ECSE 551 Mini-Project 2: Implementing Bernoulli Naive Bayes for Text Classification

Yeo Kiah Huah
McGill University
ID: 261252535

Kaitlyn Pereira
McGill University
ID: 261023292

Melody Wang
McGill University
ID: 261053910

Abstract

This project explores text classification using machine learning, focusing on classifying Reddit posts and comments into four subreddits: Ottawa, Geneva, Canberra, and Boston. We implemented both a Bernoulli Naive Bayes and Multinomial Naive Bayes classifier from scratch and compared their performances with additional classifiers from the Scikit-learn library. The dataset consists of text data labeled by subreddit, which we processed using CountVectorizer with a maximum of 3000 features. To evaluate model performance, we used k-fold cross-validation and reported accuracy metrics. After selecting the best-performing model, we applied it to the test dataset and submitted predictions to a Kaggle competition. Our results indicate that the Multinomial Naive Bayes model, with Laplace smoothing, achieves the best classification performance over Bernoulli, while alternative classifiers such as Logistic Regression provide a useful benchmark. This study highlights the effectiveness of probabilistic and linear models for text classification tasks.

1 Introduction

Text classification is a fundamental problem in natural language processing (NLP) with applications in spam detection, sentiment analysis, and topic categorization. In this project, we focus on classifying Reddit posts and comments into their respective subreddits: Ottawa, Geneva, Canberra, and Boston. Given a text sample, our goal is to predict the subreddit it originated from based on learned patterns in the training data.

To achieve this, we implemented and compared several machine learning models: a Bernoulli Naive Bayes classifier (implemented from scratch), a Multinomial Naive Bayes classifier (also from scratch) and additional classifiers from the Scikit-learn library (BernoulliNB, LogisticRegression, and DecisionTreeClassifier). The dataset consists of labeled Reddit posts and comments, which were transformed using our custom preprocessing function. We experimented with different feature subsets, hyperparameters, and vectorization techniques to achieve the best performance, as evaluated using k-fold cross-validation. The final model was then used to generate predictions on the test set for submission to a Kaggle competition.

This report details our methodology, including feature extraction, model implementation, and validation. We also analyze the results, comparing the effectiveness of different classifiers in terms of accuracy and computational efficiency. Finally, we discuss key findings and potential directions for improvement in future work.

2 Dataset

The dataset for this project consists of Reddit posts and comments labeled with their corresponding subreddit of origin: Ottawa, Geneva, Canberra, and Boston. The training data (train.csv) includes two

Preprint. Under review.

fields: the text of a post or comment (**'body'**), and where it came from (**'subreddit'**). The test data (test.csv) contains only the text of posts and their unique review IDs, for which we need to generate predictions.

Through data exploration, we find that the dataset is evenly distributed, with 350 samples per class (Appendix Fig. 1).

When applying a vectorizer to the body of the train dataset, we determined that there are **12849** different words present.

2.1 Preprocessing Steps

To clean and prepare the data for machine learning models, we performed the several key operations.

First, all of the text was converted to lowercase to ensure uniformity and eliminate case sensitivity. Next, punctuation and special characters were removed, as they do not contribute to meaningful analysis in most natural language processing tasks. Tokenization is then applied to split the text into individual words or tokens. Following this, we removed commonly used words that do not carry significant meaning, such as "the," "and," and "is", using a predefined stopwords list. Finally, lemmatization is performed to reduce words to their root forms. ensuring that variations of the same word are treated identically. This means that variations of words like "go", "goes", "and "going" would all be reduced to the root word "go" and be treated identically.

To demonstrate the effect of the preprocessing, here is the original text of a random entry from the training set, and that same text after being passed through the preprocessing function.

Original:
```
Capital Rubbish Removals will come by and do a quote on what you need
gone, or you can probably send them photos.  I had them come by this
week to quote.  I?ve got them coming back next week to pick up a large
couch, bbq, outdoor setting, tv and microwave for $290.  They will
recycle where possible, but are also realistic about what they think
Goodies Junction will accept.  Office staff have been lovely and they
have good star rating.  I did have trouble with their online ?request
a quote?  page, it didn?t go through and I had to ring them.  I?ve
used them before a couple of years back and they were good then too.
They do this all the time and make it easy.  There are other companies
that perform similar services, just google ?rubbish removal Canberra?.
Hope that helps.
```

Processed:
```
capital rubbish removal will quote can send photo week quote come
week pick couch bbq outdoor set tv microwave will recycle possible
realistic think goody junction will accept office staff lovely star
rating trouble their online request quote ring couple year time easy
other company perform similar service google rubbish removal canberra
hope help
```

By eliminating common words and combining similar words, we are able to significantly cut down the input string, and allow the Naive Bayes algorithm to focus on the most important terms. When applying the vectorizer again, we see there are now **9730** different words present.

3 Proposed Approach

3.1 Text Preprocessing

As mentioned in section 2.1, we applied preprocessing techniques to the input text, using the Natural Language Toolkit package [1]. The key goal of this was standardize the text and remove any unnecessary words. Initially, we had used the built-in nltk 'english' stopwords, however we decided that this was not sufficient. Instead, we found a more complete list online from Stopwords ISO [2] and added some other words we noticed were unnecessary and specific to this task, such as "reddit" or "http".

Next, as indicated in the assignment instructions, we used CountVectorizer() with binary=True to obtain a binary vector representation for a text. We initially set max_features=3000, but experimented with other values. We also experimented with setting binary=False to get the absolute frequency of words, and with using TF-IDF Instead of Count Vectorization.

3.2 Bernoulli Naive Bayes Implementation

For text classification, a custom Bernoulli Naive Bayes classifier was implemented from scratch. This model is well-suited for document classification with binary features (presence/absence of terms) [3]. The class is initialized with an alpha parameter with a default value of 1.0 for Laplace smoothing. The fit method calculates class priors $P(y)$ by counting class frequencies in the training data (1) and feature probabilities $P(x_j|y)$ with Laplace smoothing (2). The predict method computes the posterior log probability $\log P(y|x)$ for each class (3) and selects the class with the highest posterior probability (4).

1. $P(y) = \frac{n_y}{N}$ where n_y is the number of samples in class y and N is the total number of samples
2. $P(x_j = 1|y) = \frac{n_{j,y}+\alpha}{n_y+2\alpha}$ where $n_{j,y}$ is the number of samples in class y with feature j present, and n_y is the total number of samples in class y
3. $\log P(y|x) \propto \log P(y) + \sum_{j=1}^{m}[x_j \log P(x_j = 1|y) + (1 - x_j)\log(1 - P(x_j = 1|y))]$ where x_j is 1 if feature j is present and 0 otherwise
4. $y_{pred} = \underset{y}{\operatorname{argmax}} \log P(y|x)$

3.3 Multinomial Naive Bayes Implementation

A Multinomial Naive Bayes classifier was also implemented from scratch [4]. Similarly, the fit method calculates class priors (1) and feature probabilities with Laplace smoothing (5). However, unlike the Bernoulli variant which uses binary features (presence/absence of terms), this implementation accounts for term frequencies in documents by weighing each word's class-conditional probability by the word's count in the input document. The predict method then computes the posterior log probability for each class (6) and selects the class with the highest posterior probability (4).

5. $P(w_i|y) = \frac{count(w_i,y)+\alpha}{\sum_{j=1}^{|V|} count(w_j,y)+\alpha\cdot|V|}$ where $count(w_i, y)$ is the frequency of word i in class y and $|V|$ is the vocabulary size
6. $\log P(y|\mathbf{x}) \propto \log P(y) + \sum_{i=1}^{|V|} x_i \log P(w_i|y)$ where x_i is the frequency of word i in the document $\mathbf{x}$

3.4 Additional SciKit Classifiers

To have some comparisons for our custom models, we imported LogisticRegression, DecisionTreeClassifier, and BernoulliNB from SciKit Learn.

3.5 Model Optimization

After comparing our custom models to the baseline SciKit Learn models, we performed some experiments to tune hyperparameters and improve the validation and test accuracies. These experiments include evaluating using different numbers of features, different values for alpha, and different vectorization techniques. More details can be found in the following section.

4 Results

4.1 Model Comparisons

Using KFold from SciKit learn, we tested each of the 5 models (Custom Bernoulli Naive Bayes, Custom Multinomial Naive Bayes, SciKit Bernoulli Naive Bayes, SciKit Logistic Regression and

SciKit Decision Trees) both on the original data and the preprocessed data. The results are summarized in the graphs shown in Appendix Fig. 2.

For all models except the decision trees, we note that preprocessing significantly improves validation accuracy and decreases elapsed time. Our custom Bernoulli Naive Bayes has the same accuracy as the SciKit implementation (**0.6307** without preprocessing and **0.6429** with), but takes much longer to complete. On the other hand, our custom Multinomial Naive Bayes class is the fastest to complete, and has the highest validation accuracy (**0.6743** without preprocessing and **0.6971** with).

4.2 Feature Subsets

Next, we want to experiment with including different subsets of features. To do this, we change the parameter **max_features** in the CountVectorizer. As we determined preprocessing consistently improves performance and training time, we will only use that data from now on. The results are summarized in Appendix Fig. 3.

For the Bernoulli Naive Bayes, validation accuracy is maximized at **0.6750** with 5000 features, while the Multinomial Naive Bayes model is maximized at **0.6971** using all the features (about 10000).

4.3 Hyperparameter Tuning

For the custom models, we had used a default alpha value of 1.0 for the above comparisons. To improve accuracy further, we perform experiments with different values of alpha to determine the optimal value, using the optimal feature subsets defined above. The results are summarized in Appendix Fig. 4.

The results show that both models had an optimal alpha parameter of 0.075, improving alpha from **0.6750** to **0.6950** for Bernoulli, and remaining the same at **0.6971** for Multinomial.

