classifiers for the test data gives an idea of how efficient each classifier is and allows us to compare the performance based on the AUC score. The plot given below visualises this.



RESULTS

1. Cross-validation results on the training set for selected metrics, feature sets and implemented methods.

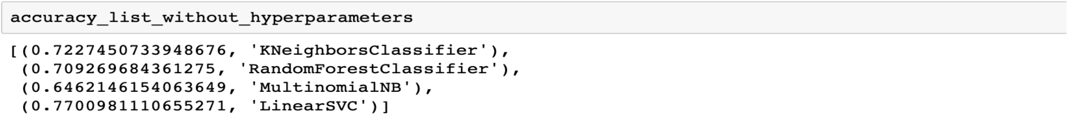
For selected metrics:



```
[({'fit_time': array([15.2449069 , 15.34216809, 15.0947392 , 15.13932395, 15.12666702]),
   'score_time': array([16.63713312, 16.85792518, 17.07731414, 17.06618714, 16.90793872]),
   'test_cohen': array([0.66805066, 0.68037011, 0.66861481, 0.67099632, 0.66987501]),
   'test_accuracy': array([0.77573529, 0.78098739, 0.77327722, 0.77689873, 0.7762533 ]),
   'test_precision': array([0.6731202 , 0.72841684, 0.69689268, 0.72329267, 0.75362335]),
   'test_recall': array([0.53541098, 0.58114242, 0.56374538, 0.55980345, 0.54951807])},
  'SVC'),
 ({'fit_time': array([23.03527713, 22.7660141 , 23.00905919, 22.60920095, 22.61260986]),
   'score_time': array([0.88352299, 0.91984105, 0.86381316, 0.83330607, 0.83615112]),
   'test_cohen': array([0.62695393, 0.61357241, 0.61211948, 0.6274649 , 0.62682885]),
   'test_accuracy': array([0.75472689, 0.74369748, 0.74539716, 0.75685654, 0.75461741]),
   'test_precision': array([0.70966642, 0.70601845, 0.74182318, 0.74271743, 0.76163605]),
   'test_recall': array([0.41244585, 0.40684314, 0.38679168, 0.41509731, 0.39613409])},
  'RandomForestClassifier'),
 ({'fit_time': array([0.02257204, 0.02051592, 0.02059793, 0.02168608, 0.021312  ]),
   'score_time': array([0.02699423, 0.02883101, 0.02752209, 0.02698016, 0.02707005]),
   'test_cohen': array([0.60670624, 0.61453635, 0.61300136, 0.61860236, 0.62352972]),
   'test_accuracy': array([0.73581933, 0.73844538, 0.74013677, 0.74419831, 0.74670185]),
   'test_precision': array([0.48983282, 0.68161573, 0.50118977, 0.58408743, 0.67852775]),
   'test_recall': array([0.39642298, 0.41494875, 0.38703462, 0.41334461, 0.40951976])},
  'MultinomialNB'),
 ({'fit_time': array([0.01127911, 0.00952601, 0.01044726, 0.01030302, 0.00962996]),
   'score_time': array([3.05869222, 3.08247876, 3.02957201, 2.99135709, 3.0216713 ]),
   'test_cohen': array([0.64721846, 0.66195098, 0.61776073, 0.65484691, 0.64353456]),
   'test_accuracy': array([0.75630252, 0.76313025, 0.73435034, 0.76160338, 0.7530343 ]),
   'test_precision': array([0.65308495, 0.70409202, 0.68785959, 0.6888728 , 0.70980559]),
   'test_recall': array([0.5307356 , 0.59561387, 0.5419772 , 0.53753336, 0.53261521])},
  'KNeighborsClassifier')]
```

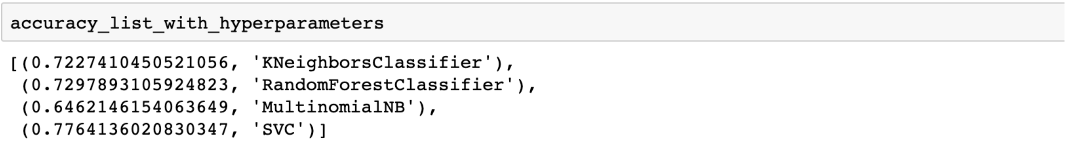
Metrics (mean)	SVM	KNN	RFC	MultiNB
cohen	0.67	0.64	0.62	0.61
accuracy	0.77	0.75	0.75	0.74
precision	0.71	0.68	0.73	0.58
recall	0.55	0.54	0.40	0.40

For implemented methods:



```
accuracy_list_without_hyperparameters
```

```
[(0.7227450733948676, 'KNeighborsClassifier'),
 (0.709269684361275, 'RandomForestClassifier'),
 (0.6462146154063649, 'MultinomialNB'),
 (0.7700981110655271, 'LinearSVC')]
```



```
accuracy_list_with_hyperparameters
```

```
[(0.7227410450521056, 'KNeighborsClassifier'),
 (0.7297893105924823, 'RandomForestClassifier'),
 (0.6462146154063649, 'MultinomialNB'),
 (0.7764136020830347, 'SVC')]
```

Metrics selected to evaluate the models are Cohen's Kappa, accuracy score, precision score and recall score. Cohen's Kappa is one of the best metrics for evaluating multi-class classifiers especially when we have imbalanced data sets. Accuracy score is the probability of how many classes the model predicted correctly. Precision score is the ability of how many selected items are relevant. Recall score is the ability of how many relevant items are selected.

