Middle Project Report

Sequential Team

- Jeremy Vijay Wongso	(109711007)
- Kholishotul Amaliah	(109711008)
- Bastian Farandy	(109711009)

TEXT MINING (EE100098)

VIRTUAL EXCHANGE PROGRAM

ASIA UNIVERSITY

FALL SEMESTER

METHODS

Specification of hardware used :

- CPU : Intel Core i7-7700HQ (4 Core, 8 Thread, 2.8 Ghz)
- GPU : Nvidia GTX 1060 6GB
- RAM : 16GB

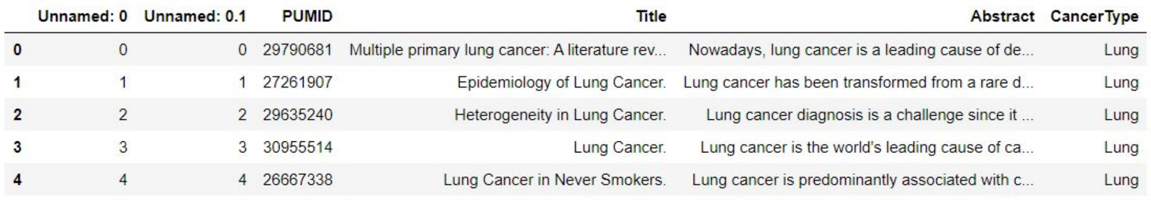
Specification of software used :

- Python version : 3.7.7

Steps :

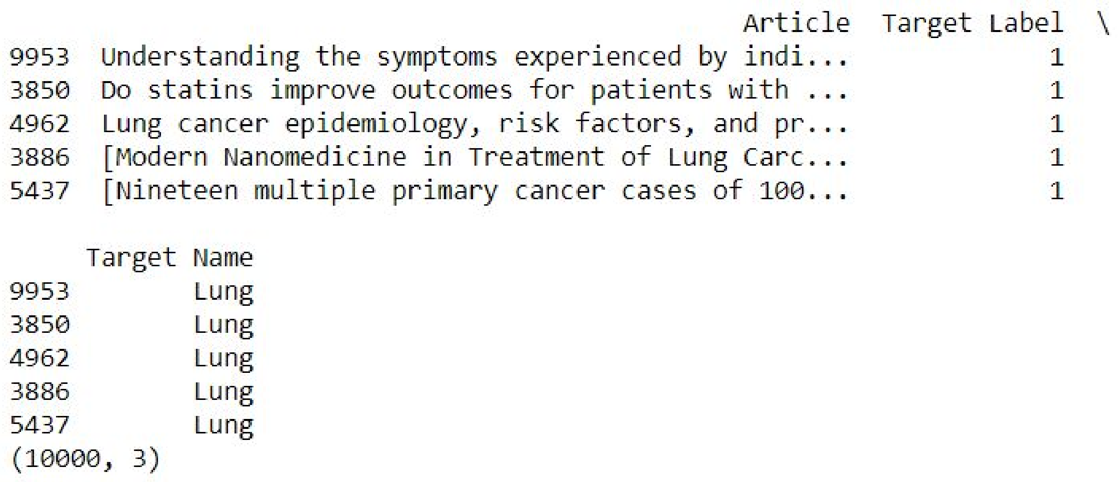
1. Data Needed

    The initial data is 100.000 information of top 10 cancer types from PubMed. The data has 6 columns, which are 2 unnamed columns (filled with index from the previous work), PUMID, Title, Abstract, and CancerType. Here is the snippet of the initial data.



	Unnamed: 0	Unnamed: 0.1	PUMID	Title	Abstract	CancerType
0	0	0	29790681	Multiple primary lung cancer: A literature rev...	Nowadays, lung cancer is a leading cause of de...	Lung
1	1	1	27261907	Epidemiology of Lung Cancer.	Lung cancer has been transformed from a rare d...	Lung
2	2	2	29635240	Heterogeneity in Lung Cancer.	Lung cancer diagnosis is a challenge since it ...	Lung
3	3	3	30955514	Lung Cancer.	Lung cancer is the world's leading cause of ca...	Lung
4	4	4	26667338	Lung Cancer in Never Smokers.	Lung cancer is predominantly associated with c...	Lung

    For extra information, we concatenate title and abstract to be content. And also do the mapping for each cancer type become numerical. To have faster computation, we choose randomly 1000 data from each cancer type. So now we have 10000 rows of data.



```
                                                Article  Target Label  \
9953  Understanding the symptoms experienced by indi...             1
3850  Do statins improve outcomes for patients with ...             1
4962  Lung cancer epidemiology, risk factors, and pr...             1
3886  [Modern Nanomedicine in Treatment of Lung Carc...             1
5437  [Nineteen multiple primary cancer cases of 100...             1

     Target Name
9953        Lung
3850        Lung
4962        Lung
3886        Lung
5437        Lung
(10000, 3)
```

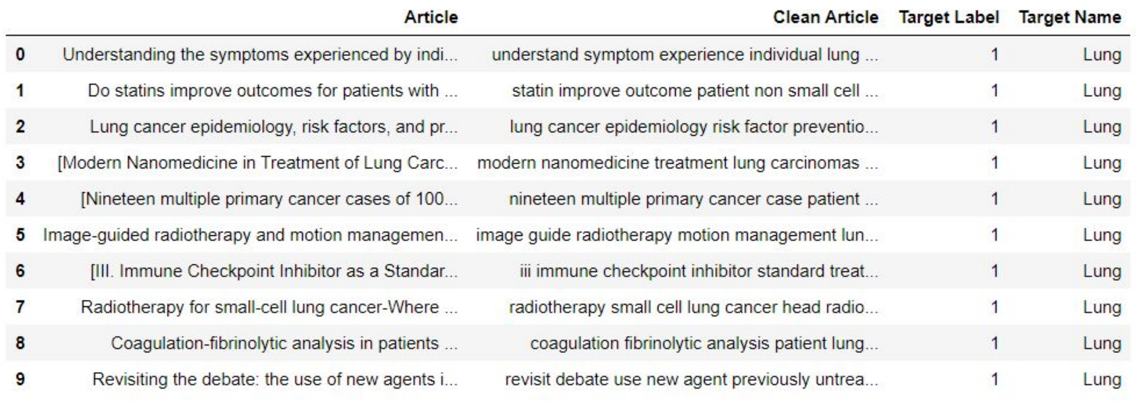
2. Text Preprocessing

Text wrangling (also called preprocessing or normalization) is a process that consists of a series of steps to wrangle, clean, and standardize textual data into a form that could be consumed by other NLP and intelligent systems powered by machine learning and deep learning. We use html stripping, contraction expansion, accented char removal, text lower case, text lemmatization, special character removal, digits removal, and stopwords removal methods.

For implementation, we use *BeautifulSoup* text scraping for the html stripping. Then for the contraction expansion, we use the *contraction* library for expanding the shortened word or syllables. Accented char removal is a process to change the accented character to ascii format. After removing the accented character, we lower all the case so that the word becomes in the same case. Then we also have lemmatization from *spacy* library to have the base form of words. For only getting the text, we remove the special character (symbol) and digits. And last, we remove the stopwords to get the more important word by using *ToktokTokenizer* from *nltk* library.