4.4 Vectorizer Experiments

Finally, we wanted to experiment with different vectorizers. While the instructions suggested using CountVectorizer() with binary=True to obtain a binary vector representation for a text, we also tried our optimized parameters with setting binary=False, and with using Term Frequency-Inverse Document Frequency (TfidfVectorizer). As shown in Appendix Fig. 5, the accuracy does not change when using Bernoulli Naive Bayes. However, for Multinomial Naive Bayes the TfidfVectorizer yields the greatest accuracy of **0.7086**.

4.5 Optimization Summary

Below is a table summarizing the validation accuracies at each stage of our investigations. This only includes our custom models (Bernoulli Naive Bayes and Multinomial Naive Bayes), and not the imported SciKit Learn functions.

Method	**Validation Accuracy (Bernoulli NB)**	**Validation Accuracy (Multinomial NB)**
Baseline	0.6307	0.6743
Preprocessing	0.6429	0.6971
Max. Features	0.6750	0.6971
Best Alpha	**0.6950**	0.6971
Vectorization	0.6950	**0.7086**

Table 1: Summary of Validation Accuracy Results for Different Methods

4.6 Test Set Accuracy (Kaggle Competition)

We generated predictions at various stages of our optimization process and submitted them to the Kaggle Competition. Since we were limited to the number of submissions per day, our optimizations had to be based on maximizing the validation accuracy. However, this did not match up perfectly with the test set accuracy. On 30% of the test data, the best Bernoulli Naive Bayes model scored a **0.6777**, while the best Multinomial Naive Bayes model scored a **0.6833**.

5 Discussions and Conclusions

5.1 Discussions

This project explored the effectiveness of Naive Bayes models for text classification, highlighting key factors that influence model performance. One critical insight was the impact of text preprocessing, where tokenization, removing stopwords, and applying stemming significantly improved classification accuracy. Additionally, the number of features included affects both the accuracy and the elapsed time, showing that more features isn't always better. Furthermore, we observed interesting performance differences between the Bernoulli Naive Bayes classifier and the Multinomial Naive Bayes classifier. We found that the Multinomial Naive Bayes model consistently outperformed the Bernoulli model, with higher accuracy in both validation and test sets, suggesting that accounting for term frequencies, as done in the Multinomial approach, provides additional valuable information for classification compared to the binary presence and absence of features used in Bernoulli. Both classifiers performed better on the validation set than on the test set, indicating potential overfitting to the training data, however the difference was fairly minimal.

5.2 Future Directions

There are several potential avenues for improving text classification performance beyond the methods used in this project. One promising approach is experimenting with **ensemble methods**, such as combining Naive Bayes and Logistic Regression, which could leverage the strengths of both models to enhance accuracy. Additionally, incorporating **deep learning models** like BERT could further refine classification by capturing contextual meaning and semantic relationships in text. Another area for exploration is looking at **n-grams**, rather than just representations of single words, to gain a better contextual representation of the input text. These enhancements could lead to more robust models capable of handling more complex language structures.

5.3 Conclusions

Ultimately, after evaluating various models and techniques, we found that **Multinomial Naive Bayes with all features, an alpha of 0.075 and TF-IDF Vectorization** achieved the best accuracy on validation and test data. This combination effectively balanced performance and computational efficiency, making it the most suitable choice for our final submission. While traditional machine learning models like Naive Bayes and Logistic Regression performed well, there remains significant potential for improvement by incorporating advanced techniques such as ensemble learning and deep learning. This project demonstrated the importance of feature selection and preprocessing in text classification, providing a strong foundation for future work in this domain.

6 Statement of Contributions

The work was divided equally between each group member:

Yeo - Scikit model comparisons, accuracy and k-fold validation.

Kaitlyn - Text preprocessing and model optimization, writing and formatting report.

Melody - Naive Bayes classes, writing report.

References

[1] NLTK Project, "NLTK: Natural Language Toolkit," 2025. [Online]. Available: https://www.nltk.org/.

[2] Stopwords ISO Github, "The most comprehensive collection of stopwords for multiple languages", Stopwords ISO, Oct 10, 2016. [Online]. Available: https://github.com/stopwords-iso/stopwords-en/blob/master/stopwords-en.txt

[3] Lecture Slides from ECSE-551: Machine Learning for Engineers by Prof. Armanfard

[4] GeeksforGeeks, "Multinomial Naive Bayes," 2021. [Online]. Available: https://www.geeksforgeeks.org/multinomial-naive-bayes/

Appendix A: Report Figures

The following section contains all figures referenced in the report. This figures are also available in better quality in the attached .ipynb file.

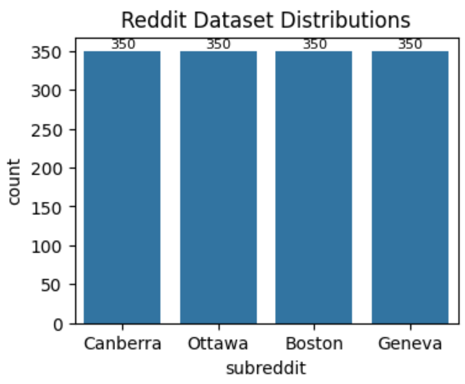
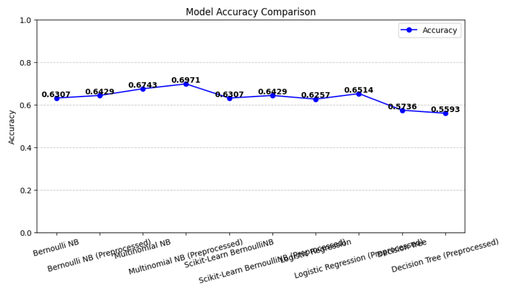
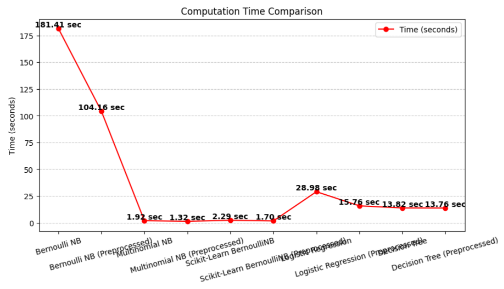


Figure 1: Distribution for Reddit Dataset

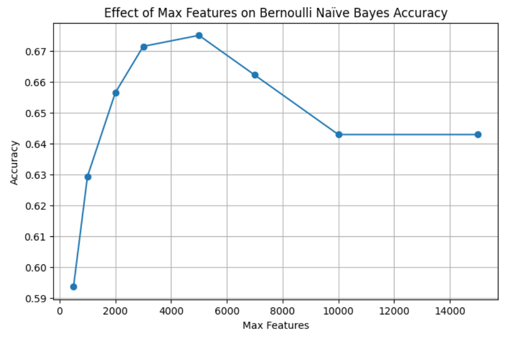


(a) Test accuracy for various models

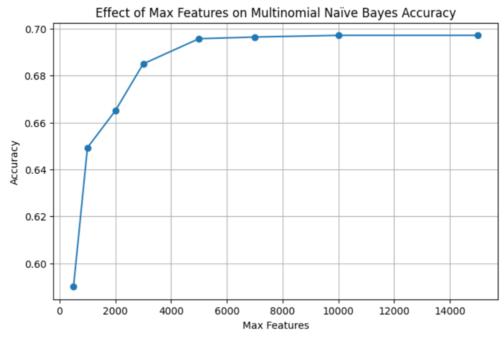


(b) Elapsed time for various models

Figure 2: Results of Different Models With and Without Preprocessing

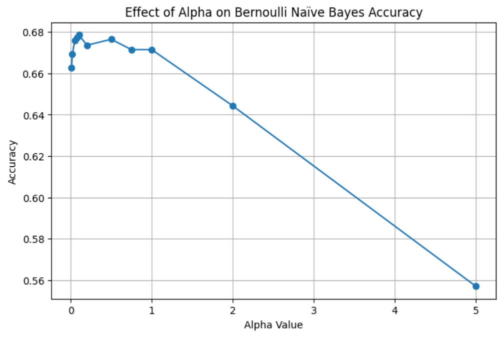


(a) Test accuracy for Bernoulli Naive Bayes

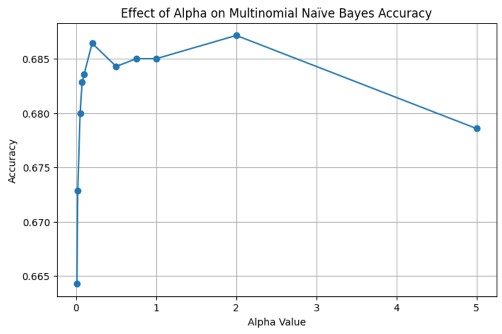


(b) Test accuracy for Multinomial Naive Bayes

Figure 3: Results of Varying Maximum Number of Features



(a) Test accuracy for Bernoulli Naive Bayes



(b) Test accuracy for Multinomial Naive Bayes

Figure 4: Results of Varying Alpha Values

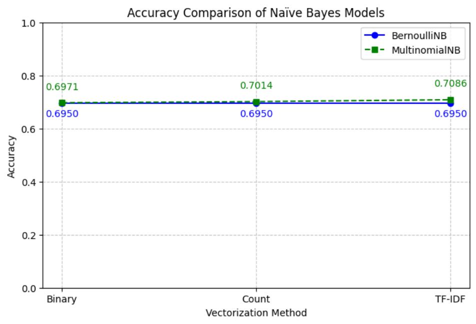


Figure 5: Results of Using Various Vectorizers

Appendix B: Code

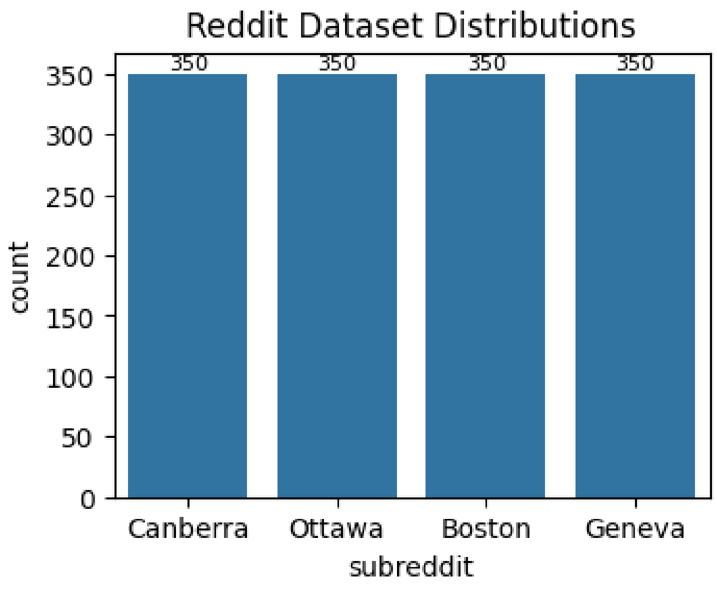
The following section contains the code in the form of a pdf file of the Jupiter Notebook. We recommend opening the actual code file, submitted alongside this report, to view the full output blocks and all relevant code/analysis.