From the evaluation metrics above, SVM always performs the best among all four classifiers.

2. Final results for each class calculated on the whole test set using the final selected method with its hyper-parameters

Topic Name	Precision	Recall	F1
ARTS CULTURE ENTERTAINMENT	0.50	0.67	0.57
BIOGRAPHIES PERSONALITIES PEOPLE	0.75	0.20	0.32
DEFENCE	0.78	0.54	0.64
DOMESTIC MARKETS	0.00	0.00	0.00
FOREX MARKETS	0.52	0.35	0.42
HEALTH	0.83	0.71	0.77
MONEY MARKETS	0.49	0.67	0.56
SCIENCE AND TECHNOLOGY SHARE LISTINGS	0.00	0.00	0.00
SHARE LISTINGS	0.60	0.43	0.50
SPORTS	0.95	0.98	0.97

3. Final article recommendations using the final selected method with its hyper-parameters.

Topic Name	Suggested Articles	Precision	Recall	F1
ARTS CULTURE ENTERTAINMENT	9703,9830,9933, 9952	0.50	1.00	0.67
BIOGRAPHIES PERSONALITIES PEOPLE	9896,9526,9988, 9940	0.75	0.60	0.67
DEFENCE	9607,9770,9616, 9670,9559,9759, 9987,9576,9773	0.78	1.00	0.88
DOMESTIC MARKETS	9640,9989	0.00	0.00	0.00
FOREX MARKETS	9671,9565,9530, 9529,9977,9551, 9986,9682,9588, 9798	0.60	0.60	0.60
HEALTH	9982,9947,9810, 9661,9873,9621, 9929,9807,9833, 9926	0.80	1.00	0.89

MONEY MARKETS	9961,9808,9678, 9766,9725,9555, 9761,9853,9634, 9586	0.50	0.56	0.53
SCIENCE AND TECHNOLOGY		0.00	0.00	0.00
SHARE LISTINGS	9562,9666,9715, 9601,9518	0.60	1.00	0.75
SPORTS	9787,9992,9791, 9754,9630,9752, 9608,9513,9520, 9800	0.90	1.00	0.95

DISCUSSION

Models	Advantages	Disadvantages
Multinomial Naïve Bayes	1.The classification efficiency is stable. 2.Less sensitive to the actual data, and the algorithm is relatively simple, often used for text classification. 3.fast speed.	We need to know the prior probability P (F1, F2 ... | C) Therefore, the reason for the hypothetical prior model that hesitates at certain times leads to poor prediction results.
KNN Classifier	1.It is simple and mature, suitable for classification and regression. 2.It is suitable for non-linear regression.	1. The amount of calculation of the test sample is large, and the memory overhead is large. 2. The k value must be specified. If the k value is not properly selected, the classification accuracy cannot be guaranteed
Random Forest Classifier	1.It has excellent accuracy. 2.Can effectively run on a large dataset. 3.Ability to handle high-dimensional features without dimensionality reduction. 4.The results of the default value can also live well.	Overfitting on some noisy classification or regression problems; for data with different values of attributes, attributes with more values divided will have a greater impact on RF, resulting in unreliable weights.
SVM	1.Can be used for linear/non-linear. 2.Can be used for regression SVR, low generalization error, easy to explain, anti-data disturbance, small storage .	1.For large dataset, it may be not suitable. 2.It does not support multiple classification.

From the precision, recall and F1 table, we can find out that some articles could get pretty high precision scores but really low recall scores. Especially when we have a large data sets but with imbalanced class distribution, we probably can get a really good precision score but a poor recall score if that class is supported by larger number of data sets. So in this way, we should use a trade-off between precision and recall score which is F1 score. F1 score conveys the balance between the precision and recall. If we can find a higher F1 score, then the corresponding precision and recall score can also be pretty good.

$$F1 = \frac{2TP}{2TP + FN + FP} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

CONCLUSION

Of the four classifiers, SVM outperforms the other classifiers and the performs better. The recommendation of articles by category is accurate by manual inspection.

REFERENCES

1.Text classification:
https://towardsdatascience.com/text-classification-in-python-dd95d264c802
https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568

2.Recommendation system and evaluating metrics:
http://fastml.com/evaluating-recommender-systems/
https://www.machinelearningplus.com/nlp/cosine-similarity/

3.Plotting metrics:
https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
https://towardsdatascience.com/understanding-data-science-classification-metrics-in-scikit-learn-in-python-3bc336865019

4.Bayesian Multinomial Naïve Bayes Classifier to Text Classification
https://www.researchgate.net/publication/317173563_Bayesian_Multinomial_Naive_Bayes_Classifier_to_Text_Classification

5.Classification and Regression by RandomForest
https://www.researchgate.net/profile/Andy_Liaw/publication/228451484_Classification_and_Regression_by_RandomForest/links/53fb24cc0cf20a45497047ab/Classification-and-Regression-by-RandomForest.pdf

6.Weinberger, K.Q., Blitzer, J. and Saul, L.K., 2006. Distance metric learning for large margin nearest neighbor classification. In Advances in neural information processing systems (pp. 1473-1480).

7.Understanding Support Vector Machine(SVM) algorithm from examples (along with code)
https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

8.Natural Language Processing: Count Vectorization with scikit-learn
https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e

9.Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents
https://www.researchgate.net/publication/326425709_Text_Mining_Use_of_TF-IDF_to_Examine_the_Relevance_of_Words_to_Documents