3. Dataset

So, the dataset of our middle project is clean data. It is saved under the name PubMed_CleanArticles_Top1-10_cancerTypes.csv in the Output folder.



	Article	Clean Article	Target Label	Target Name
0	Understanding the symptoms experienced by indi...	understand symptom experience individual lung ...	1	Lung
1	Do statins improve outcomes for patients with ...	statin improve outcome patient non small cell ...	1	Lung
2	Lung cancer epidemiology, risk factors, and pr...	lung cancer epidemiology risk factor preventio...	1	Lung
3	[Modern Nanomedicine in Treatment of Lung Carc...	modern nanomedicine treatment lung carcinomas ...	1	Lung
4	[Nineteen multiple primary cancer cases of 100...	nineteen multiple primary cancer case patient ...	1	Lung
5	Image-guided radiotherapy and motion managemen...	image guide radiotherapy motion management lun...	1	Lung
6	[III. Immune Checkpoint Inhibitor as a Standar...	iii immune checkpoint inhibitor standard treat...	1	Lung
7	Radiotherapy for small-cell lung cancer-Where ...	radiotherapy small cell lung cancer head radio...	1	Lung
8	Coagulation-fibrinolytic analysis in patients ...	coagulation fibrinolytic analysis patient lung...	1	Lung
9	Revisiting the debate: the use of new agents i...	revisit debate use new agent previously untrea...	1	Lung

4. Text Mining

Our text mining focus is text classification. We divide into 3 scenarios.

1) Text Classification using Term Frequency (TF)

The Bag of Words model, usually known as term frequency, represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0), or even weighted values. The model's name is such because each document is represented literally as a bag of its own words, disregarding word order, sequences, and grammar.

For implementation, we split the processed data into 2 sets, which are training set and test set. With ratio 0.67 for the training set and 0.33 for the test set. We use the *CountVectorizer* from sklearn library to extract the feature.

2) Text Classification using Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF stands for term frequency-inverse document frequency. It's a combination of two metrics, term frequency (tf) and inverse document frequency (idf). Term frequency, denoted by tf, is what we computed in the Bag of Words model in the previous section. Term frequency in any document vector is denoted by the raw frequency value of that term in a particular document. Inverse document frequency denoted by idf is the inverse of the document frequency for each term and is computed by dividing the total number of documents in our corpus by the document frequency for each term and then applying logarithmic scaling to the result.

For implementation, we split the processed data into 2 sets, which are training set and test set. With ratio 0.67 for the training set and 0.33 for the test set. We use the *TfidfVectorizer* from sklearn library to extract the feature.

3) Text Classification using TF-IDF with Tuning Parameter

For implementation, we split the processed data into 2 sets, which are training set and test set. With ratio 0.67 for the training set and 0.33 for the test set. We use the *TfidfVectorizer* from sklearn library to extract the feature. For tuning our model, we use *GridSearchCV* and *RandomizedSearchCV*.

GridSearchCV implements a "fit" and a "score" method. It also implements "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

RandomizedSearchCV implements a "fit" and a "score" method. It also implements "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings.

In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter. If all parameters are presented as a list, sampling without replacement is performed. If at least one parameter is given as a distribution, sampling with replacement is used. It is highly recommended to use continuous distributions for continuous parameters.

Then for each scenarios, we use 6 models of classification, which are :

1) Multinomial Naïve Bayes

The Naïve Bayes algorithm is a supervised learning algorithm from Bayes algorithm that each feature is completely independent of the others. Multinomial Naïve Bayes is an extension of the algorithm for predicting and classifying data points, where the number of distinct classes or outcomes are more than two. In this case, the feature vectors are usually assumed to be word counts from the bag of words model, but TF-IDF-based weights also work. One limitation is that negative weight based features can't be fed into this algorithm.

For implementation, we use the *MultinomialNB* package from sklearn library. And we set alpha value to 1. For more accuracy, we set the cross validation value to 5.

2) Logistic Regression

The logistic regression model is a logistic model that uses the logistic mathematical function to estimate the parameter values. Considering a binary classification problem of predicting two classes, a 0 or a 1, in the logistic model, the log-odds (the logarithm of the odds) for the class/category labeled as 1 are basically the equation of the linear regression model (linear combination of one or more independent features, which can be categorical or continuous). However, we need to predict discrete classes or categories. Thus, the corresponding probability of the class labeled 1 can vary between 0 and 1, depicting the confidence of the prediction. The function that helps us convert the log-odds to probability is the logistic function.

For implementation, we use the *LogisticRegression* package from sklearn library. And we set the penalty value to l2, the maximum iteration is 100, and the C value is 1. For more accuracy, we set the cross validation value to 5.

3) Linear SVM

In machine learning, support vector machines, known popularly as SVMs, are supervised learning algorithms. They are used for classification, regression, novelty and anomaly, and outlier detection. Considering a binary classification problem, if we have training data such that each data point or observation belongs to a specific class, the SVM algorithm can be trained based on this data such that it can assign future data points into one of the two classes. This algorithm represents the training data samples as points in space such that points belonging to either class can be separated by a wide gap between them (hyperplane) and the new data points to be predicted are assigned classes based on which side of this hyperplane they fall into. This process is for a typical linear classification process. However, SVM can also perform non-linear classification by an interesting approach known as a kernel trick, where kernel functions are used to operate on high-dimensional feature spaces that are non-linear separable. Usually, inner products between data points in the feature space help achieve this.

For implementation, we use the *LinearSVC* package from sklearn library. And we set the penalty value to l2 and the C value to 1. For more accuracy, we set the cross validation value to 5.

4) SGD

SGDClassifier is a Linear classifier (SVM, logistic regression, a.o.) with SGD training. This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning, see the partial_fit method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

For implementation, we use the *SGDClassifier* package from sklearn library. And we set the loss value to hinge, the penalty value to l2, and the maximum iteration is 5. For more accuracy, we set the cross validation value to 5.

5) Random Forest

A random forest is a meta-estimator or an ensemble model that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. In random forests, all the trees are trained in parallel (bagging model/bootstrap aggregation). Besides this, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. Also, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. Thus the randomness introduced in a random forest is both due to random sampling of data and random selection of features when splitting nodes in each tree. Hence, due to this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random decision tree). However, due to averaging, the overall variance of the model decreases significantly as compared to the increase in bias and hence it gives us an overall better model.

For implementation, we use the *RandomForestClassifier* package from sklearn library. And we set n_estimator to 10. For more accuracy, we set the cross validation value to 5.