ECSE551Assignment2

March 16, 2025

1 Dataset Exploration

[ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Load Data
train_df = pd.read_csv("train.csv", encoding="latin-1")

# Create the figure and axis
f, ax = plt.subplots(figsize=(4, 3))

# Plot the distributions
ax = sns.countplot(x="subreddit", data=train_df)
ax.set_title("Reddit Dataset Distributions")
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=8, color='black')

plt.show()
```



```
[ ]: # Load Data
     train_df = pd.read_csv("train.csv", encoding="latin-1")

     # Vectorization (Binary Features)
     vectorizer = CountVectorizer(binary=True)
     X = vectorizer.fit_transform(train_df['body']).toarray()

     selected_words = vectorizer.get_feature_names_out()
     print(len(selected_words))
```

```
12849
```

1.1 Data Preprocessing

english stopwords master list: https://github.com/stopwords-iso/stopwords-en/blob/master/stopwords-en.txt

```
[12]: stopwords_en = [
         "ll", "tis", "twas", "ve", "a", "able", "ableabout", "about", "above",␣
       ↪"abroad", "abst", "accordance",
         "according", "accordingly", "across", "act", "actually", "ad", "added",␣
       ↪"adj", "adopted", "ae", "af", "affected",
         "affecting", "affects", "after", "afterwards", "ag", "again", "against",␣
       ↪"ago", "ah", "ahead", "ai", "aint", "aint",
         "al", "all", "allow", "allows", "almost", "alone", "along", "alongside",␣
       ↪"already", "also", "although", "always", "am",
         "amid", "amidst", "among", "amongst", "amoungst", "amount", "an", "and",␣
       ↪"announce", "another", "any", "anybody", "anyhow",
         "anymore", "anyone", "anything", "anyway", "anyways", "anywhere", "ao",␣
       ↪"apart", "apparently", "appear", "appreciate",
         "appropriate", "approximately", "aq", "ar", "are", "area", "areas", "aren",␣
       ↪"arent", "arise", "around", "arpa", "as",
         "aside", "ask", "asked", "asking", "asks", "associated", "at", "auth",␣
       ↪"available", "aw", "away", "awfully", "az",
         "b", "ba", "back", "backed", "backing", "backs", "backward", "backwards",␣
       ↪"bb", "bd", "be", "became", "because", "become",
         "becomes", "becoming", "been", "before", "beforehand", "began", "begin",␣
       ↪"beginning", "beginnings", "begins", "behind",
         "being", "beings", "believe", "below", "beside", "besides", "best",␣
       ↪"better", "between", "beyond", "bf", "bg", "bh", "bi",
         "big", "bill", "billion", "biol", "bj", "bm", "bn", "bo", "both", "bottom",␣
       ↪"br", "brief", "briefly", "bs", "bt", "but",
         "buy", "bv", "bw", "by", "bz", "c", "cmon", "cs", "call", "came", "cannot",␣
       ↪"cant", "caption", "case", "cases",
   "cause", "causes", "cc", "cd", "certain", "certainly", "cf", "cg", "ch",␣
↪"changes", "ci", "ck", "cl", "clear", "clearly",
   "click", "cm", "cmon", "cn", "co", "co.", "com", "come", "comes",␣
↪"computer", "con", "concerning", "consequently", "consider",
   "considering", "contain", "containing", "contains", "copy",␣
↪"corresponding", "could", "couldve", "couldnt", "course", "cr",
   "cry", "cs", "cu", "currently", "cv", "cx", "cy", "cz", "d", "dare",␣
↪"daren't", "darent", "date", "de", "dear", "definitely",
   "describe", "described", "despite", "detail", "did", "didnt", "differ",␣
↪"different", "differently", "directly", "dj", "dk",
   "dm", "do", "does", "doesnt", "doing", "don", "dont", "done", "doubtful",␣
↪"down", "downed", "downing", "downs", "downwards",
   "due", "during", "dz", "e", "each", "early", "ec", "ed", "edu", "ee",␣
↪"effect", "eg", "eh", "eight", "eighty", "either",
   "eleven", "else", "elsewhere", "empty", "end", "ended", "ending", "ends",␣
↪"enough", "entirely", "er", "es", "especially",
   "et", "etal", "etc", "even", "evenly", "ever", "evermore", "every",␣
↪"everybody", "everyone", "everything", "everywhere", "ex",
   "exactly", "example", "except", "f", "face", "faces", "fact", "facts",␣
↪"fairly", "far", "farther", "felt", "few", "fewer",
   "ff", "fi", "fifteen", "fifth", "fifty", "fify", "fill", "find", "finds",␣
↪"fire", "first", "five", "fix", "fj", "fk", "fm",
   "fo", "followed", "following", "follows", "for", "forever", "former",␣
↪"formerly", "forth", "forty", "forward", "found", "four",
   "fr", "free", "from", "front", "full", "fully", "further", "furthered",␣
↪"furthering", "furthermore", "furthers", "fx", "g",
   "ga", "gave", "gb", "gd", "ge", "general", "generally", "get", "gets",␣
↪"getting", "gf", "gg", "gh", "gi", "give", "given",
   "gives", "giving", "gl", "gm", "gmt", "gn", "go", "goes", "going", "gone",␣
↪"good", "goods", "got", "gotten", "gov", "gp", "gq",
   "gr", "great", "greater", "greatest", "greetings", "group", "grouped",␣
↪"grouping", "groups", "gs", "gt", "gu", "gw", "gy",
   "h", "had", "hadnt", "half", "happens", "hardly", "has", "hasnt", "have",␣
↪"haven", "havent", "having", "he", "hed", "hell",
   "hello", "help", "hence", "her", "here", "hereafter", "hereby", "herein",␣
↪"heres", "hereupon", "hers", "herself", "he", "hi",
   "hid", "high", "higher", "highest", "him", "himself", "his", "hither",␣
↪"home", "homepage", "hopefully", "how", "howd", "howll",
   "hows", "howbeit", "however", "hr", "ht", "htm", "html", "http", "hu",␣
↪"hundred", "i", "id", "ie", "if", "ignored", "ii",
   "ill", "im", "immediate", "immediately", "importance", "important", "in",␣
↪"inasmuch", "inc", "inc.", "indeed", "index",
   "indicate", "indicated", "indicates", "information", "inner", "inside",␣
↪"insofar", "instead", "int", "interest", "interested",
    "interesting", "interests", "into", "invention", "inward", "io", "iq",␣
↪"ir", "is", "isnt", "it", "itd", "itll", "its", "itself",
    "ive", "je", "jm", "jo", "join", "jp", "just", "k", "ke", "keep", "keeps",␣
↪"kept", "keys", "kg", "kh", "ki", "kind", "km",
    "kn", "knew", "know", "known", "knows", "kp", "kr", "kw", "ky", "kz", "l",␣
↪"la", "large", "largely", "last", "lately", "later",
    "latest", "latter", "latterly", "lb", "lc", "least", "length", "less",␣
↪"let", "lets", "li", "like", "liked", "likely", "line",
    "little", "long", "longer", "longest", "look", "looking", "looks", "low",␣
↪"lower", "lr", "ls", "lt", "ltd", "lu", "lv", "ly",
    "m", "ma", "made", "mainly", "make", "makes", "making", "man", "many",␣
↪"may", "maynt", "mc", "md", "me", "mean", "means",
    "meantime", "meanwhile", "member", "members", "men", "merely", "mg", "mh",␣
↪"microsoft", "might", "mightve", "mightnt", "mil",
    "mill", "million", "mind", "mine", "minus", "miss", "misses", "ml", "mm",␣
↪"moment", "more", "moreover", "most", "mostly",
    "much", "mucho", "must", "mustnt", "my", "myself", "na", "name", "namely",␣
↪"nay", "nc", "nd", "near", "nearly", "necessarily",
    "necessary", "need", "needless", "needing", "needs", "neg", "neither",␣
↪"never", "nevermore", "nevertheless", "new", "newer",
    "newest", "next", "no", "nobody", "non", "none", "nonetheless", "noone",␣
↪"nor", "normally", "not", "noted", "nothing",
    "notwithstanding", "novel", "now", "nowadays", "nowhere", "nt", "numerous",␣
↪"obviously", "of", "off", "offered", "offering",
    "offers", "often", "oh", "okay", "old", "older", "oldest", "om", "omg",␣
↪"on", "onboard", "once", "one", "ones", "only",
    "onto", "open", "opens", "operate", "opposite", "or", "order", "ordered",␣
↪"ordering", "orders", "org", "organization",
    "ought", "oughtnt", "our", "ours", "ourselves", "out", "outside", "over",␣
↪"overall", "overcome", "overlook", "own",
    "owns", "p", "pa", "page", "pages", "part", "particular", "particularly",␣
↪"partly", "past", "per", "perhaps", "placed",
    "places", "plenty", "point", "pointed", "pointing", "points", "policy",␣
↪"possibly", "potential", "present", "presently",
    "presumably", "probably", "provides", "q", "question", "questions",␣
↪"quick", "quickly", "quite", "rather", "really",
    "reason", "reasons", "recent", "recently", "regardless", "regards",␣
↪"relatively", "respect", "respectively", "right",
    "rightly", "round", "ru", "run", "running", "ry", "s", "said", "same",␣
↪"saw", "say", "saying", "says", "sc", "second",
    "seconds", "see", "seeing", "seem", "seemed", "seeming", "seems", "seen",␣
↪"self", "selves", "sense", "sensitive", "seriously",
    "seven", "several", "shall", "shallnt", "shan", "shant", "she", "shed",␣
↪"shell", "shes", "should", "shouldnt", "show",
    "showed", "shown", "shows", "side", "sign", "significant", "similarly",␣
  ↪"since", "six", "size", "so", "some", "somebody",
    "somehow", "someone", "something", "sometime", "sometimes", "somewhat",␣
  ↪"somewhere", "soon", "sorry", "specifically",
    "state", "states", "still", "sub", "subsequently", "such", "summarize",␣
  ↪"summary", "suppose", "supposed", "sure", "surely",
    "t", "take", "taken", "takes", "taking", "tell", "tells", "than", "thank",␣
  ↪"thanks", "thanx", "that", "thatll", "thats",
    "thatd", "thats", "the", "them", "themselves", "then", "there",␣
  ↪"thereafter", "thereby", "therefore", "therein", "there'll",
    "there's", "thereupon", "these", "they", "theyre", "theyll", "this",␣
  ↪"thisll", "those", "though", "thoughly", "through",
    "throughout", "thus", "till", "to", "together", "too", "top", "totally",␣
  ↪"tough", "toward", "towards", "tried", "tries",
    "truly", "try", "trying", "ts", "thus", "twice", "two", "type",␣
  ↪"typically", "u", "un", "under", "unfortunately", "unless",
    "until", "up", "upon", "use", "used", "useful", "uses", "usually", "v",␣
  ↪"various", "very", "via", "vid", "view",
    "views", "vs", "w", "wa", "wait", "wants", "was", "wasnt", "wasnt", "we",␣
  ↪"wed", "well", "wells", "we'll", "we'd",
    "we've", "we've", "wellll", "went", "were", "weren", "werent", "werent",␣
  ↪"we've", "we'll", "willingly", "with", "within",
    "without", "wonder", "wonders", "wont", "words", "work", "worked",␣
  ↪"working", "works", "world", "would", "wouldve",
    "wouldnt", "wouldn't", "wrong", "x", "xa", "xe", "xi", "xii", "xiii",␣
  ↪"xml", "xrd", "ye", "yea", "yeah", "yet", "you",
    "youll", "you'd", "youre", "youve", "your", "yourself", "yours",␣
  ↪"yourself", "yourselves", "z", "zero", "zv",
    "who", "what", "when", "where", "why", "how", "reddit", "r", "www", "http",␣
  ↪"https"
]

