**Introduction**

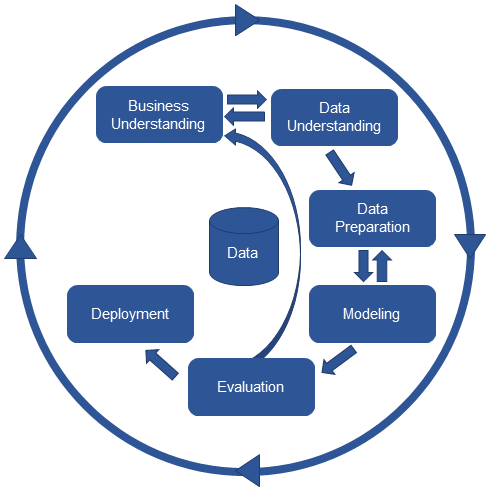
As textual data becomes ubiquitous in most industries today, it is critical to classify texts appropriately. On the report, I explore two algorithms to improving the classification accuracy of text using supervised learning for predefined classes. The aim of the report is to document the practical implementation of Machine Learning methods and Natural Language Processing used in building a text classification model to classify the class of a text. The final model was trained to classify texts using predefined classes sports\_&\_gaming, pop\_culture, daily\_life, business\_&\_entrepreneurs, science\_&\_technology and arts\_&\_culture, an essential task that has several real-world applications in diverse industries. The project examines various learning outcomes using Data Mining and Data Analytics tools such as Scikit-learn and NLTK.

The project follows a CRISP-DM methodology that includes business understanding, data understanding, data preparation, modeling, evaluation and deployment. The dataset that used was a JSON file format having the column of interest labelled text. The Preprocessing steps included removing mentions and URLs, removing unnecessary punctuations, tokenization and fixing contractions. Stopwords were also removed before using Multinomial Naïve Bayes and the Linear Support Vector Classifier models. The hyperparameters were tuned using GridSearchCV and RandomizedSearchCV for the models optimization.

In addition, two types of models were used in the project, Multinomial Naïve Bayes and the Linear Support Vector Classifier and the performance of each model was evaluated using several metrics such as overall accuracy, precision score, recall score and f1 score. To obtain the optimum accuracy a voting classifier was used to pool the best predictions for better classification accuracy (Wijaya, 2021).

**Methodology**

Using the provided dataset, the text classification follows a systematic approach presented on the report methodology. After importing the required libraries as a standard procedure, I first loaded the provided dataset using a pandas function pd.read\_json() since it was a JSON file. I followed by checking if there existed any missing values in the dataset and confirmed that there was none. Next, I printed the unique values to identify the classes in the dataset to further explore the data then using the function value\_counts() that provided more information to the choice of the model to use, hence following the CRISP-DM methodology as displayed below.



The subsequent step was to preprocess the textual data by cleaning, tokenizing and lemmatizing. On the cleaning function def clean\_text(text) I first removed URLs, mentions, unnecessary punctuations on the text then converted it to lowercase. Furthermore, I fixed contractions, tokenized the text finally removing the stopwords using nltk.corpus.stopwords and lemmatize the words using nltk.stem.WordNetLemmatizer. All this was applied using the method .apply() to carry out the operation on the text column of the dataframe.

The afterward step was to divide the data to training and test sets with a test size of 20% and a random state of 42 while calculating class weight as 1/count for each class in the training set because the classes were imbalanced in the dataset. I then chose and tested a Multinomial Naïve Bayes model where I created a pipeline of a vectorizer, transformer, and classifier. The vectorizer and transformer were CountVectorizer() and TfidfTransformer() respectively, while the classifier was MultinomialNB(). After evaluating the model on the test data, the class of arts & culture was not sufficiently predicted hence I decided to check the best combination of hyperparameters to tune for the pipeline.

I then chose to use two models Multinomial Naive Bayes and Linear Support Vector Classifier using the scikit-learn library. For Multinomial Naive Bayes I defined the hyperparameters to tune the model including ngram range, use\_idf, alpha, fit\_prior, class\_prior, and force\_alpha followed by grid search of 5-fold cross-validation to find the best combination of hyperparameters. I then created a pipeline by chaining TfidfVectorizer and LinearSVC functions for the Linear SVM classifier, also defining the hyperparameters to tune that included C, max\_features, and stop\_words followed by a random search of 5-fold cross validation on the training set to find the best combination of hyperparameters. On both occasions, the best model was evaluated on the test data where the classification report and the best hyperparameters for the model were printed.