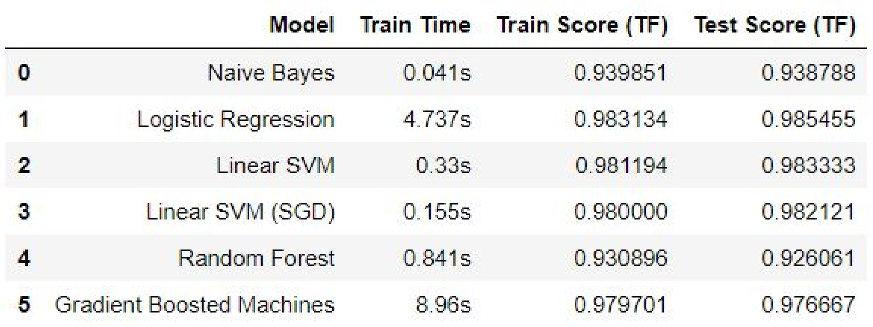
6) Gradient Boosted Machine

Gradient boosting machines, popularly known as GBMs, build an additive model in a forward stagewise sequential fashion; they allow for the optimization of arbitrary differentiable loss functions. GBMs can usually work on any combination of models (weak learners) and loss functions. Scikit-Learn uses GBRTs (Gradient Boosted Regression Trees), which are generalized boosting models that can be applied to arbitrary differentiable loss functions. The beauty of this model is that it is accurate and can be used for both regression and classification problems.

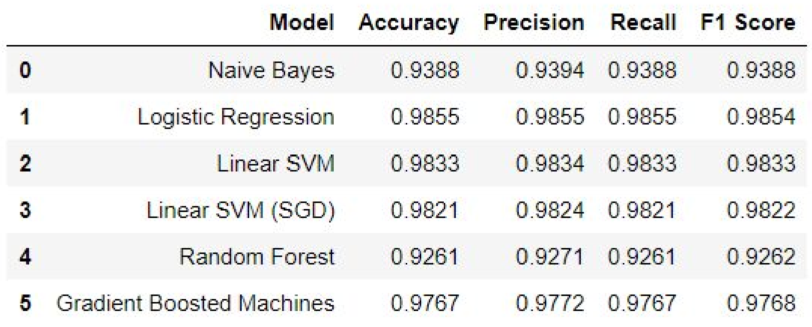
For implementation, we use the *GradientBoostingClassifier* package from sklearn library. And we set n_estimator to 10. For more accuracy, we set the cross validation value to 5.

RESULTS AND DISCUSSIONS

1. Text Classification using TF



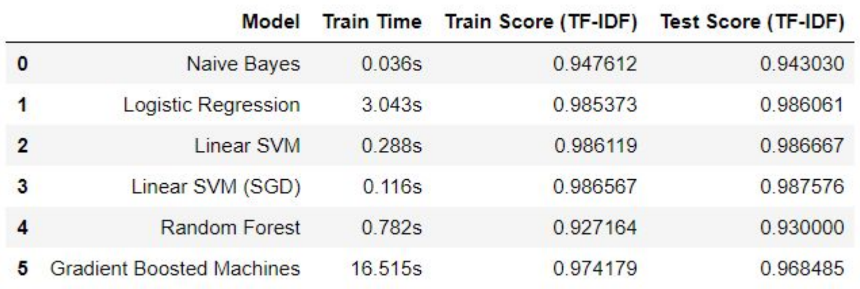
	Model	Train Time	Train Score (TF)	Test Score (TF)
0	Naive Bayes	0.041s	0.939851	0.938788
1	Logistic Regression	4.737s	0.983134	0.985455
2	Linear SVM	0.33s	0.981194	0.983333
3	Linear SVM (SGD)	0.155s	0.980000	0.982121
4	Random Forest	0.841s	0.930896	0.926061
5	Gradient Boosted Machines	8.96s	0.979701	0.976667



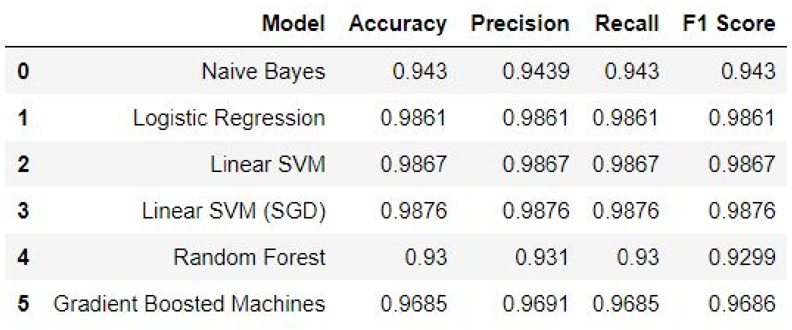
	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.9388	0.9394	0.9388	0.9388
1	Logistic Regression	0.9855	0.9855	0.9855	0.9854
2	Linear SVM	0.9833	0.9834	0.9833	0.9833
3	Linear SVM (SGD)	0.9821	0.9824	0.9821	0.9822
4	Random Forest	0.9261	0.9271	0.9261	0.9262
5	Gradient Boosted Machines	0.9767	0.9772	0.9767	0.9768

Observation :
Based on the training time, Naive Bayes has the least time execution, which is 0.041s. While the Logistic Regression method has the best accuracy score. For the term frequency method, the best model is Linear SVM, because the time execution is less than the mean time and the accuracy is above the mean accuracy.

2. Text Classification using TF-IDF



	Model	Train Time	Train Score (TF-IDF)	Test Score (TF-IDF)
0	Naive Bayes	0.036s	0.947612	0.943030
1	Logistic Regression	3.043s	0.985373	0.986061
2	Linear SVM	0.288s	0.986119	0.986667
3	Linear SVM (SGD)	0.116s	0.986567	0.987576
4	Random Forest	0.782s	0.927164	0.930000
5	Gradient Boosted Machines	16.515s	0.974179	0.968485



	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.943	0.9439	0.943	0.943
1	Logistic Regression	0.9861	0.9861	0.9861	0.9861
2	Linear SVM	0.9867	0.9867	0.9867	0.9867
3	Linear SVM (SGD)	0.9876	0.9876	0.9876	0.9876
4	Random Forest	0.93	0.931	0.93	0.9299
5	Gradient Boosted Machines	0.9685	0.9691	0.9685	0.9686

Observation :

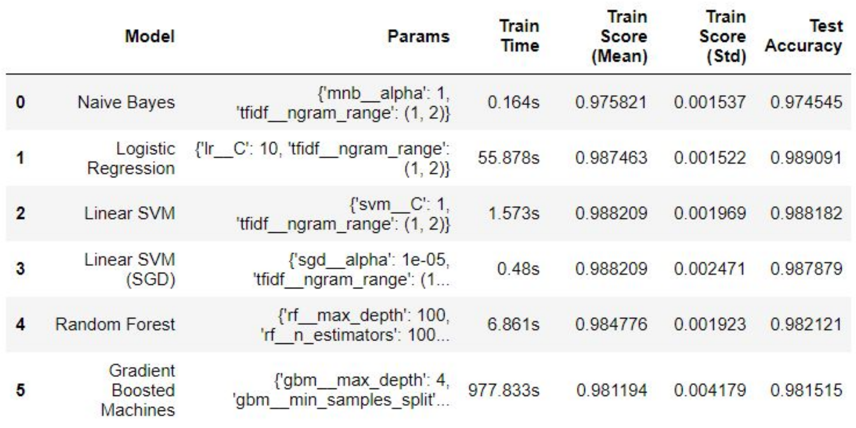
Based on the training time, Naive Bayes has the least time execution, which is 0.036s. While the SGDClassifier method has the best accuracy score.