import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import sklearn

nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')

lemmatizer = WordNetLemmatizer()
```

```
def get_wordnet_pos(word):
    """Map POS tagging to WordNet POS for better lemmatization."""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": nltk.corpus.wordnet.ADJ, "N": nltk.corpus.wordnet.NOUN,␣
  ↪"V": nltk.corpus.wordnet.VERB, "R": nltk.corpus.wordnet.ADV}
    return tag_dict.get(tag, nltk.corpus.wordnet.NOUN)

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'[-/\'._"]', ' ', text) # Replace '-', '/', and "'" with␣
  ↪space
    text = re.sub(r'[^\w\s]', '', text)# Remove punctuation
    words = word_tokenize(text)
    words = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) for word in␣
  ↪words if word not in stopwords_en]
    return ' '.join(words)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

```
[ ]: train_df = pd.read_csv("train.csv", encoding="latin-1")

     print("\nOriginal:")
     print(train_df['body'].iloc[252])
     print("\nProcessed:")
     print(preprocess_text(train_df['body'].iloc[252]))
```

```
Original:
Capital Rubbish Removals will come by and do a quote on what you need gone, or
you can probably send them photos. I had them come by this week to quote. I?ve
got them coming back next week to pick up a large couch, bbq, outdoor setting,
tv and microwave for $290. They will recycle where possible, but are also
realistic about what they think Goodies Junction will accept. Office staff have
been lovely and they have good star rating. I did have trouble with their online
?request a quote? page, it didn?t go through and I had to ring them. I?ve used
them before a couple of years back and they were good then too. They do this all
```

```
the time and make it easy.

There are other companies that perform similar services, just google ?rubbish
removal Canberra?.

Hope that helps.

Processed:
capital rubbish removal will quote can send photo week quote come week pick
couch bbq outdoor set tv microwave will recycle possible realistic think goody
junction will accept office staff lovely star rating trouble their online
request quote ring couple year time easy other company perform similar service
google rubbish removal canberra hope help
```

[ ]:
```python
# Load Data
train_df = pd.read_csv("train.csv", encoding="latin-1")

# Preprocess
train_df['body'] = train_df['body'].apply(preprocess_text)

# Vectorization (Binary Features)
vectorizer = CountVectorizer(binary=True)
X = vectorizer.fit_transform(train_df['body']).toarray()

selected_words = vectorizer.get_feature_names_out()
print(len(selected_words))
```

```
9730
```

2 Bernoulli Naïve Bayes Class

[13]:
```python
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score


class BernoulliNaiveBayes:

    def __init__(self, alpha=1.0):
        self.alpha = alpha # for LaPlace smoothing
        self.class_priors = {}
        self.feature_probs = {}
        self.classes = None
        self.n_features = None

    def fit(self, X, y):
        X = np.asarray(X)
        y = np.asarray(y)

        n_samples, self.n_features = X.shape

        self.classes = np.unique(y)
        n_classes = len(self.classes)

        # class priors P(y)
        for c in self.classes:
            self.class_priors[c] = np.log(np.sum(y == c) / n_samples)

        # feature probabilities P(x_j|y) with Laplace smoothing
        for c in self.classes:
            # samples for this class
            X_c = X[y == c]
            n_samples_c = X_c.shape[0]

            # feature probabilities for this class
            feature_probs_c = {}

            # for each feature, calculate P(x_j=1|y=c)
            for j in range(self.n_features):
                # number of samples where feature j is present (x_j=1)
                n_feature_present = np.sum(X_c[:, j])

                # Laplace smoothing
                feature_probs_c[j] = (n_feature_present + self.alpha) /␣
↪(n_samples_c + 2 * self.alpha)

            self.feature_probs[c] = feature_probs_c

       # return self

    def predict(self, X):
        X = np.asarray(X)
        n_samples = X.shape[0]
        predictions = []

        # for each sample
        for i in range(n_samples):
            class_log_probs = {}

            # for each class
            for c in self.classes:
                # start with log class prior
                log_prob = self.class_priors[c]

                # for each feature
                for j in range(self.n_features):
                    # get P(x_j=1|y=c)
                    p_feature_given_class = self.feature_probs[c][j]

                    # If feature is present (x_j=1), add log(P(x_j=1|y=c))
                    # If feature is absent (x_j=0), add log(1-P(x_j=1|y=c))
                    if X[i, j] == 1:
                        log_prob += np.log(p_feature_given_class)
                    else:
                        log_prob += np.log(1 - p_feature_given_class)

                class_log_probs[c] = log_prob

            # select class with highest log probability
            predicted_class = max(class_log_probs, key=class_log_probs.get)
            predictions.append(predicted_class)

        return np.array(predictions)

    def score(self, X, y):
        return accuracy_score(y, self.predict(X))
```

3 Multinomial Naive Bayes Class

[14]:
```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from scipy.sparse import issparse

class MultinomialNB:
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.class_log_prior_ = {}
        self.feature_log_prob_ = {}
        self.classes_ = None
        self.n_features_ = None

    def fit(self, X, y):
        X = np.asarray(X)
        if issparse(X):
            X = X.toarray()
        y = np.asarray(y)

        n_samples, self.n_features_ = X.shape
        self.classes_ = np.unique(y)
        n_classes = len(self.classes_)

        # Calculate class prior probabilities
        for c in self.classes_:
            self.class_log_prior_[c] = np.log(np.sum(y == c) / n_samples)

        # Calculate feature probabilities for each class with Laplace smoothing
        for c in self.classes_:
            # Select samples from this class
            X_c = X[y == c]

            # Calculate term frequencies for this class
            feature_count = np.sum(X_c, axis=0)

            # Total count of all features for this class
            total_count = np.sum(feature_count)

            # Apply Laplace smoothing and compute log probabilities
            smoothed_fc = feature_count + self.alpha
            smoothed_total = total_count + self.alpha * self.n_features_

            # Log probabilities for each feature given the class
            self.feature_log_prob_[c] = np.log(smoothed_fc / smoothed_total)

        return self
    def predict(self, X):

        X = np.asarray(X)
        if issparse(X):
            X = X.toarray()

        n_samples = X.shape[0]
        n_classes = len(self.classes_)

        # Initialize log probabilities for each class
        log_probs = np.zeros((n_samples, n_classes))

        # For each class, calculate log probability for each sample
        for i, c in enumerate(self.classes_):
            # Start with class prior log probability
            log_probs[:, i] = self.class_log_prior_[c]

            # Add weighted feature log probabilities
            for j in range(self.n_features_):
                # Weight feature log probability by feature value (term␣
↪frequency)
                log_probs[:, i] += X[:, j] * self.feature_log_prob_[c][j]

        return self.classes_[np.argmax(log_probs, axis=1)]

    def score(self, X, y):
        return accuracy_score(y, self.predict(X))
```

