**IFT6390B Kaggle competition**

**Text classification**

IFT6390B – Fundamentals of Machine Learning

Presented to Dr. Dhanya Sridhar

Presented by Cheng Dong Zhang

Université de Montréal

November 13th, 2024

**Table of Contents**

[Introduction 1](#_Toc182352056)

[I. Machine Learning Pipeline 1](#_Toc182352057)

[Data Loading 1](#_Toc182352058)

[Data Exploration and Visualization 1](#_Toc182352059)

[Data Preprocessing 2](#_Toc182352060)

[Class Balancing 2](#_Toc182352061)

[Feature Engineering 2](#_Toc182352062)

[Model Selection and Training 3](#_Toc182352063)

[Numpy Implementation 3](#_Toc182352064)

[Package Implementation 3](#_Toc182352065)

[Results 5](#_Toc182352066)

[Best Model on Validation sets: 5](#_Toc182352067)

[Best Model on Test set: 5](#_Toc182352068)

[Potential reasons in Overfitting and Potential improvements 5](#_Toc182352069)

[Appendix 6](#_Toc182352070)

# Introduction

Text classification is a powerful tool in natural language process that allows users to categorize large volume of data automatically and efficiently. The capabilities of a machine learning model to acurrately classify text data can drive decision-making, improve operational efficiency amongst other benefits for a variety of different fields and industries. For example, an accurate text classification can be used to sort through legal documents of a law firm, which they receive in high volume, which enhances efficiency and time usage of all employees of the firm to further help their clients.

In this paper, I propose a comprehensive approach to developing a text classification algorithm. I start by exploring the nature and structure of the textual data, examining and analyzing patterns and importance of all features. I then clean the data, filter the data and use it to test models to fit for the data using Python and Numpy only, then using any packages available.

# I. Machine Learning Pipeline

## Data Loading

The data was loaded from NumPy and CSV files, with:

* X\_train: training data with shape (num\_documents, num\_terms), loaded from a .npy file.
* y\_train: training labels from a CSV file, focusing on the second column for class labels.
* X\_test: test data loaded as a NumPy array.
* vocab\_data: vocabulary data to map term indices to words.

## Data Exploration and Visualization

To better understand the data, we have done a few visualizations:

* **Class Distribution:**
  + A bar chart showed the imbalance between class 0 and class 1.
  + This allows us to verify how large the class imbalance is and separate the data for further explorations. (See Appendix)
* **Feature Appearance:**
  + Visualizations plotted the average frequency of words for each class.
  + These visualizations allowed us to see if there are outliers in the features that were abnormal and identify them. (See Appendix)
* **Correlation Analysis:**
  + A bar chart visualizing the correlation between the feature X and the result Y. (See Appendix)
  + This allows us to further visualize how each feature individually impacts the result of the predicted class. Using this analysis, we saw that almost all features possess little to no correlations to the predicted class individually.
* **Variance Analysis:**
  + A bar chart visualizing the Variance between the feature X and the result Y
    - By plotting the Variances, we saw that the features had greatly varying variance values and some extreme outliers in variances. In order to tame the variances, we will try to log transform the data.
  + A bar chart visualizing the log transformed variance of the features found in the training data. (See Appendix)

## Data Preprocessing

**Data was processed through several key steps:**

* **Stop Word Removal:**
  + Common English stop words (e.g., "the," "and" "or") are removed from the dataset because they typically carry little to no informative value for distinguishing between classes. Retaining these words can introduce noise, slow down training, and reduce model performance. By removing stop words, we focus the model on more meaningful words that may better represent class-specific information. (See Appendix)

## Class Balancing

**Due to class imbalance, Synthetic Minority Over-sampling Technique (SMOTE) was applied:**

* **Before SMOTE:** 
  + Class imbalance was notable with a majority class 0 was far outnumbering the minority class 1. According to the data visualization, around 75% of the data was of class 0 and remaining 25% was of class 1, making class 0 three times more prevalent in the training data set than that of class 1. This imbalance can result in a biased model performance.
* **After SMOTE:** 
  + The class distribution was balanced, resulting in equal representation of both classes. (See Appendix)

## Feature Engineering

**With class-balancing complete, the following adjustments were made:**

* **Log Transformation:** 
  + Applied to non-zero mean columns, normalizing data skewness and the variance.
* **High Variance Feature Selection:** 
  + Reduced dimensionality by dropping low-variance features that contribute minimally to the learning process
* **Final Feature Transformation (Milestone 2 only):** 
  + Selected features were further processed with the Variance Threshold and TF-IDF transformations, followed by clipping and Chi-square selection.
    - These processes were also methods tested to clear outliers and verify the features association with the target variable
* **Anova Feature Selection (Milestone 2 only):**
  + Selects the features that differentiates best between the target classes using an Analysis of Variance F-test feature selection. This technique check for features with the most significant mean differences across classes.
* **XGboost Freature Importance Filtering (Milestone 2 only):**
  + Uses XGboost to identify features that have high importance and reflects feature interactions
* **Mean Difference Feature Filtering:**
  + This is a simple measure of difference in mean values between each target classes and identifying features with the largest disparities

## Model Selection and Training

### Numpy Implementation

Utilized an implementation of the Naïve Bayes Classifier with a Gaussian Maximum Likelihood Estimation used in Lab 2 of the course. This methods assumes the features follows an isotropic Gaussian distribution.

GaussianMaxLikelihood:

* The train method computes the mean and variance of each feature class
* The loglikelihood function calculates the log-likelihood of the test data under the gaussian distribution of the feature class learned from the previous step

Naïve Bayes Classifier

* Combines the Gaussian MLE model to incorporate feature interactions

This model was heavily tested with a variety of feature numbers to see optimal top features to optimize learning process. By selecting all features containing the largest difference in mean values, the models learning process seems to peak around 860 top features and increase in F1 score of the model starts stagnating. (See Appendix)

### Package Implementation

Multiple models were evaluated, each trained on the processed, resampled data:

* Logistic Regression
* Naive Bayes
* Random Forest

Each model was evaluated using 5-fold cross-validation with F1 scores calculated per model. The best-performing model, determined by the highest mean F1 score, was trained on the entire resampled dataset. (See Appendix)

Separately, the XGBoost model was also tested using a hyper parameter grid and cross validated the F1 score to determine whether the model was more performant than the other simpler models

The grid included:

* Number of boosting rounds
* Learning Rate
* Maximum depth of tree
* Minimum sum of instances needed in a child
* Subsample fraction of data to build each tree
* Fraction of features to be chosen to build each tree
* Minimum of loss reduction
* Balancing of positive and negative weights to handle class imbalances

After fitting a randomized search to find the best parameters yielding the highest f1 score, we find that

* scale\_pos\_weight=1
* subsample=0.8
* n\_estimators=50
* min\_child\_weight=1
* max\_depth=7
* learning\_rate=0.1
* gamma=0.5
* colsample\_bytree=0.6,

After performing the cross validated F1 test, we see that

Concluding that the logistic regression yields the highest F1 score and we shall move forward utilizing the logistic regression for Milestone 2.

# Results

## Best Model on Validation sets:

* Logistic Regression achieved the highest cross-validation mean F1 score amongst its peers when testing using the validation sets by splitting the training data, but did not yield the highest F1 score in the test set amongst all models. This models success indicates that logistic regression was effective at generalizing the high-dimensionality of the dataset with such sparse features by avoiding the overfitting that could happen easily using stronger and more powerful models.

## Best Model on Test set:

* Numpy implemented Naïve bayes with Gaussian MLE scored the highest with the test set once uploaded onto Kaggle, with a F1 score of 69%, compared to the 68% from the milestone 2 model. While the models from Milestone 2 much outperformed the naïve bayes numpy implemented method in the train/validation cross validated F1 score, we can see that it was overtrained and was not able to predict accurately the document classification.

The discrepancy between the validation sets and the test sets suggest that the dataset had potential in overfitting to the training dataset. The higher the complexity of the models, combined with extensive feature selection and hyperparameter tuning through randomized search grids lead to models that capture lots of the noise in the training data instead of being able to learn generalized patterns. These models show strong results during cross validations but disappointing results when applied onto the test sets. In contrast, the Naïve Bayes with Gaussian MLE approach, despite lower performances during the training validation splits, generalized much better and did not fall into the common overlearning issue seen in other models.

## Potential reasons in Overfitting and Potential improvements

Extensive Feature Selection:

We selected features purely based on the validation sets F1 score, choosing the parameter which seem to maximize its values, which may have caused overtraining. Instead, we should’ve plotted the F1 score and chosen the parameter value when F1 scores starts to flatten like we did for the parameter selection for the naïve bayes.

# Appendix

Class Distribution:

A graph with blue and orange squares

Description automatically generated

Feature Appearance:

A graph of a number of values

Description automatically generated with medium confidence

A graph of words with text

Description automatically generated with medium confidence

Correlation Analysis:

A graph with blue bars

Description automatically generated

A comparison of a graph

Description automatically generated with medium confidence

Variance Analysis:

A graph showing a number of columns

Description automatically generated

Stop Word Removal:

['a', 'an', 'the', 'and', 'or', 'but', 'if', 'while', 'with', 'without', 'of', 'at', 'by', 'for', 'to', 'in', 'on', 'from', 'up', 'down', 'out', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now’, ‘we', 'our', 'theoretical', 'of', 'the', 'and', 'in', 'this', 'were', 'is', 'was', 'some', 'are', 'been', 'will']

Smote Resampling

A graph with blue and orange squares

Description automatically generated

Naïve Bayes feature selection Process

A graph with blue and orange lines

Description automatically generated