For the TF-IDF method, the best model is SGDClassifier, because the time execution is less than the mean time and has the best accuracy score.

Compared to the term frequency, TF-IDF has more accuracy than TF one. It happened because TF-IDF also considered the important word and less important word as well.

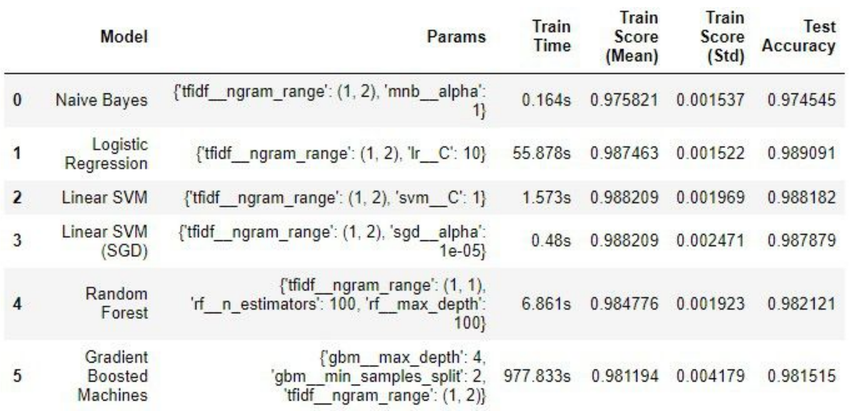
3. Text Classification using TF-IDF with Tuning Parameter

With Grid Search CV



	Model	Params	Train Time	Train Score (Mean)	Train Score (Std)	Test Accuracy
0	Naive Bayes	{'mnb__alpha': 1, 'tfidf__ngram_range': (1, 2)}	0.164s	0.975821	0.001537	0.974545
1	Logistic Regression	{'lr__C': 10, 'tfidf__ngram_range': (1, 2)}	55.878s	0.987463	0.001522	0.989091
2	Linear SVM	{'svm__C': 1, 'tfidf__ngram_range': (1, 2)}	1.573s	0.988209	0.001969	0.988182
3	Linear SVM (SGD)	{'sgd__alpha': 1e-05, 'tfidf__ngram_range': (1...	0.48s	0.988209	0.002471	0.987879
4	Random Forest	{'rf__max_depth': 100, 'rf__n_estimators': 100...	6.861s	0.984776	0.001923	0.982121
5	Gradient Boosted Machines	{'gbm__max_depth': 4, 'gbm__min_samples_split'...	977.833s	0.981194	0.004179	0.981515

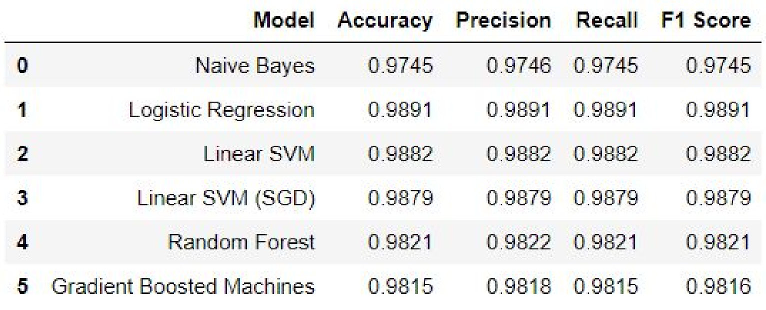
With Random Search CV



	Model	Params	Train Time	Train Score (Mean)	Train Score (Std)	Test Accuracy
0	Naive Bayes	{'tfidf__ngram_range': (1, 2), 'mnb__alpha': 1}	0.164s	0.975821	0.001537	0.974545
1	Logistic Regression	{'tfidf__ngram_range': (1, 2), 'lr__C': 10}	55.878s	0.987463	0.001522	0.989091
2	Linear SVM	{'tfidf__ngram_range': (1, 2), 'svm__C': 1}	1.573s	0.988209	0.001969	0.988182
3	Linear SVM (SGD)	{'tfidf__ngram_range': (1, 2), 'sgd__alpha': 1e-05}	0.48s	0.988209	0.002471	0.987879
4	Random Forest	{'tfidf__ngram_range': (1, 1), 'rf__n_estimators': 100, 'rf__max_depth': 100}	6.861s	0.984776	0.001923	0.982121
5	Gradient Boosted Machines	{'gbm__max_depth': 4, 'gbm__min_samples_split': 2, 'tfidf__ngram_range': (1, 2)}	977.833s	0.981194	0.004179	0.981515

Observation:
Both Grid and Random Search methods give the same optimal parameters, so the training time and score accuracy are the same(by using the same parameters). The difference between two methods is the time that it needs to find the optimal parameters.

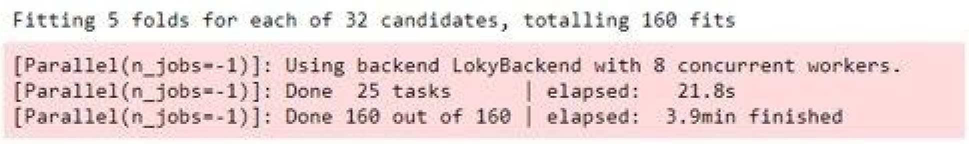
Comparison of performance score



	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.9745	0.9746	0.9745	0.9745
1	Logistic Regression	0.9891	0.9891	0.9891	0.9891
2	Linear SVM	0.9882	0.9882	0.9882	0.9882
3	Linear SVM (SGD)	0.9879	0.9879	0.9879	0.9879
4	Random Forest	0.9821	0.9822	0.9821	0.9821
5	Gradient Boosted Machines	0.9815	0.9818	0.9815	0.9816

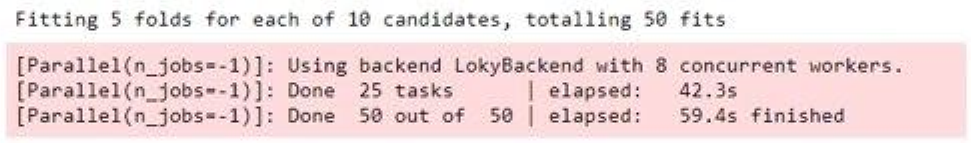
Comparison of time in search for optimal parameters

Random Forest - Using Grid Search CV



```
Fitting 5 folds for each of 32 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   21.8s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:  3.9min finished
```