4 Model Evaluation

Perform K-fold Cross Validation and Evaluate the Models:

- Custom Bernoulli NB
- Custom Multinomial NB
- SciKit Bernoulli
- SciKit Logistic Regression
- SciKit Decision Tree

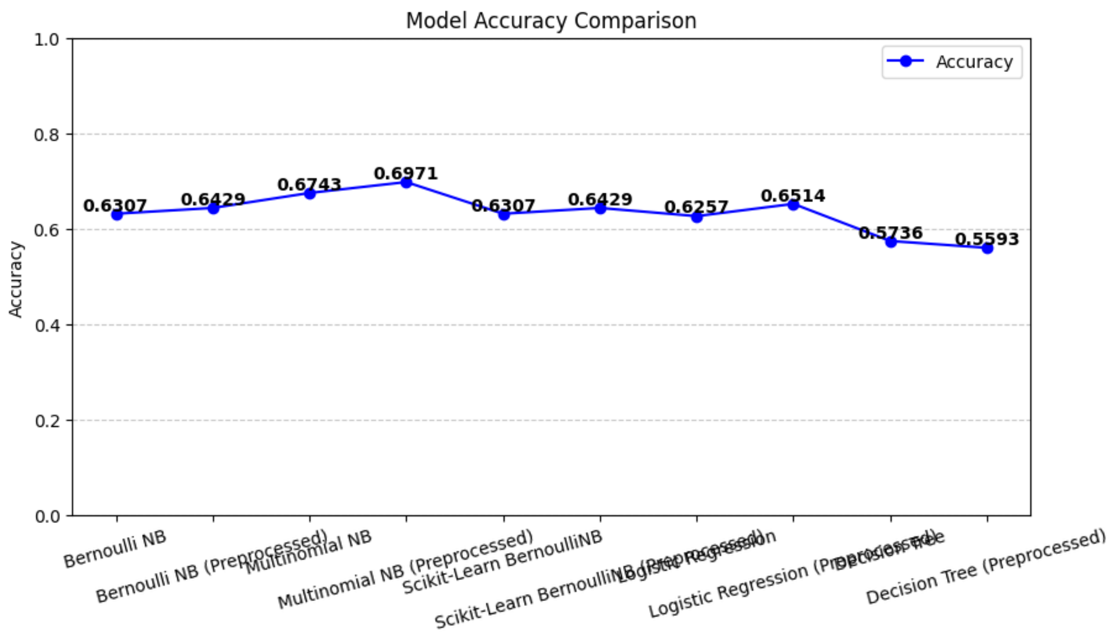
Evaluate all with and without preprocessing

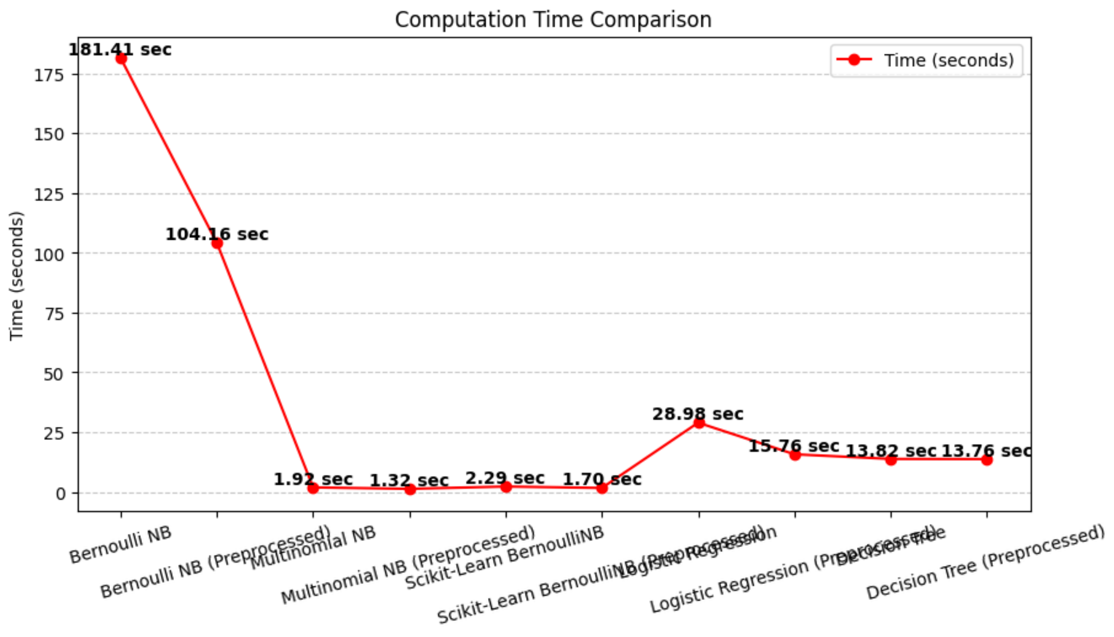
[ ]:
```
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier



# Load Data
```

```python
df = pd.read_csv("train.csv", encoding="latin-1")

# Define models and preprocessing settings
models = {
    "Bernoulli NB": BernoulliNaiveBayes(alpha=1.0),
    "Bernoulli NB (Preprocessed)": BernoulliNaiveBayes(alpha=1.0),
    "Multinomial NB": MultinomialNB(alpha=1.0),
    "Multinomial NB (Preprocessed)": MultinomialNB(alpha=1.0),
    "Scikit-Learn BernoulliNB": BernoulliNB(),
    "Scikit-Learn BernoulliNB (Preprocessed)": BernoulliNB(),
    "Logistic Regression": LogisticRegression(),
    "Logistic Regression (Preprocessed)": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Decision Tree (Preprocessed)": DecisionTreeClassifier(random_state=42),
}

accuracies = {}
times = {}

kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Iterate through each model type
for model_name, model in models.items():
    X_text = df["body"].copy()

    # Apply preprocessing if required
    if "Preprocessed" in model_name:
        X_text = X_text.apply(preprocess_text)

    y = df["subreddit"]

    vectorizer = CountVectorizer(binary=True)
    X = vectorizer.fit_transform(X_text).toarray()
    label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)

    model_accuracies = []
    start_time = time.time()

    # Perform KFold cross-validation
    for train_idx, test_idx in kf.split(X):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y_encoded[train_idx], y_encoded[test_idx]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        model_accuracies.append(acc)

    # Store results
    avg_accuracy = np.mean(model_accuracies)
    accuracies[model_name] = avg_accuracy
    times[model_name] = time.time() - start_time

    print(f"{model_name} - Accuracy: {avg_accuracy:.4f}, Time:␣
 ↪{times[model_name]:.2f} sec")

# Convert results into lists for plotting
model_names = list(accuracies.keys())
accuracy_values = list(accuracies.values())
time_values = list(times.values())

# Plot Accuracy
plt.figure(figsize=(10, 5))
plt.plot(model_names, accuracy_values, marker='o', linestyle='-', color='b',␣
 ↪label="Accuracy")
for i, txt in enumerate(accuracy_values):
    plt.text(model_names[i], accuracy_values[i], f"{txt:.4f}", ha='center',␣
 ↪va='bottom', fontsize=10, fontweight='bold')
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison")
plt.xticks(rotation=15)
plt.ylim(0, 1)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.legend()
plt.show()

# Plot Computation Time
plt.figure(figsize=(10, 5))
plt.plot(model_names, time_values, marker='o', linestyle='-', color='r',␣
 ↪label="Time (seconds)")
for i, txt in enumerate(time_values):
    plt.text(model_names[i], time_values[i], f"{txt:.2f} sec", ha='center',␣
 ↪va='bottom', fontsize=10, fontweight='bold')
plt.ylabel("Time (seconds)")
plt.title("Computation Time Comparison")
plt.xticks(rotation=15)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.legend()
plt.show()
```

```
Bernoulli NB - Accuracy: 0.6307, Time: 181.41 sec
Bernoulli NB (Preprocessed) - Accuracy: 0.6429, Time: 104.16 sec
```

```
Multinomial NB - Accuracy: 0.6743, Time: 1.92 sec
Multinomial NB (Preprocessed) - Accuracy: 0.6971, Time: 1.32 sec
Scikit-Learn BernoulliNB - Accuracy: 0.6307, Time: 2.29 sec
Scikit-Learn BernoulliNB (Preprocessed) - Accuracy: 0.6429, Time: 1.70 sec
Logistic Regression - Accuracy: 0.6257, Time: 28.98 sec
Logistic Regression (Preprocessed) - Accuracy: 0.6514, Time: 15.76 sec
Decision Tree - Accuracy: 0.5736, Time: 13.82 sec
Decision Tree (Preprocessed) - Accuracy: 0.5593, Time: 13.76 sec
```