Then, I created an ensemble of the four classifiers using scikit-learn's VotingClassifier that included two Multinomial Naive Bayes models and two Linear Support Vector Classification model, both included the first defined model of is type and the tuned hyperparameters model. For the Multinomial Naive Bayes the second pipeline of its type included a different alpha of 1 whereas the first included the best hyperparameters determined earlier. The Linear SVC pipelines only had different hyperparameters. I used the voting classifier to combine the best predictions of the four pipelines where it was fitted on the voting\_clf object. Lastly, I also evaluated the ensemble model to come up with the final prediction results.

Finally, after creating the ensemble model, I saved the model to a file using the function joblib.dump() to enable deployment, that followed next by first loading the saved model from the file and then collected a sample Twitter data. I had to preprocess the sample Twitter data since it was not of the same format as the training dataset. I used the same def clean\_text: function to clean the collected Twitter sample data, then passed it to the ensemble model to predict the class of the tweet. Afterwards, using pandas I created a dataframe to hold the results that included the predicted class of the tweet. Then I printed and visualized the results to present the predicted class.

To evaluate further the performance of the final model using the scikit-learn library, I plotted a confusion matrix using the predicted and actual classes that helped in identifying the number of correct and incorrect predictions made by the model. I also visualized the classification report using scikit-learn library that provided additional evaluation metrics such as precision score, recall score and f1 score for each class, as well as the overall performance of the model.

**Results**

Figure 1:

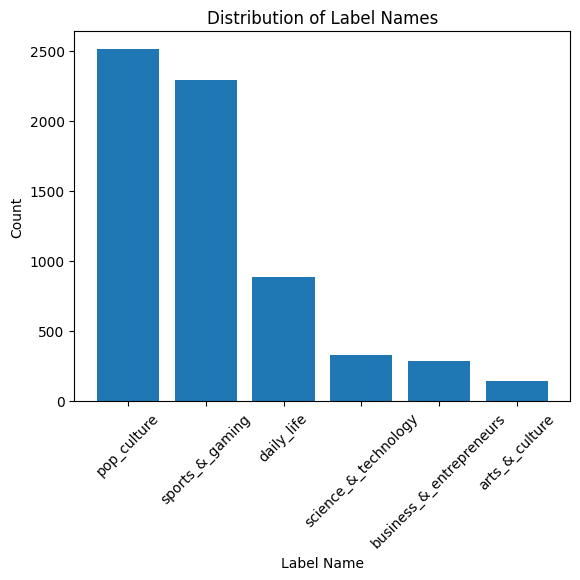
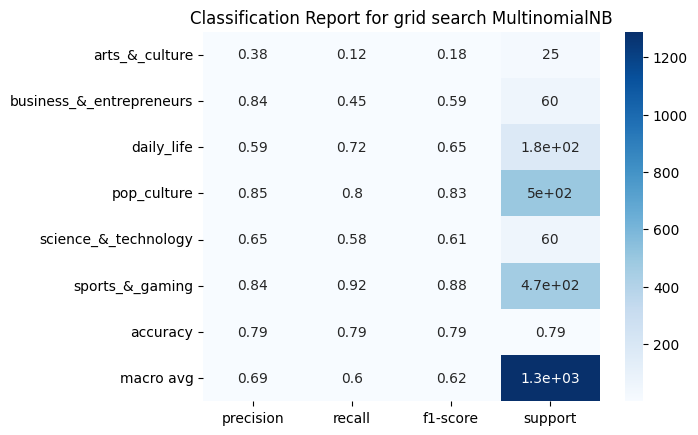
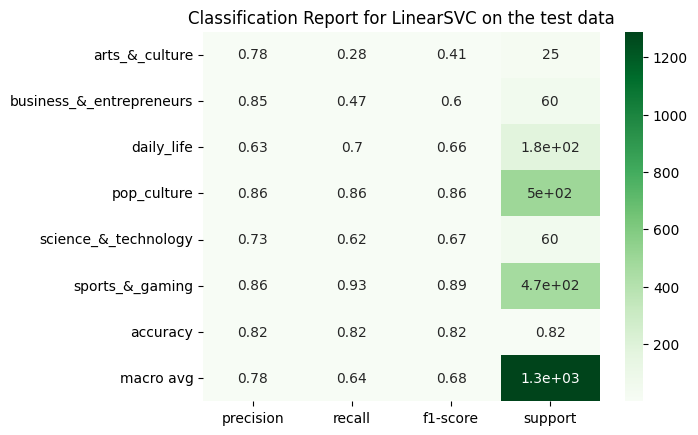


Figure 2:



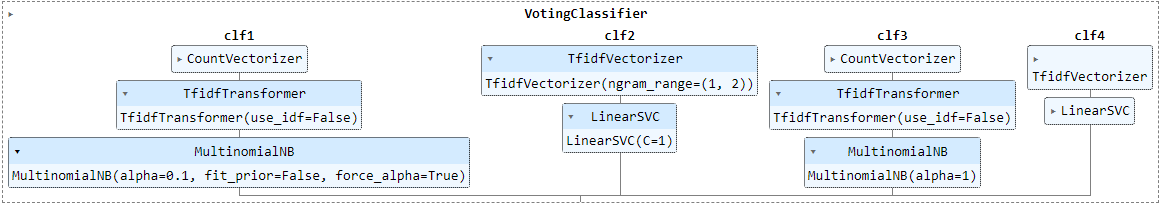
The classification report above was used to evaluate the performance of the Multinomial Naive Bayes model on the test data.

Figure 3:



The classification report above was used to evaluate the performance of the LinearSVC model on the test data.

Figure 4:



The figure above shows the final ensemble model voting classifier.

Figure 5:

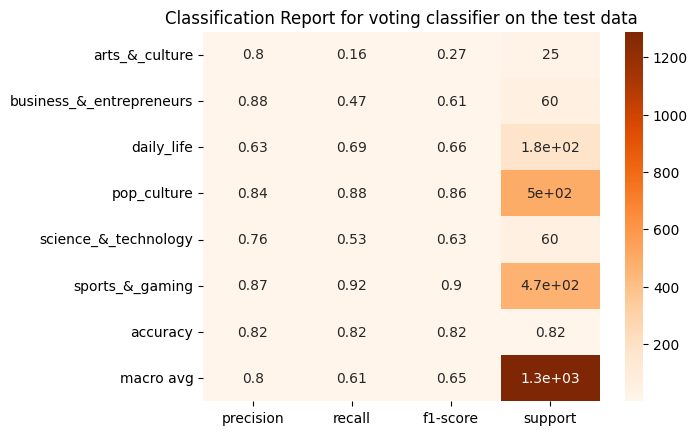
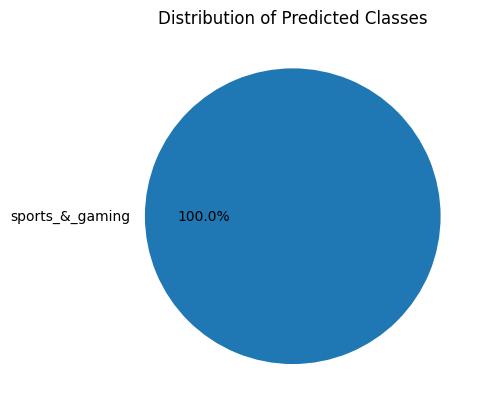


Figure 6:



Figure 7:



**Evaluation & Discussion**

Figure 1, displays the distribution of classes in the dataset, where the x-axis represents the different classes while the y-axis shows the number of instances in each class. From the bar plot, it is clear that the majority of instances are from pop\_culture and sports\_&\_gaming classes whereas, business\_&\_entrepreneurs with 287 instances, and arts\_&\_culture with 144 instances have the least of instances. Based on this results it is evident the dataset is not balanced and may negatively impact the performance of the model, where a model would have performed better on the classes with most examples compared to those with least examples.

The effect is evident considering the Classification Report heatmap for grid search MultinomialNB from Figure 2. The classification report was based on evaluating the test data after tuning the hyperparameters using the grid search where the precision score shows the proportion of the predicted labels that were correctly predicted for each class. The class with the highest precision score was pop\_culture with a score of 0.85 indicating that 85% of the predicted texts for pop\_culture class were correct. The sports\_&gaming class also had a high precision score of 0.84 hence 84% of the predicted sports\_&\_gaming class were correct. That was when arts\_&\_culture class had a precision score of 0.38 meaning only 38% of the predicted texts were correct at the time.

The recall score shows the fraction of true instances that were correctly identified for each class. Again, arts\_&\_culture class had the lowest score of 0.12 meaning that only 12% of the true instances of arts\_&\_culture were correctly identified by the model, compared to the highest sports\_&\_gaming class that had a score of 0.92 meaning that 92% of the true instances of sports\_&gaming were correctly identified by the model, showing that when a text was indeed about sports the model predicted correctly.

The f1 score combines both precision score and recall score to provide an overall evaluation of the model's performance for each class. The pop\_culture and sports\_&\_gaming classes having the highest score, while the arts\_&\_culture class had the lowest f1-score signifying that the model struggled to identify instances in this class, possible due to the limited number of instances available in the dataset. To solve the issue I calculated the weights of the classes in order to balance the dataset. I then defined the hyperparameters to tune followed by a grid search to obtain the best combination of hyperparameters for MultinomialNB.

Furthermore, I chose to examine a LinearSVC model after defining the parameters to tune and performing a randomized search to find the paramount hyperparameters, Figure 3 shows the classification report of the resulting model. As shown in figure 3, compared to the previous MultinomialNB model, the precision score, recall score and f1 score have improved significantly in most of the classified classes especially on the arts\_&\_culture class. That is attributed to the fact that LinearSVC is a more complex model that can capture the relationships between the features and the labels better than the MultinomialNB model.

Nevertheless, the recall score for science\_&\_technology decreased compared to the previous model, suggesting that the model may have had difficulty correctly identifying text related to science and technology. However, that was an acceptable tradeoff considering the accuracy of the LinearSVC improved to 0.82 hence achieving better performance than the previous model that had a weighted average of 0.79. Therefore, the LinearSVC model was a better candidate for predicting the classes of the collected tweets.

I then resulted to combine the different pipeline models in order to achieve better accuracy as shown in Figure 4. The ensemble model consisted of four individual models from the previous tests. The results show improvement in most of the classes compared to the previous two models, particularly for the classes with low precision and recall scores. Overall, the LinearSVC model outperformed the MultinomialNB model where the ensemble model leveraged the strengths of both models to produce the better results.

Conversely, the f1 score of the arts\_&\_culture class was still the lowest compared to sports\_&\_gamming that was still the highest demonstrating that there might still be some imbalance on the dataset that might be solved by further iteration. Yet, considering there was not much difference in overall accuracy, I decided the results to be used as the finest realizable estimates on the model.

Figure 6 shows the heatmap of the resultant confusion matrix of the final ensemble model. The confusion matrix highlights the number of predictions for each class with the actual and predicted labels of the test data having the brighter color signifying the more correct predictions. Figure 6 shows that the ensemble model performed well by predicting correctly most of the text for the arts\_&\_culture class as it had a high number of true positives and fewer false negatives.

Whereas, the sports\_&\_gaming class had the highest number of false positives and false negatives despite having a high f1 score showing it was more difficult for the model to predict correctly. From the results, it shows even though arts\_&\_culture has a low f1 score compared to sports\_&\_gaming most of the text for art\_&\_culture were predicted correctly (T, 2019). Nevertheless, though the confusion matrix is a useful tool for understanding the strengths and weaknesses of a model, it only provides a summary of the performance of the model across all the classes and does not provide details on the reasons for misclassification.

Finally, I inspected the distribution of predicted classes to determine the precision of the model in predicting classes for the new textual data, figure 7 shows distribution of predicted classes. The model predicts the sample text to its correct class with a 100% accuracy for the specific text after deployment. That indicated the model had been productively trained and is able to accurately classify new textual data or tweets. Nonetheless, the model was only tested on a single sample text further arrays of textual data should be used after a more robust deployment technique, which I did not use at this particular project.

**Future Works**

There are several areas for improvement and further exploration that I observed, One of the major issue was that for class imbalance in the dataset that affected even the final model’s performance. Considering that, the use of oversampling or undersampling or both techniques are possible solutions to the situation in order to balance the number of instances in each class. If that could have been explored and emerged to be the solution, it would have improved performance of the model significantly on the underrepresented classes.

Another area of exploration would be that of feature extraction techniques such as contextual embedding could have been explored as I only used bag-of-words representation with n-gram features (Mars, 2022). That is because contextual embedding methods like GloVe, Word2Vec and BERT are supposed to be more effective in capturing the semantic relationships between words and may lead to better performance in text classification tasks.

Moreover, the ensemble method used on the project verified to be useful in improving the overall performance of the model than any model alone. Due to that fact, other ensemble methods available could have been experimented on so as to further improve the model's performance. Stacking is one of the ensemble techniques that were not explored, that involves training many different models then using their output as the inputs of the final model. Furthermore, blending ensemble technique was not used either. It is similar to stacking but uses simple linear combinations of the outputs of the individual models instead of training a final model. To end, boosting ensemble method was also not explored; the ensemble method involves iteratively training multiple weak models and adjusting the weights of the training examples based on the performance of the previous models.

Lastly, deployment being the final step of the project. A more robust deployment technique could have been used to make the model more easily accessible and usable by others. Provided more time development of a web framework such as Flask or Django would have been possible to communicate with the trained model. Another deployment method would have been the use of containerization platforms like Docker, hence creating a Docker image of the trained model with its dependencies enabling the model to be deployed across many different platforms. Additionally, it would be easily deployed and run on the different platforms without having to worry about compatibility issues, as it would be in an isolated environment. In conclusion, there might exist other potential areas for future work on the project.