Random Forest - Using Random Search CV



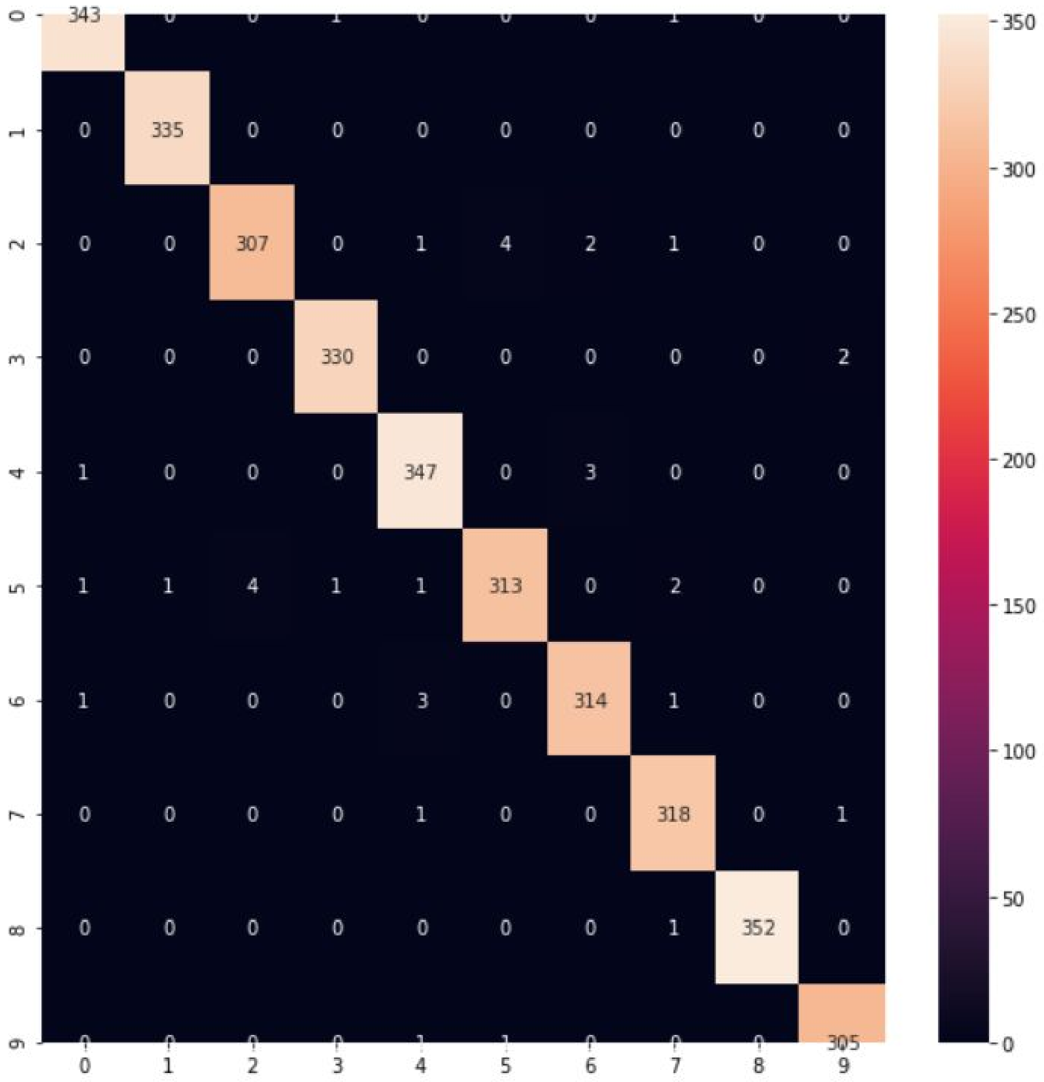
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   42.3s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   59.4s finished
```

Observation:

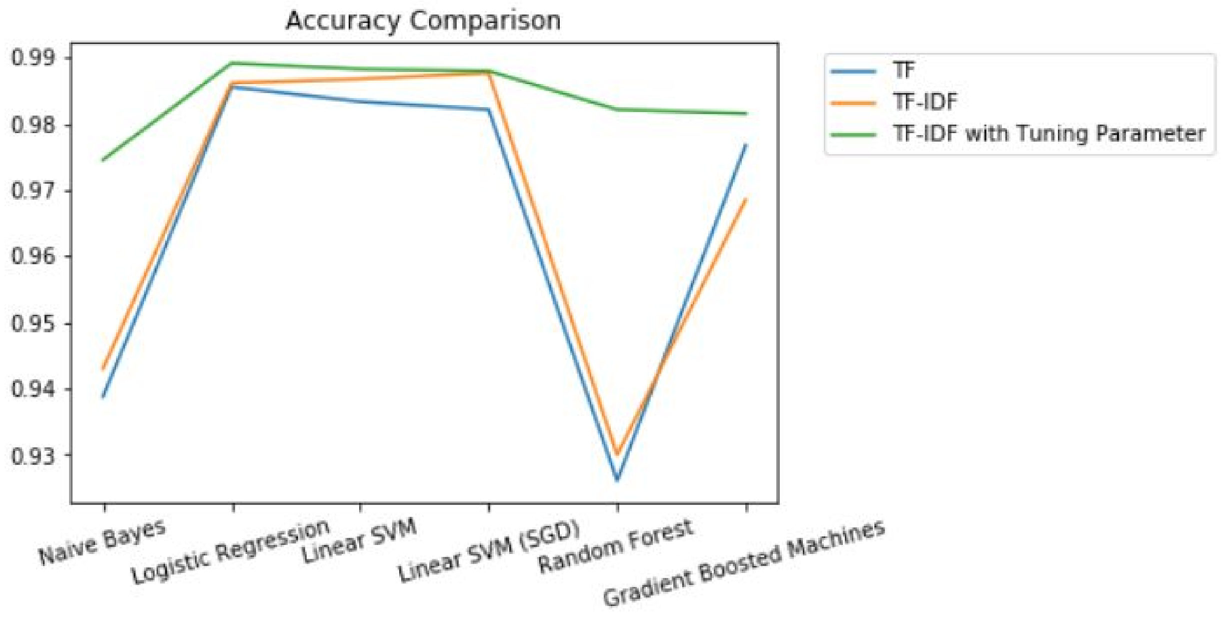
While both methods (Grid and Random Search) give the same optimal parameter, the time that it takes to find that optimal parameter is different. Random Search takes less time than Grid Search. This happened because random search finds optimal parameter by randomly choosing a parameter, while grid search finds optimal parameter by order.

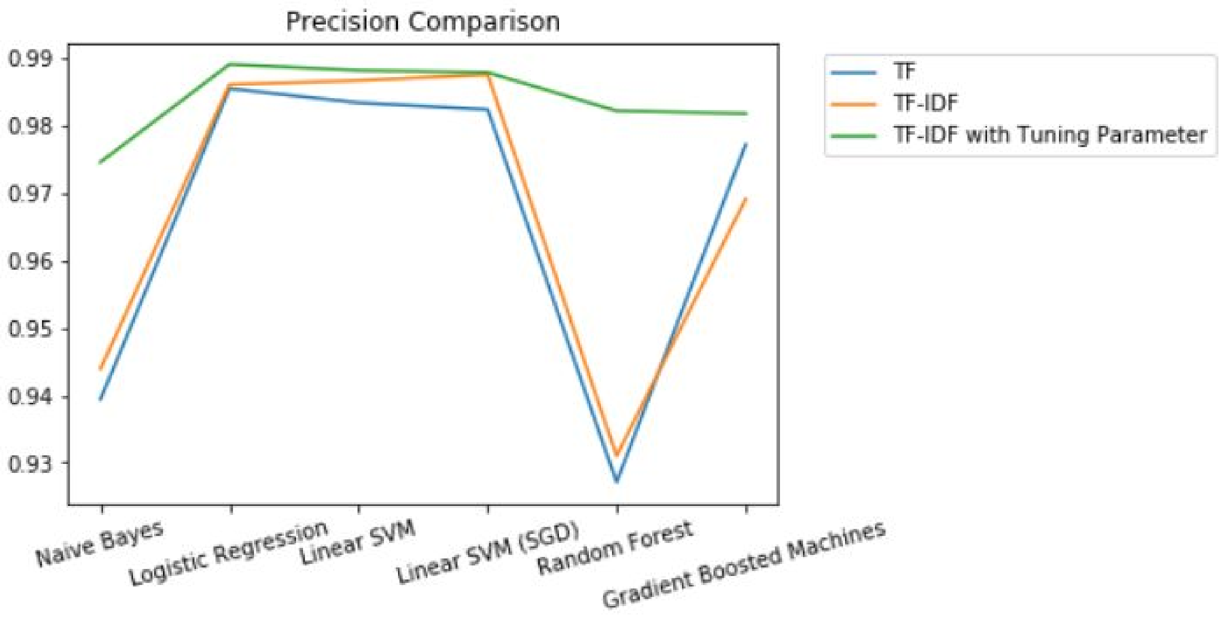
Overall Comparison

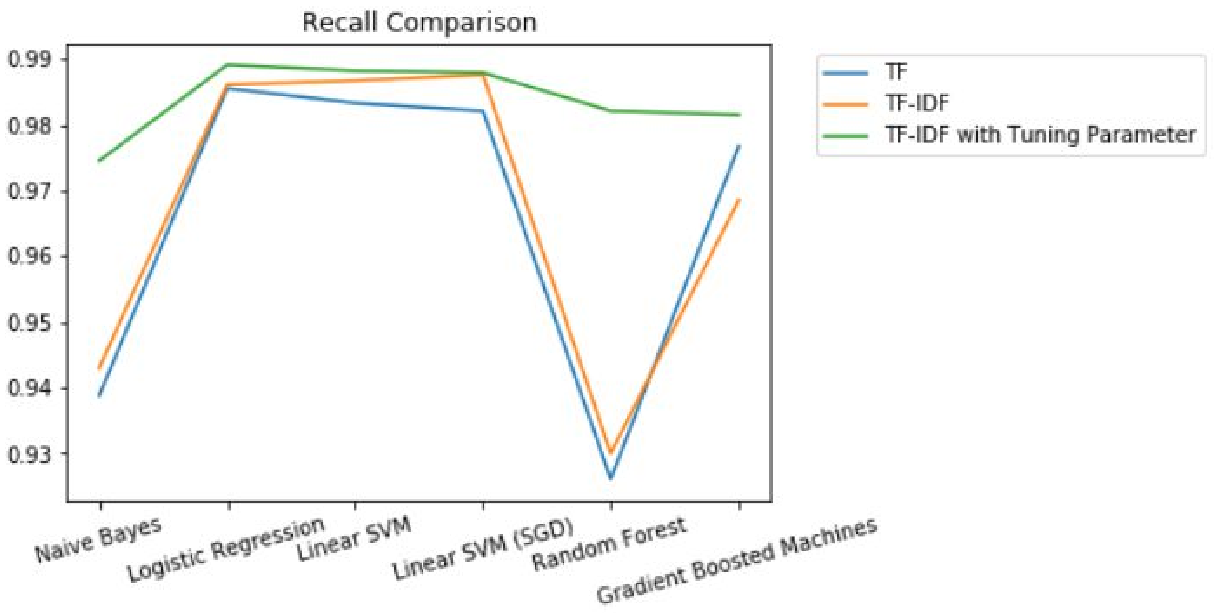
Confusion Matrix

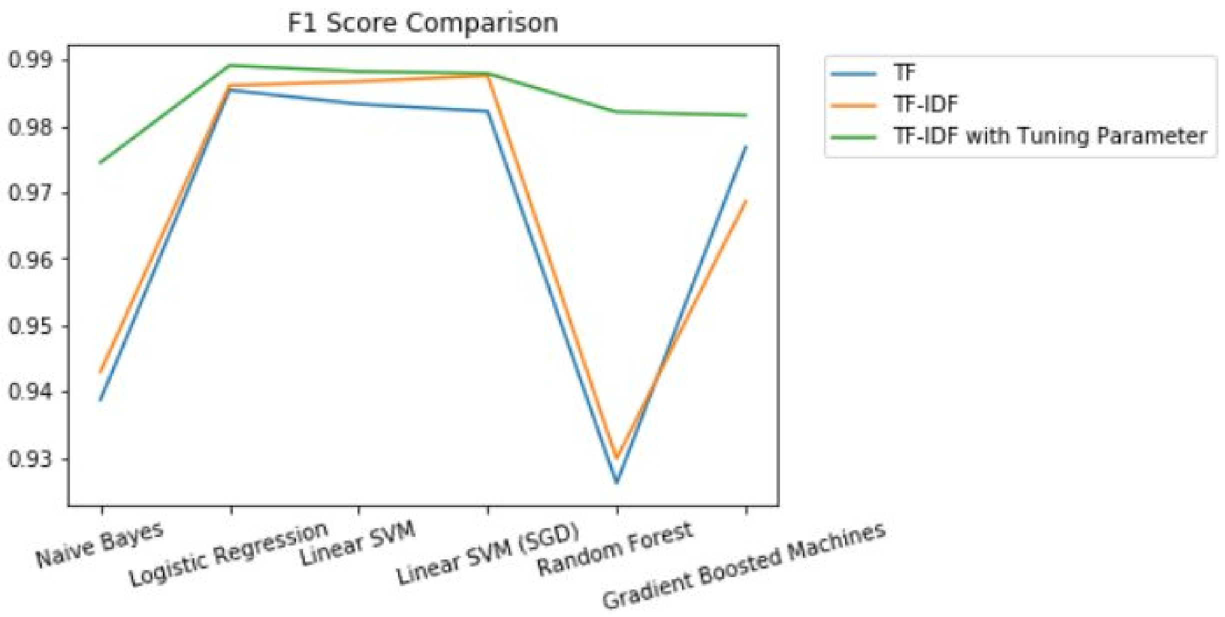


The above image is a confusion matrix based on our model that performs best which is Logistic Regression. We can see that the Linear Regression model with the tuned hyperparameter produces only 4 misclassification data at most. From the confusion matrix above, we can conclude that Linear Regression models can predict almost all the correct labels.


Accuracy Comparison
0.99
0.98
0.97
0.96
0.95
0.94
0.93
TF
TF-IDF
TF-IDF with Tuning Parameter
Naive Bayes
Logistic Regression
Linear SVM
Linear SVM (SGD)
Random Forest
Gradient Boosted Machines


Precision Comparison
0.99
0.98
0.97
0.96
0.95
0.94
0.93
TF
TF-IDF
TF-IDF with Tuning Parameter
Naive Bayes
Logistic Regression
Linear SVM
Linear SVM (SGD)
Random Forest
Gradient Boosted Machines

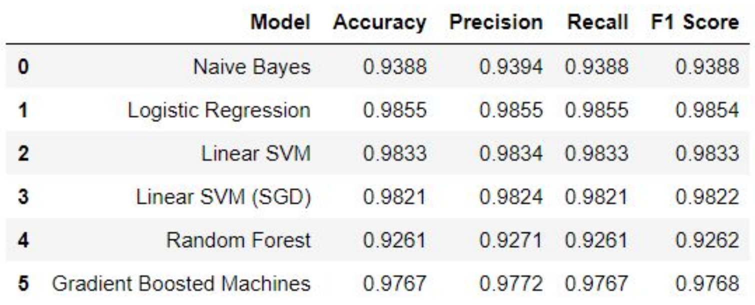




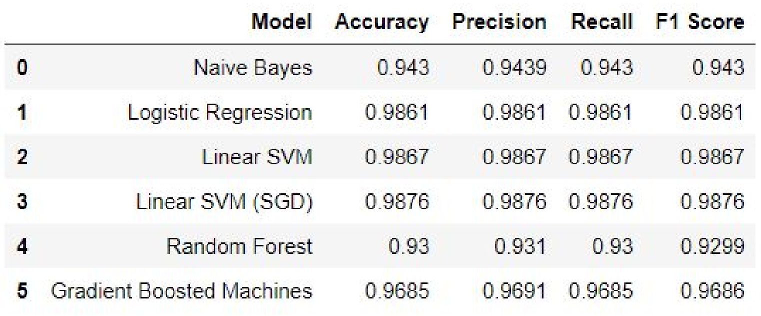
Based on the figures above, we can see that the TF-IDF with Tuning parameter is the best method to use rather than TF or TF-IDF. The green line is always above the orange and blue ones. While the best performance (from the accuracy, precision, recall, and F1 Score) is Logistic Regression using TF-IDF with Tuning Parameter, which has 98.91% score. Followed by Linear SVM, SGD, Random Forest, Gradient Boosted Machines, and Naive Bayes.

CONCLUSION

We can conclude that TF-IDF has better performance than TF has, because the TF-IDF model contains information on the more important words and the less important ones as well, while the Bag of Words just creates a set of vectors containing the count of word occurrences in the document (reviews).