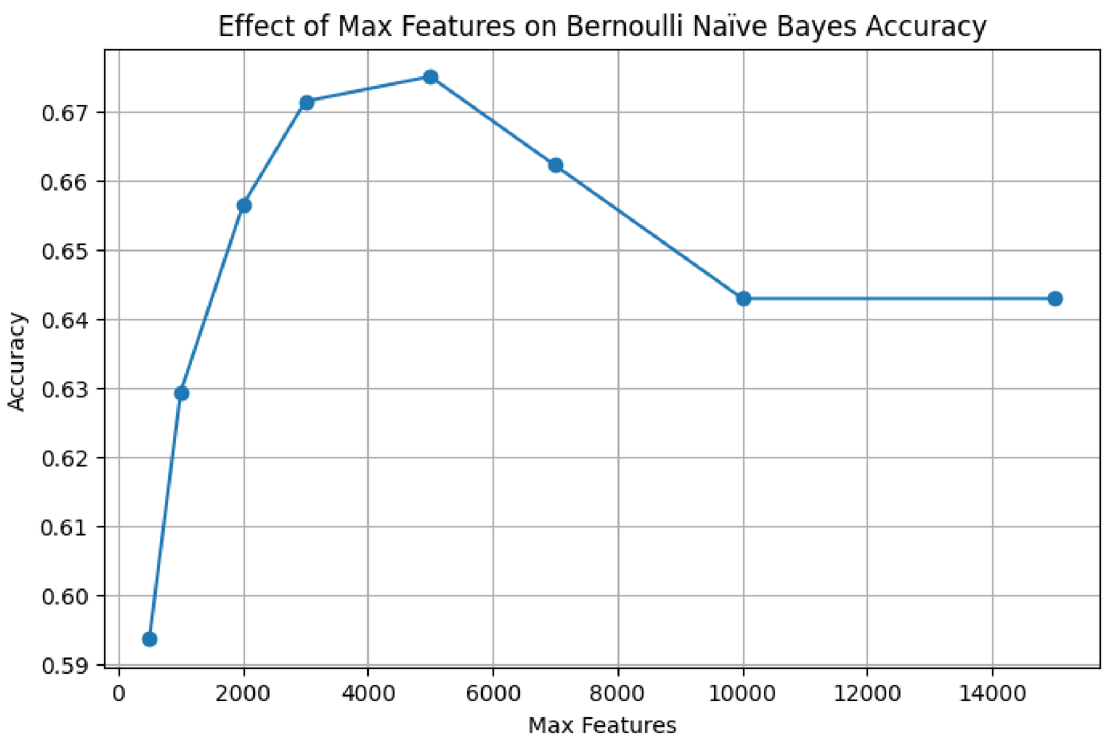
5 Model Optimization

Testing different max features

Bernoulli Naive Bayes

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

# Load Data
train_df = pd.read_csv("train.csv", encoding="latin-1")
test_df = pd.read_csv("test.csv", encoding="latin-1")

# Preprocess
train_df['body'] = train_df['body'].apply(preprocess_text)
test_df['body'] = test_df['body'].apply(preprocess_text)

# Define hyperparameter search space for max_features
max_features_values = [500, 1000, 2000, 3000, 5000, 7000, 10000, 15000]
best_max_features = None
best_accuracy = 0
feature_accuracies = []

kf = KFold(n_splits=5, shuffle=True, random_state=42)

for max_features in max_features_values:
    vectorizer = CountVectorizer(binary=True, max_features=max_features)
    X = vectorizer.fit_transform(train_df['body']).toarray()
    y = train_df['subreddit'].values

    accuracies = []

    for train_index, val_index in kf.split(X):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        nb_model = BernoulliNaiveBayes(alpha=1.0)  # Keep alpha fixed
        nb_model.fit(X_train, y_train)
        y_pred = nb_model.predict(X_val)
        accuracies.append(accuracy_score(y_val, y_pred))
    avg_accuracy = np.mean(accuracies)
    feature_accuracies.append(avg_accuracy)
    print(f"Max Features: {max_features}, Accuracy: {avg_accuracy:.4f}")

    if avg_accuracy > best_accuracy:
        best_accuracy = avg_accuracy
        best_max_features = max_features

print(f"\nBest max_features: {best_max_features}, Best Accuracy: {best_accuracy:
 ↪.4f}")

# Plot max_features vs accuracy
plt.figure(figsize=(8, 5))
plt.plot(max_features_values, feature_accuracies, marker='o', linestyle='-')
plt.xlabel("Max Features")
plt.ylabel("Accuracy")
plt.title("Effect of Max Features on Bernoulli Naïve Bayes Accuracy")
plt.grid(True)
plt.show()
```

```
Max Features: 500, Accuracy: 0.5936
Max Features: 1000, Accuracy: 0.6293
Max Features: 2000, Accuracy: 0.6564
Max Features: 3000, Accuracy: 0.6714
Max Features: 5000, Accuracy: 0.6750
Max Features: 7000, Accuracy: 0.6621
Max Features: 10000, Accuracy: 0.6429
Max Features: 15000, Accuracy: 0.6429

Best max_features: 5000, Best Accuracy: 0.6750
```



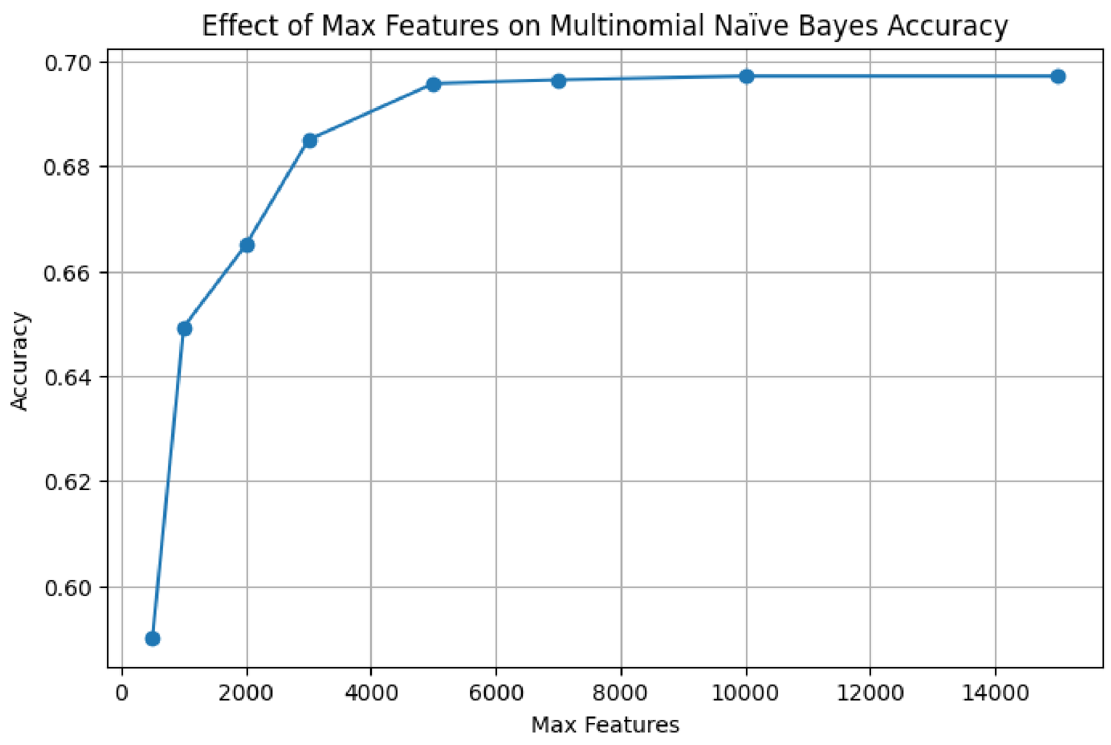
Multinomial Naive Bayes

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

# Load Data
train_df = pd.read_csv("train.csv", encoding="latin-1")
test_df = pd.read_csv("test.csv", encoding="latin-1")

# Preprocess
train_df['body'] = train_df['body'].apply(preprocess_text)
test_df['body'] = test_df['body'].apply(preprocess_text)

# Define hyperparameter search space for max_features
max_features_values = [500, 1000, 2000, 3000, 5000, 7000, 10000, 15000]
best_max_features = None
best_accuracy = 0
feature_accuracies = []
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

for max_features in max_features_values:
    vectorizer = CountVectorizer(binary=True, max_features=max_features)
    X = vectorizer.fit_transform(train_df['body']).toarray()
    y = train_df['subreddit'].values

    accuracies = []

    for train_index, val_index in kf.split(X):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        nb_model = MultinomialNB(alpha=1.0)  # Keep alpha fixed
        nb_model.fit(X_train, y_train)
        y_pred = nb_model.predict(X_val)
        accuracies.append(accuracy_score(y_val, y_pred))

    avg_accuracy = np.mean(accuracies)
    feature_accuracies.append(avg_accuracy)
    print(f"Max Features: {max_features}, Accuracy: {avg_accuracy:.4f}")

    if avg_accuracy > best_accuracy:
        best_accuracy = avg_accuracy
        best_max_features = max_features

print(f"\nBest max_features: {best_max_features}, Best Accuracy: {best_accuracy:
 ↪.4f}")

# Plot max_features vs accuracy
plt.figure(figsize=(8, 5))
plt.plot(max_features_values, feature_accuracies, marker='o', linestyle='-')
plt.xlabel("Max Features")
plt.ylabel("Accuracy")
plt.title("Effect of Max Features on Multinomial Naïve Bayes Accuracy")
plt.grid(True)
plt.show()
```

```
Max Features: 500, Accuracy: 0.5900
Max Features: 1000, Accuracy: 0.6493
Max Features: 2000, Accuracy: 0.6650
Max Features: 3000, Accuracy: 0.6850
Max Features: 5000, Accuracy: 0.6957
Max Features: 7000, Accuracy: 0.6964
Max Features: 10000, Accuracy: 0.6971
Max Features: 15000, Accuracy: 0.6971
```

```
Best max_features: 10000, Best Accuracy: 0.6971
```



