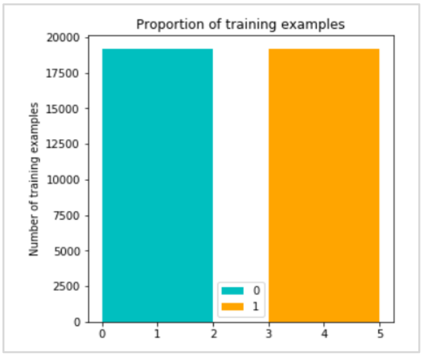
Problem Statement

The data is derived from Stack Overflow comments where the target variable ("Outcome") suggests whether a comment received a high number of upvotes or not. Therefore, we use binary text classification analysis to predict if a given comment would receive a higher number of upvotes or fewer or no upvotes.

Exploratory Data Analysis

The training dataset consists of 44,183 samples and the test dataset consists of 28,200 samples. There are no null values in either dataset. The target variable in the training dataset is a binary variable where 1 indicates a high number of upvotes and 0 indicates very few or zero upvotes for the corresponding comment. The distribution of the target variable is relatively balanced (refer figure alongside) with 57% labels classified as 1 and 43% as 0.



Data Preprocessing & Feature Engineering

The initial data preprocessing was based on the standard practice for text analytics which involved the following steps to prepare the training samples for building the model:

- Changing to lowercase
- Removing stop words
- Removing punctuations
- Removing frequent and rare words
- Removing URLs
- Stemming
- Lemmatization

However, the performance of the models was significantly affected using these methods even with careful selection and different combinations. This reiterates the importance of selecting preprocessing parameters that are appropriate for the dataset under consideration. To overcome this, the subsequent data preprocessing was based on the insights derived from EDA plots (refer Appendix 1).

An important step, previously disregarded, was the importance of punctuations in the dataset. Since the data is derived from Stack Overflow comments, mathematical symbols and punctuations are inherent to the data as many comments include solutions to the queries asked. Moreover, comments with solutions would be more useful to the user and therefore it is important to retain them during the data preprocessing step. To achieve this, a custom token pattern was created to include such punctuations and mathematical symbols based on the EDA plots. The only other step included in the revised data preprocessing was changing the comments to lowercase before feature engineering.

For feature engineering, following Sklearn tokenizers were employed to the processed data at character, word and n-gram levels:

1. CountVectorizer
2. TfidfVectorizer

Out of the various combinations, the models performed best when combined with CountVectorizer at word-level and therefore, it was used for the modeling process.

About CountVectorizer:
The following parameters were tuned based on pre-processing results for feature engineering to improve the accuracy of the models:

- *token_pattern:* a customized token pattern was used to improve the accuracy significantly
- *max_df*, *min_df:* no limits set to ensure the importance of frequent and rare words were retained. Moreover, the model performance was affected when limits were set
- *stop_words:* no stop words were removed based on previous criteria
- *ngram_range:* set to (1, 3) to include unigrams, bigrams and trigrams
- *analyzer:* set to 'word' based on earlier conclusions

Model Selection & Validation

After data pre-processing and feature engineering, the following models were tested and the final model was based on accuracy score for validation dataset. The initial validation of the models was performed using Sklearn's train_test_split() function with 20% data being used for validation. The summary of relevant model performances and corresponding observations have been included in the below table:

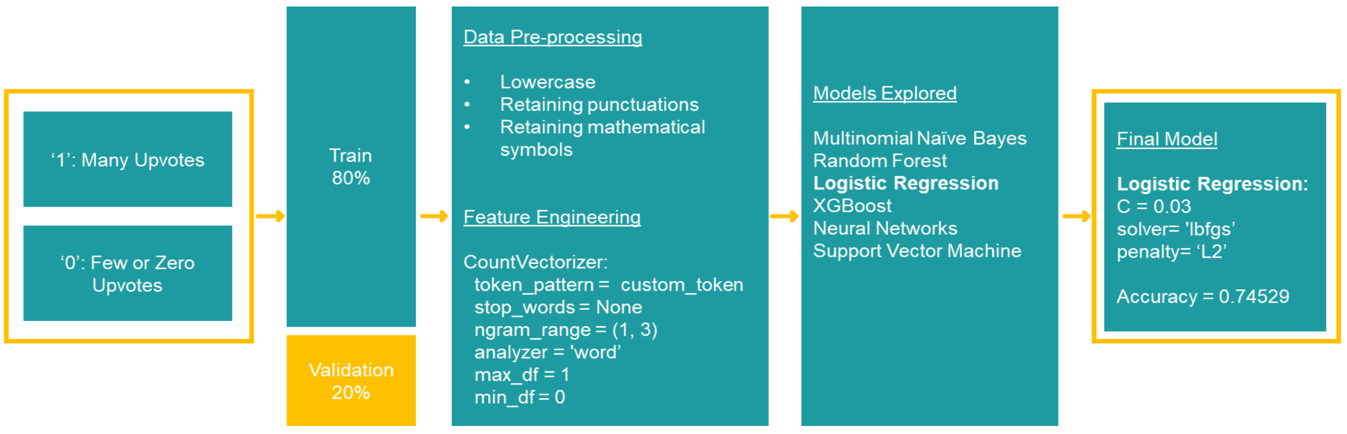
Model	Accuracy	Remarks
Naïve Bayes	0.6715	Poor accuracy
Random Forest	0.7145	Very slow
Logistic Regression	**0.7406**	**Fast and best accuracy**
XGBoost	0.7157	Slow but good accuracy
LSTM	0.7190	Slow and good accuracy
Support Vector Machine	0.6928	Extremely slow and computationally expensive
Ensemble of Logit models	0.7268	Fast and good accuracy

From the aforementioned models, Logistic Regression performed best during the training and validation process. Considering this and its high interpretability compared to other models, it was chosen as the final model for further analysis. To improve the accuracy of the Logistic Regression model further, it was trained using 95% training samples which led to an overall increase in accuracy during the validation process. Following this, GridSearchCV from the Sklearn library was used to find the best performing parameters for the model. These parameters include:

- *C = 0.03*: model accuracy improves significantly when stronger regularization is used
- *solver = 'lbfgs'*: no significant differences between different solvers
- *penalty = 'L2'*: model performs better with Ridge regression over Lasso regression

Furthermore, the model performance was validated using Confusion Matrix and corresponding Classification Report (refer Appendix 2)

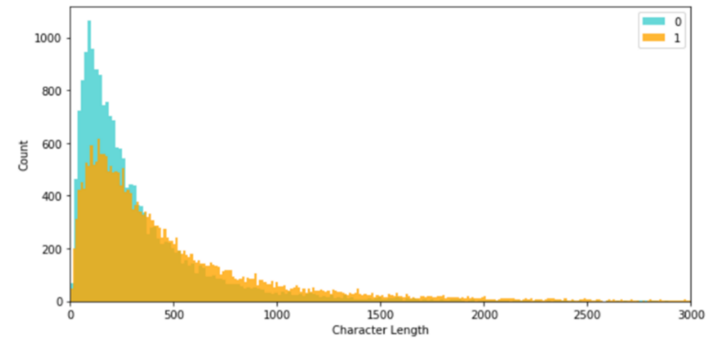
Overview of Methodology



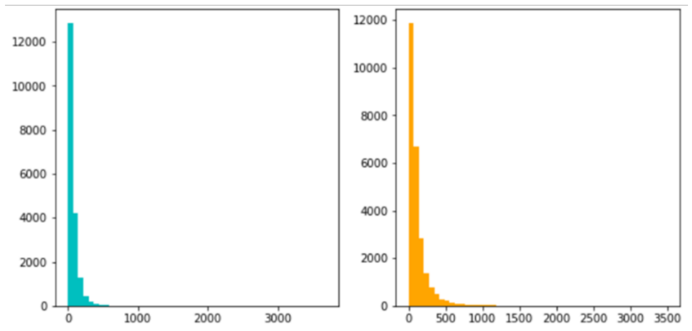
Appendix 1: EDA Plots

🟦 Class 0
🟧 Class 1

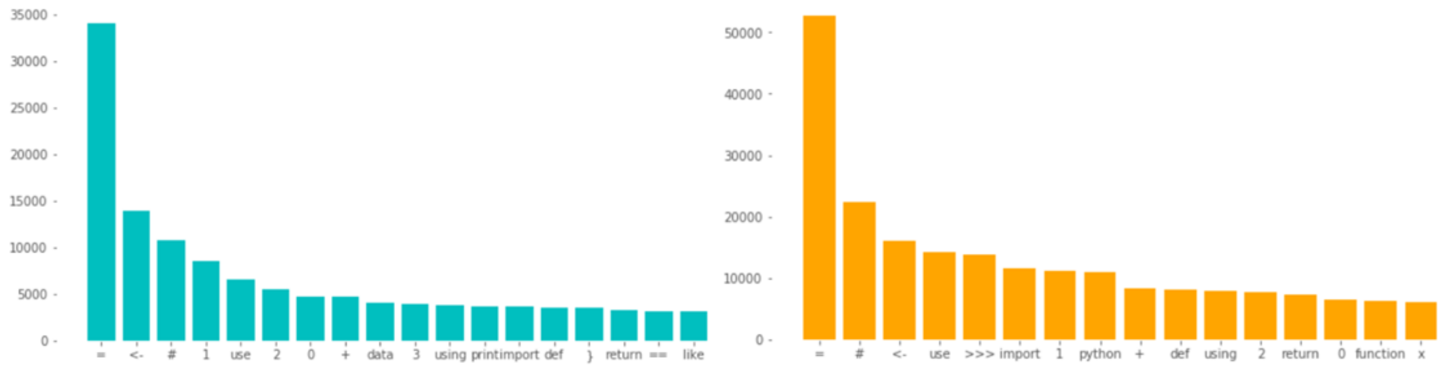
Distribution of number of characters in a comment



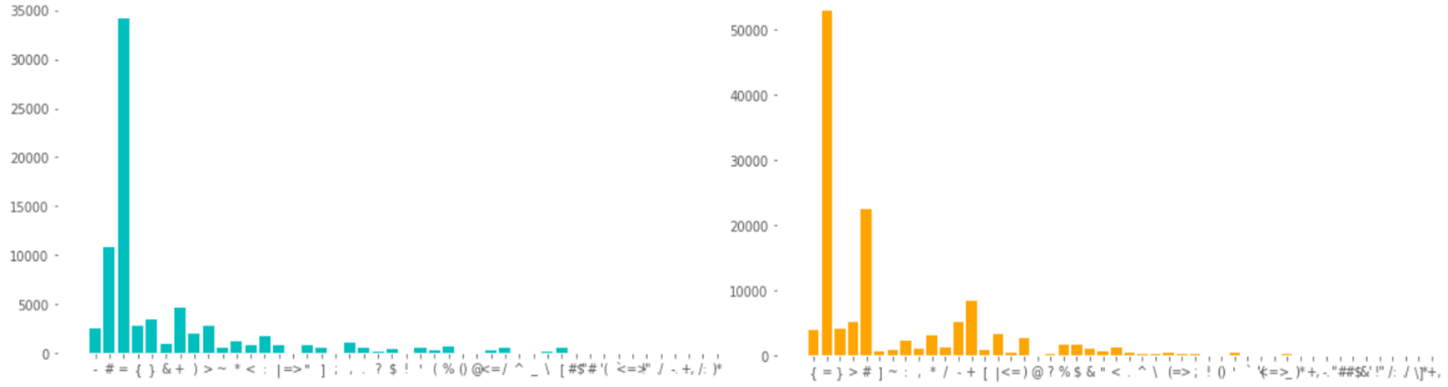
Distribution of words in a comment



Top 20 Common Words for Each Label



Punctuations and Symbols – for Custom Token Pattern

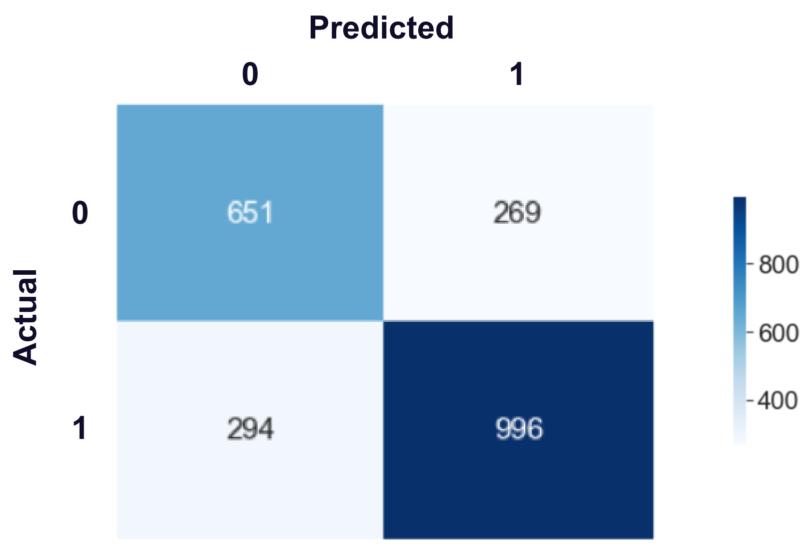


Appendix 2: Validation Plots (for final model)

Classification Report:

	Precision	Recall	F1-score	Support
0	0.71	0.69	0.70	945
1	0.77	0.79	0.78	1265
accuracy			0.75	2210
macro avg	0.74	0.74	0.74	2210
weighted avg	0.74	0.75	0.74	2210

Confusion Matrix:


Predicted
Actual