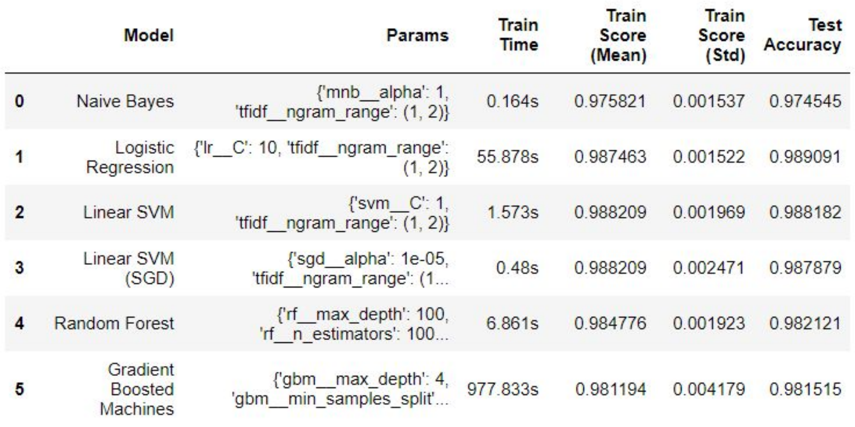


	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.9388	0.9394	0.9388	0.9388
1	Logistic Regression	0.9855	0.9855	0.9855	0.9854
2	Linear SVM	0.9833	0.9834	0.9833	0.9833
3	Linear SVM (SGD)	0.9821	0.9824	0.9821	0.9822
4	Random Forest	0.9261	0.9271	0.9261	0.9262
5	Gradient Boosted Machines	0.9767	0.9772	0.9767	0.9768

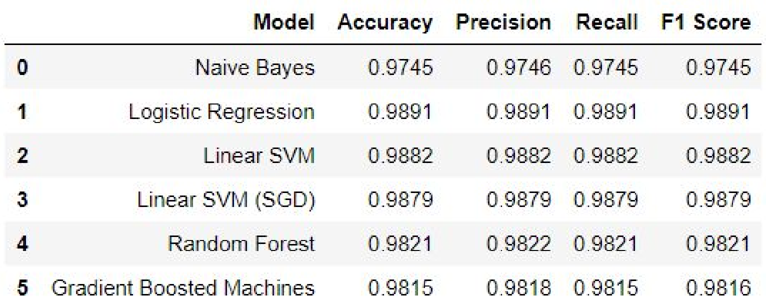


	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.943	0.9439	0.943	0.943
1	Logistic Regression	0.9861	0.9861	0.9861	0.9861
2	Linear SVM	0.9867	0.9867	0.9867	0.9867
3	Linear SVM (SGD)	0.9876	0.9876	0.9876	0.9876
4	Random Forest	0.93	0.931	0.93	0.9299
5	Gradient Boosted Machines	0.9685	0.9691	0.9685	0.9686

In terms of performance comparison. When hyperparameter tuning is involved, the models give a better performance, in terms of accuracy. This happened because the models are using the optimal parameters.



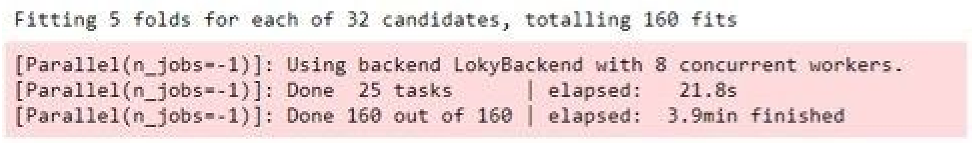
	Model	Params	Train Time	Train Score (Mean)	Train Score (Std)	Test Accuracy
0	Naive Bayes	{'mnb__alpha': 1, 'tfidf__ngram_range': (1, 2)}	0.164s	0.975821	0.001537	0.974545
1	Logistic Regression	{'lr__C': 10, 'tfidf__ngram_range': (1, 2)}	55.878s	0.987463	0.001522	0.989091
2	Linear SVM	{'svm__C': 1, 'tfidf__ngram_range': (1, 2)}	1.573s	0.988209	0.001969	0.988182
3	Linear SVM (SGD)	{'sgd__alpha': 1e-05, 'tfidf__ngram_range': (1...	0.48s	0.988209	0.002471	0.987879
4	Random Forest	{'rf__max_depth': 100, 'rf__n_estimators': 100...	6.861s	0.984776	0.001923	0.982121
5	Gradient Boosted Machines	{'gbm__max_depth': 4, 'gbm__min_samples_split'...	977.833s	0.981194	0.004179	0.981515



	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.9745	0.9746	0.9745	0.9745
1	Logistic Regression	0.9891	0.9891	0.9891	0.9891
2	Linear SVM	0.9882	0.9882	0.9882	0.9882
3	Linear SVM (SGD)	0.9879	0.9879	0.9879	0.9879
4	Random Forest	0.9821	0.9822	0.9821	0.9821
5	Gradient Boosted Machines	0.9815	0.9818	0.9815	0.9816

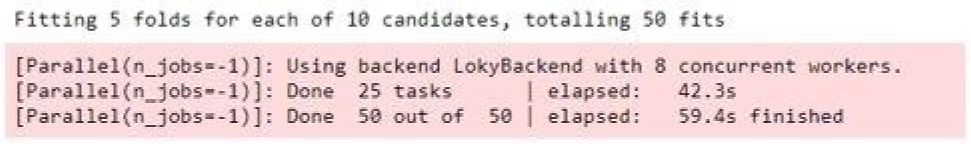
Random search performs more efficiently (in time) while giving the same accuracy as Grid Search. This happened because random search finds optimal parameter by randomly choose a parameter.

Using Grid Search