Testing different alpha values

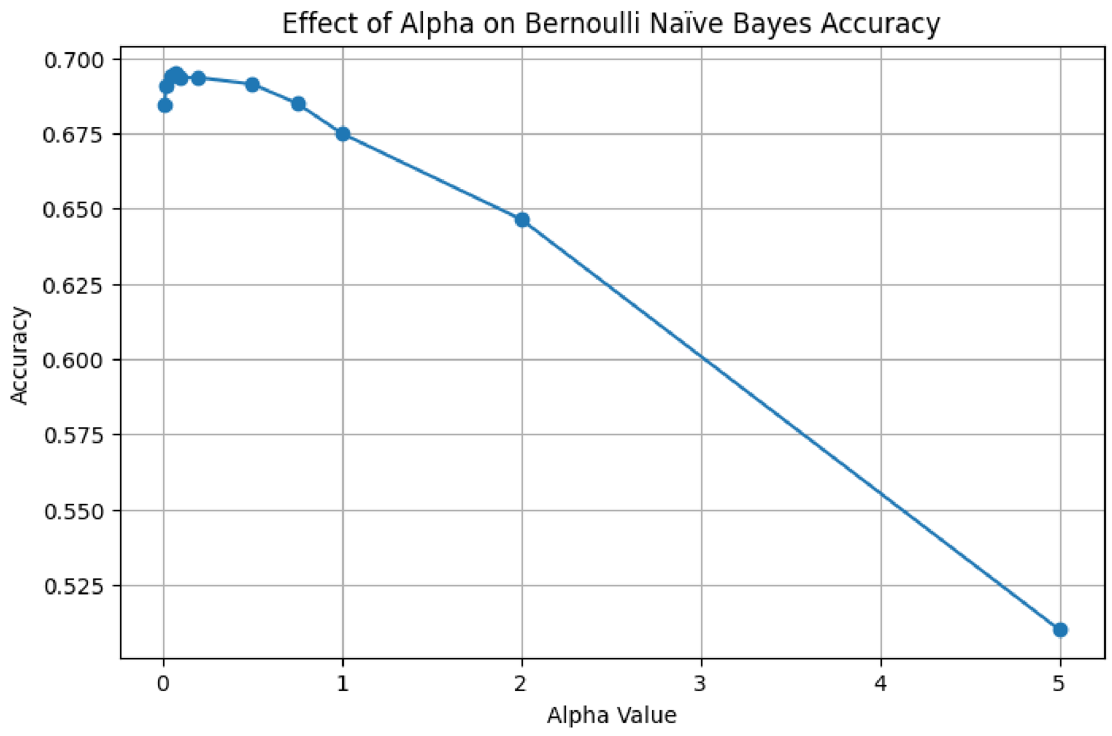
Bernoulli Naive Bayes

```
[ ]: import matplotlib.pyplot as plt

# Load Data
train_df = pd.read_csv("train.csv", encoding="latin-1")
test_df = pd.read_csv("test.csv", encoding="latin-1")

# Preprocess
train_df['body'] = train_df['body'].apply(preprocess_text)
test_df['body'] = test_df['body'].apply(preprocess_text)

vectorizer = CountVectorizer(binary=True, max_features=5000)
X = vectorizer.fit_transform(train_df['body']).toarray()
y = train_df['subreddit'].values

# Define hyperparameter search space
alpha_values = [0.01, 0.02, 0.05, 0.075, 0.1, 0.2, 0.5, 0.75, 1.0, 2.0, 5.0]
best_alpha = None
best_accuracy = 0
```

```python
alpha_accuracies = []

kf = KFold(n_splits=5, shuffle=True, random_state=42)

for alpha in alpha_values:
    accuracies = []

    for train_index, val_index in kf.split(X):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        nb_model = BernoulliNaiveBayes(alpha=alpha)
        nb_model.fit(X_train, y_train)
        y_pred = nb_model.predict(X_val)
        accuracies.append(accuracy_score(y_val, y_pred))

    avg_accuracy = np.mean(accuracies)
    alpha_accuracies.append(avg_accuracy)
    print(f"Alpha: {alpha}, Accuracy: {avg_accuracy:.4f}")

    if avg_accuracy > best_accuracy:
        best_accuracy = avg_accuracy
        best_alpha = alpha

print(f"\nBest alpha: {best_alpha}, Best Accuracy: {best_accuracy:.4f}")

# Plot alpha vs accuracy
plt.figure(figsize=(8, 5))
plt.plot(alpha_values, alpha_accuracies, marker='o', linestyle='-')
plt.xlabel("Alpha Value")
plt.ylabel("Accuracy")
plt.title("Effect of Alpha on Bernoulli Naïve Bayes Accuracy")
plt.grid(True)
plt.show()
```

```
Alpha: 0.01, Accuracy: 0.6843
Alpha: 0.02, Accuracy: 0.6907
Alpha: 0.05, Accuracy: 0.6943
Alpha: 0.075, Accuracy: 0.6950
Alpha: 0.1, Accuracy: 0.6936
Alpha: 0.2, Accuracy: 0.6936
Alpha: 0.5, Accuracy: 0.6914
Alpha: 0.75, Accuracy: 0.6850
Alpha: 1.0, Accuracy: 0.6750
Alpha: 2.0, Accuracy: 0.6464
Alpha: 5.0, Accuracy: 0.5100
```

```
Best alpha: 0.075, Best Accuracy: 0.6950
```

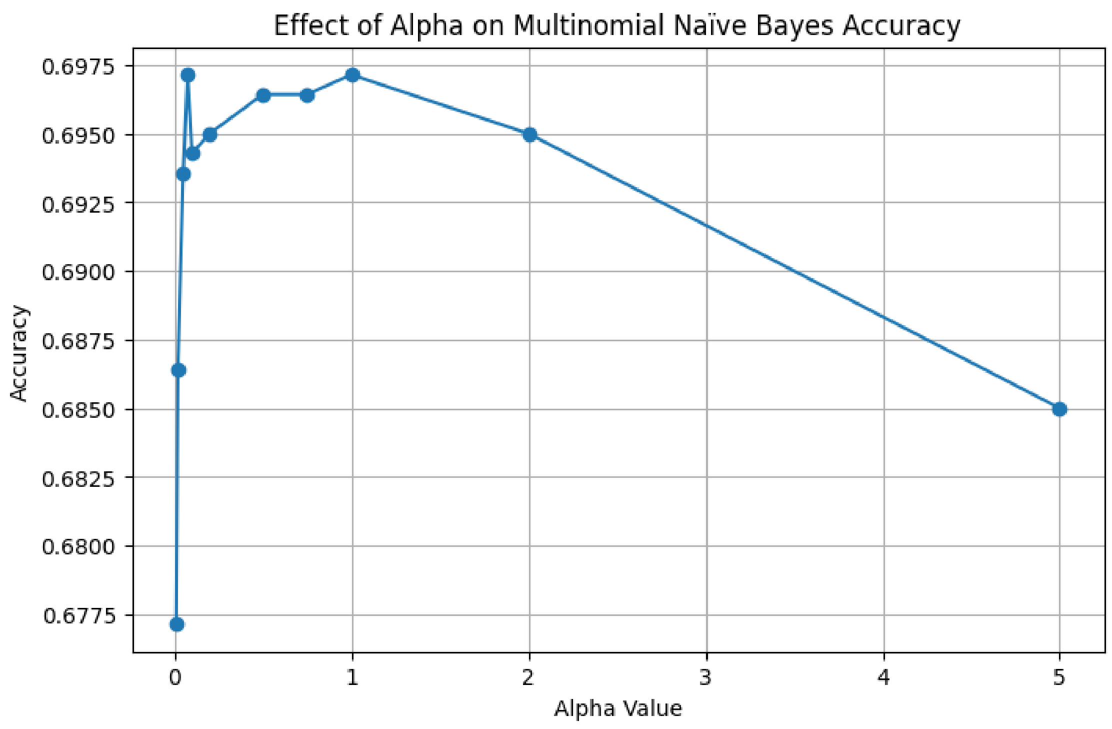


Multinomial Naive Bayes

```
[ ]: import matplotlib.pyplot as plt

     # Load Data
     train_df = pd.read_csv("train.csv", encoding="latin-1")
     test_df = pd.read_csv("test.csv", encoding="latin-1")

     # Preprocess
     train_df['body'] = train_df['body'].apply(preprocess_text)
     test_df['body'] = test_df['body'].apply(preprocess_text)

     vectorizer = CountVectorizer(binary=True, max_features=10000)
     X = vectorizer.fit_transform(train_df['body']).toarray()
     y = train_df['subreddit'].values

     # Define hyperparameter search space
     alpha_values = [0.01, 0.02, 0.05, 0.075, 0.1, 0.2, 0.5, 0.75, 1.0, 2.0, 5.0]
     best_alpha = None
     best_accuracy = 0
     alpha_accuracies = []
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

for alpha in alpha_values:
    accuracies = []

    for train_index, val_index in kf.split(X):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        nb_model = MultinomialNB(alpha=alpha)
        nb_model.fit(X_train, y_train)
        y_pred = nb_model.predict(X_val)
        accuracies.append(accuracy_score(y_val, y_pred))

    avg_accuracy = np.mean(accuracies)
    alpha_accuracies.append(avg_accuracy)
    print(f"Alpha: {alpha}, Accuracy: {avg_accuracy:.4f}")

    if avg_accuracy > best_accuracy:
        best_accuracy = avg_accuracy
        best_alpha = alpha

print(f"\nBest alpha: {best_alpha}, Best Accuracy: {best_accuracy:.4f}")

# Plot alpha vs accuracy
plt.figure(figsize=(8, 5))
plt.plot(alpha_values, alpha_accuracies, marker='o', linestyle='-')
plt.xlabel("Alpha Value")
plt.ylabel("Accuracy")
plt.title("Effect of Alpha on Multinomial Naïve Bayes Accuracy")
plt.grid(True)
plt.show()
```

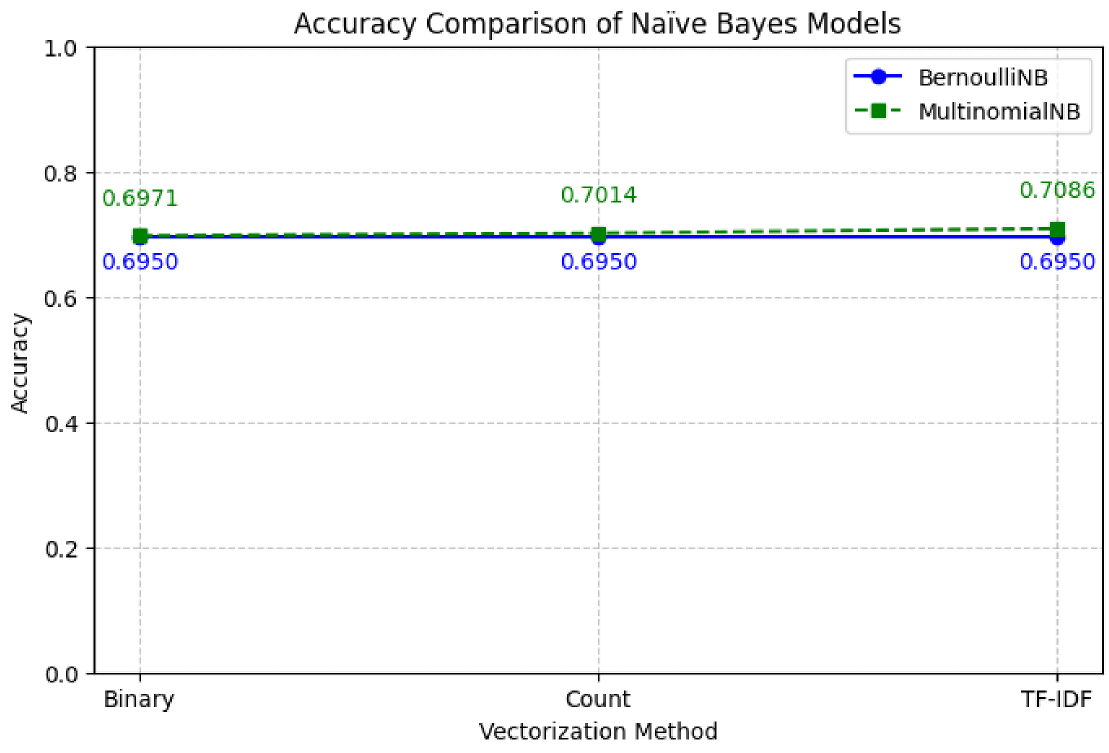
```
Alpha: 0.01, Accuracy: 0.6771
Alpha: 0.02, Accuracy: 0.6864
Alpha: 0.05, Accuracy: 0.6936
Alpha: 0.075, Accuracy: 0.6971
Alpha: 0.1, Accuracy: 0.6943
Alpha: 0.2, Accuracy: 0.6950
Alpha: 0.5, Accuracy: 0.6964
Alpha: 0.75, Accuracy: 0.6964
Alpha: 1.0, Accuracy: 0.6971
Alpha: 2.0, Accuracy: 0.6950
Alpha: 5.0, Accuracy: 0.6850

Best alpha: 0.075, Best Accuracy: 0.6971
```



Vectorizer Experiments

[16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import KFold
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score

# Load Data
train_df = pd.read_csv("train.csv", encoding="latin-1")
test_df = pd.read_csv("test.csv", encoding="latin-1")

# Preprocess
train_df['body'] = train_df['body'].apply(preprocess_text)
test_df['body'] = test_df['body'].apply(preprocess_text)

# Define hyperparameters
alpha = 0.075
max_features_bernoulli = 5000
kf = KFold(n_splits=5, shuffle=True, random_state=42)

vectorizers = {
    "Binary": CountVectorizer(binary=True),
    "Count": CountVectorizer(binary=False),
    "TF-IDF": TfidfVectorizer()
}

accuracies_bernoulli = {}
accuracies_multinomial = {}

for vec_name, vectorizer in vectorizers.items():
    print(f"\n{vec_name} Vectorization:")

    # BernoulliNB (with max_features limit)
    if vec_name == "Binary" or vec_name == "Count":
        vectorizer_bernoulli = CountVectorizer(binary=(vec_name == "Binary"),␣
 ↪max_features=max_features_bernoulli)
    else:
        vectorizer_bernoulli =␣
 ↪TfidfVectorizer(max_features=max_features_bernoulli)

    X_bernoulli = vectorizer_bernoulli.fit_transform(train_df['body']).toarray()
    y = train_df['subreddit'].values

    accuracies_bernoulli[vec_name] = []
    for train_index, val_index in kf.split(X_bernoulli):
        X_train, X_val = X_bernoulli[train_index], X_bernoulli[val_index]
        y_train, y_val = y[train_index], y[val_index]

        nb_model = BernoulliNB(alpha=alpha)
        nb_model.fit(X_train, y_train)
        y_pred = nb_model.predict(X_val)
        accuracies_bernoulli[vec_name].append(accuracy_score(y_val, y_pred))

    print(f"BernoulliNB ({vec_name}) Accuracy: {np.
 ↪mean(accuracies_bernoulli[vec_name]):.4f}")

    # MultinomialNB (no max_features limit)
    vectorizer_multinomial = vectorizer
    X_multinomial = vectorizer_multinomial.fit_transform(train_df['body']).
 ↪toarray()

    accuracies_multinomial[vec_name] = []
    for train_index, val_index in kf.split(X_multinomial):
        X_train, X_val = X_multinomial[train_index], X_multinomial[val_index]
        y_train, y_val = y[train_index], y[val_index]

        nb_model = MultinomialNB(alpha=alpha)
        nb_model.fit(X_train, y_train)
        y_pred = nb_model.predict(X_val)
        accuracies_multinomial[vec_name].append(accuracy_score(y_val, y_pred))

    print(f"MultinomialNB ({vec_name}) Accuracy: {np.
↪mean(accuracies_multinomial[vec_name]):.4f}")

# Plot the results
labels = list(vectorizers.keys())  # ["Binary", "Count", "TF-IDF"]
bernoulli_means = [np.mean(accuracies_bernoulli[label]) for label in labels]
multinomial_means = [np.mean(accuracies_multinomial[label]) for label in labels]

plt.figure(figsize=(8, 5))
plt.plot(labels, bernoulli_means, marker="o", linestyle="-",␣
↪label="BernoulliNB", color="blue")
plt.plot(labels, multinomial_means, marker="s", linestyle="--",␣
↪label="MultinomialNB", color="green")

# Add text labels
for i, label in enumerate(labels):
    plt.text(i, bernoulli_means[i] - 0.05, f"{bernoulli_means[i]:.4f}",␣
↪ha="center", fontsize=10, color="blue")
    plt.text(i, multinomial_means[i] + 0.05, f"{multinomial_means[i]:.4f}",␣
↪ha="center", fontsize=10, color="green")

plt.ylim(0, 1)
plt.xlabel("Vectorization Method")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison of Naïve Bayes Models")
plt.legend()
plt.grid(True, linestyle="--", alpha=0.7)
plt.show()
```

```
Binary Vectorization:
BernoulliNB (Binary) Accuracy: 0.6950
MultinomialNB (Binary) Accuracy: 0.6971

Count Vectorization:
BernoulliNB (Count) Accuracy: 0.6950
MultinomialNB (Count) Accuracy: 0.7014

TF-IDF Vectorization:
BernoulliNB (TF-IDF) Accuracy: 0.6950
MultinomialNB (TF-IDF) Accuracy: 0.7086
```



6 Submission Generation

Best Model, Bernoulli Naive Bayes

```
[ ]: train_df = pd.read_csv("train.csv", encoding="latin-1")
     test_df = pd.read_csv("test.csv", encoding="latin-1")

     train_df['body'] = train_df['body'].apply(preprocess_text)
     test_df['body'] = test_df['body'].apply(preprocess_text)

     vectorizer = CountVectorizer(binary=True, max_features=5000)
     X = vectorizer.fit_transform(train_df['body']).toarray()
     y = train_df['subreddit'].values

     model = BernoulliNaiveBayes(alpha=0.075)
     model.fit(X, y)

     X_test = vectorizer.transform(test_df['body']).toarray()
     test_df['subreddit'] = model.predict(X_test)

     # Save submission file
     submission = test_df[['id', 'subreddit']]
```

```
submission.to_csv("submission.csv", index=False)
```

this scored 0.6777 on the test set on kaggle

Best Model, Multinomial Naive Bayes

```
[ ]: train_df = pd.read_csv("train.csv", encoding="latin-1")
     test_df = pd.read_csv("test.csv", encoding="latin-1")

     train_df['body'] = train_df['body'].apply(preprocess_text)
     test_df['body'] = test_df['body'].apply(preprocess_text)

     vectorizer = TfidfVectorizer()
     X = vectorizer.fit_transform(train_df['body']).toarray()
     y = train_df['subreddit'].values

     model = MultinomialNB(alpha=0.075)
     model.fit(X, y)

     X_test = vectorizer.transform(test_df['body']).toarray()
     test_df['subreddit'] = model.predict(X_test)

     # Save submission file
     submission = test_df[['id', 'subreddit']]
     submission.to_csv("submission.csv", index=False)
```

this scored 0.6833 on the test set on kaggle

(functions to get code in pdf form)

```
[10]: !jupyter nbconvert --to pdf  "/content/drive/My Drive/Colab Notebooks/ECSE551/
       ↪ECSE551Assignment2.ipynb"
```

```
[NbConvertApp] Converting notebook /content/drive/My Drive/Colab
Notebooks/ECSE551/ECSE551Assignment2.ipynb to pdf
[NbConvertApp] Support files will be in ECSE551Assignment2_files/
[NbConvertApp] Making directory ./ECSE551Assignment2_files
[NbConvertApp] Writing 190091 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 438457 bytes to /content/drive/My Drive/Colab
Notebooks/ECSE551/ECSE551Assignment2.pdf
```