```
Fitting 5 folds for each of 32 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   21.8s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:  3.9min finished
```

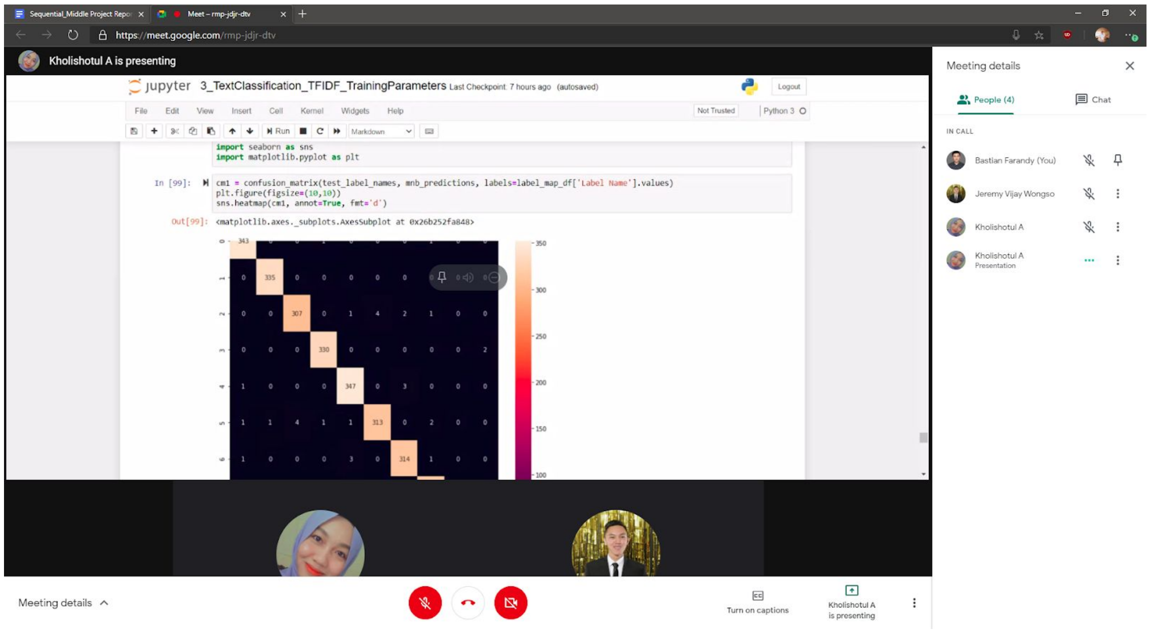
Using Random Search



```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   42.3s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   59.4s finished
```

For further improvement, we can use advanced feature engineering like Word2Vec word embedding and GloVe models for having more accuracy and compare it to the tf-idf models.

Group Meeting


Kholishotul A is presenting
jupyter 3_TextClassification_TFIDF_TrainingParameters Last Checkpoint: 7 hours ago (autosaved)
Meeting details
People (4)
Chat
IN CALL
Bastian Farandy (You)
Jeremy Vijay Wongso
Kholishotul A
Kholishotul A
Presentation
Turn on captions
Kholishotul A is presenting

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [99]: cm1 = confusion_matrix(test_label_names, mnb_predictions, labels=label_map_df['Label Name'].values)
         plt.figure(figsize=(10,10))
         sns.heatmap(cm1, annot=True, fmt='d')
Out[99]: <matplotlib.axes._subplots.AxesSubplot at 0x26b252fa848>
```

Personal Comment

Jeremy Vijay Wongso :

From this project I have learned to solve one of the classic problems in NLP, which is text classification. I also have learned about important concepts like bag of words, TF-IDF to extract features from the document and learn how to use the classification models using the extracted features. From my point of view, from the dataset that we used in this project, all of those classifier algorithms were almost equally matched when optimized, and sometimes, if we have enough dataset, choice of algorithm can make hardly any difference.

It is very important for us to find the best hyperparameter for the classifier models so that the models can perform classification tasks at their finest. We can see that after the hyperparameter is tuned, the models perform slightly better than without the tuned hyperparameter.

Kholishotul Amaliah :

In my opinion, it is such a great thing that we can classify text using NLP. Although the accuracy is not perfectly 100%, the mean score is 97%. This kind of classification can be implemented in many fields, not only medical, but also education, reviews, management, filtering opinion, and anything that uses text for information. I am sure that this can be useful for our daily life, especially for information retrieval.

Based on the observation above, I think the best model is SGDClassifier using TF-IDF with Tuning Parameter. Because the accuracy is above the mean accuracy while the time is less than 1 s, which is short time execution.

Bastian Farandy :

From my personal opinion, I think that TF-IDF with tuned parameters give the best performance and efficiency compared to TF only. The downside for using tuned parameters is the long waiting time for the Grid or Random Search method to find the optimal parameters. For the model, I prefer the Linear SVM (SGD) models because while the accuracy is slightly lower than Linear SVM and Linear Regression models, the training time is significantly lower compared to the two other models. The accuracy of the Linear SVM (SGD) model is just slightly worse than Linear SVM and Linear Regression models.

LINK :

The full code of program can be found in
https://github.com/kholishotula/TextMining_MiddleProject
The Youtube link can be found in
https://youtu.be/QqfDBGUc9E8

REFERENCES

Sarkar, D. (2019). *Text Analytics with Python : A Practitioner's Guide to Natural Language.* Bangalore: Apress.

sklearn.linear_model.SGDClassifier. (2020, November 16). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier

sklearn.model_selection.GridSearchCV. (2020, November 17). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV

sklearn.model_selection.RandomizedSearchCV. (2020, November 17). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV