Fake News Identification

Final Report

1. Introduction

In today's world, there's an overwhelming amount of fake news and misinformation. It's becoming increasingly difficult for people and organizations to distinguish between what's true and what's false, especially since information spreads rapidly through social media and online platforms. This can seriously harm the reputation of individuals and companies, while also creating confusion and division among audiences. By developing machine learning algorithms that can accurately detect and separate fake news from real news, we can help protect reputations, promote reliable information, and reduce the chaos caused by misinformation.

2. Point of Interest

Developed a Chrome extension concept that analyzes online content and displays a True or False indicator for users. The extension also includes a fact-checking feature that tags posts on platforms like Facebook and Twitter, marking them as true or false based on verified information.

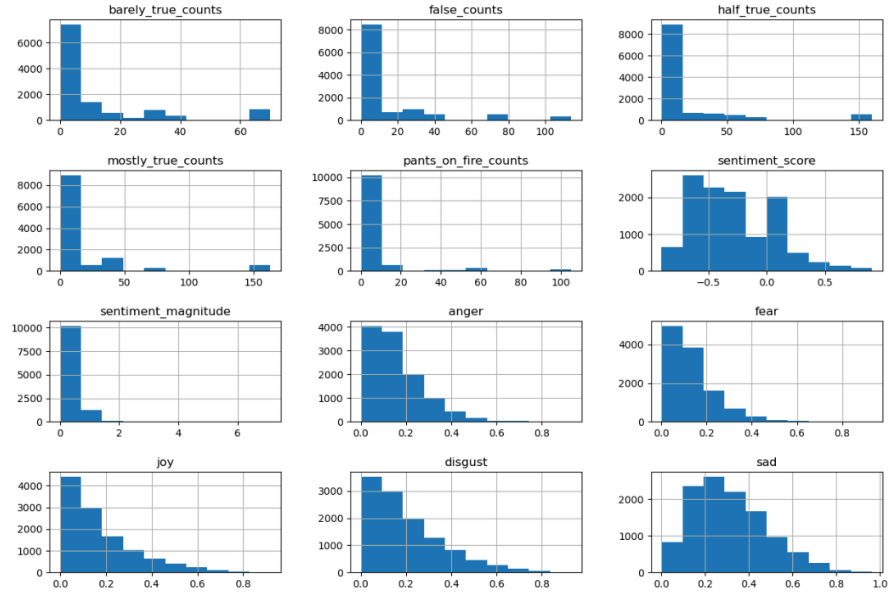
3. Data Wrangling

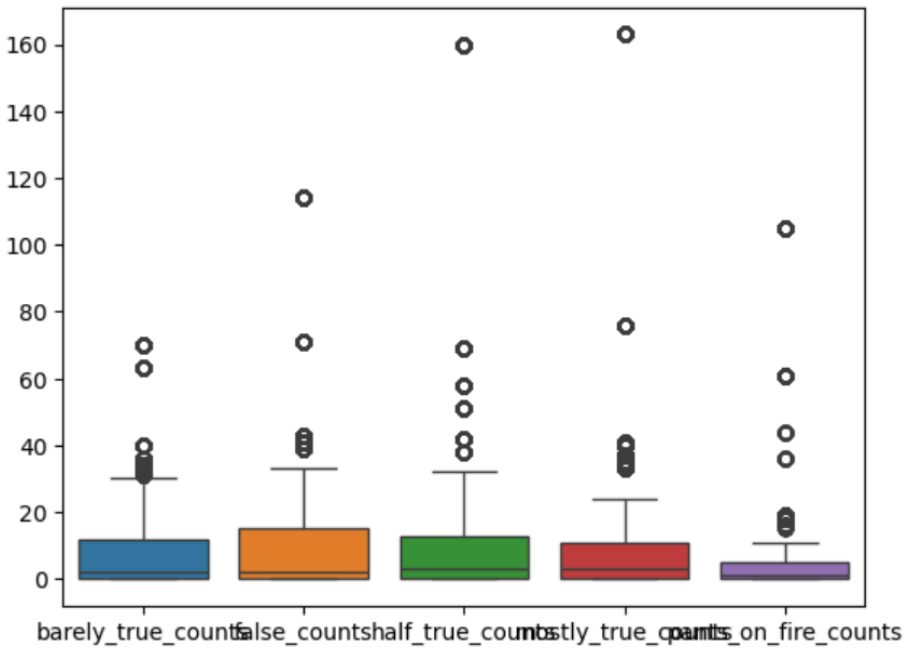
We began by examining the dataset to understand the different columns and the type of information they contained.

ID	2635.json
label	false
statement	Says the Annies List political group supports ...
subject	abortion
speaker	dwayne-bohac
speaker_job	State representative
state_info	Texas
party_affiliation	republican
barely_true_counts	0.0
false_counts	1.0
half_true_counts	0.0
mostly_true_counts	0.0
pants_on_fire_counts	0.0
context	a mailer
sentiment	NEGATIVE
sentiment_score	-0.5
sentiment_magnitude	0.5
anger	0.121137
fear	0.008926
joy	0.026096
disgust	0.263479
sad	0.531887
speaker_id	_0_
list	[0, 1]
sentiment_code	_NEG_

It was important to identify any missing data and assess the potential impact. In this dataset, several columns — including context, sentiment, sentiment_code, state_info, and speaker_job — had significant missing values. The dataset consisted of 25 columns and 11,518 rows. For this project, our primary focus was on the statement column, which contains the news content, while the label column served as the target variable for classification. Initially, when trying to print out the statements, the text was being truncated. This was resolved by setting pd.set_option("display.max_colwidth", None), allowing full text visibility. Next, I examined the

distribution of values for key categorical columns like label, subject, state_info, speaker, and party_affiliation using value_counts(). It was important to assess the diversity of the data for effective model building. I also visualized all numerical features and created boxplots to detect potential outliers, which could negatively impact model performance.



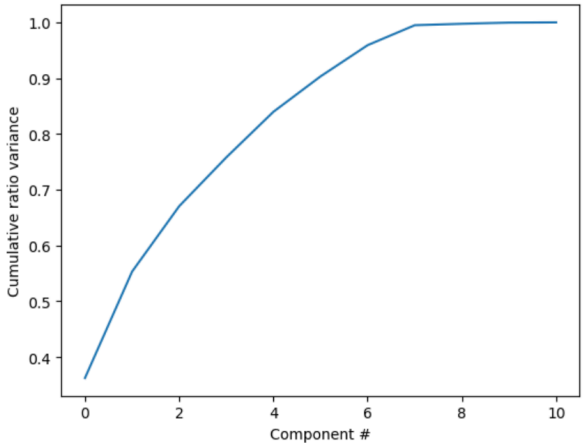


When reviewing the state_info column, I noticed it included more than 50 unique states and territories, but since the focus of this project was on the United States, it made sense to retain only records from U.S. states for better model generalization within a specific region. Additionally, duplicate records were identified and removed as they added no value to the analysis. Given the significant missing data in certain string-based columns, the decision was made to drop those rows, as imputing string values would be unreliable and inconsistent. Finally,

the dataset included a label called pants-fire, which indicated news stories that were entirely fabricated. For simplicity and better model clarity, these records were recorded under the false label category.

4. EDA

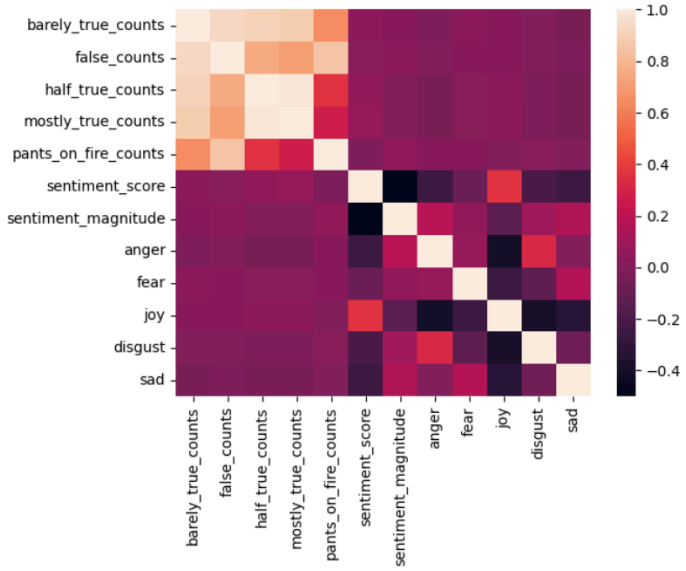
In further analyzing the data, the next step was to perform Principal Component Analysis (PCA) to identify which features contribute the most to the overall variance. Before applying PCA, it was necessary to scale the data to ensure consistency across feature ranges. After scaling, the data was fitted to the PCA model, and the resulting explained variance showed that approximately four features accounted for 80% of the variance, while about six features explained around 90%.



Before building a classification model, the target labels (True/False) were converted into numeric values. The statement column was selected as the predictive feature, and the label column as the target. The data was then split into training and testing sets for model evaluation. To extract meaningful patterns from the text data, TfidfVectorizer was used to convert the news statements into numerical feature vectors based on word importance. Following this, the Multinomial Naive Bayes (MultinomialNB) algorithm was applied. This probabilistic classifier, based on Bayes' Theorem, assumes the features follow a multinomial distribution, making it well-suited for text data like word counts and term frequencies. The model produced interesting results. For instance, words such as *half*, *40*, *three*, and *workers* were more commonly associated with statements labeled as true, while words like *Barack*, *Obamacare*, *Scott*, and *she* were frequently linked to false statements.

True words	P(word | %)
half	0.80
40	0.77
three	0.77
workers	0.76
weve	0.75
lowest	0.75
since	0.74
average	0.74
georgia	0.74
ohio	0.73
False words	**P(word | %)**
barack	0.43
gov	0.43
your	0.43
my	0.43
stimulus	0.43
her	0.43
she	0.41
wisconsin	0.40
scott	0.38
obamacare	0.34

Additionally, a correlation analysis was performed to check for relationships between other fields in the dataset. However, the results indicated little to no significant correlation between most features, limiting their usefulness for further model enhancement.



5. Pre - processing

It became evident that a binary classification approach was more effective than multiclass classification for this dataset. Therefore, labels like 'pants-fire' and 'barely-true' were consolidated into a false category, while 'half-true' and 'mostly-true' were grouped under true. To compare feature extraction techniques, both TfidfVectorizer and CountVectorizer were implemented. Model training began with CatBoostClassifier, LogisticRegression, and RandomForestClassifier. All models yielded similar accuracy scores. The CatBoost model, with a learning rate of 0.2 and CountVectorizer, achieved an accuracy of 0.61813, while it performed

slightly lower at 0.60453 with TfidfVectorizer. This is a confusion matrix and classification report for CatBoost.

```
confusion_matrix(y_test, y_pred)
```

```
array([[372, 398],
       [276, 719]], dtype=int64)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.57      0.48      0.52       770
           1       0.64      0.72      0.68       995

    accuracy                           0.62      1765
   macro avg       0.61      0.60      0.60      1765
weighted avg       0.61      0.62      0.61      1765
```

The LogisticRegression model scored 0.59603 using CountVectorizer and 0.60169 with TfidfVectorizer. The RandomForestClassifier followed a similar performance pattern. This is a confusion matrix and classification report for LogisticRegression (*left*) and RandomForestClassifier (*right*).

```
model2 = LogisticRegression()
model2.fit(xv_train2, y_train)
model2.score(xv_test2, y_test)
```

```
0.601699716713881
```

```
confusion_matrix(y_test, y_pred2)
```

```
array([[235, 535],
       [178, 817]], dtype=int64)
```

```
print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.57      0.31      0.40       770
           1       0.60      0.82      0.70       995

    accuracy                           0.60      1765
   macro avg       0.59      0.56      0.55      1765
weighted avg       0.59      0.60      0.57      1765
```

```
model3 = RandomForestClassifier()
model3.fit(xv_train2, y_train)
model3.score(xv_test2, y_test)
```

```
0.6022662889518413
```

```
confusion_matrix(y_test, y_pred3)
```

```
array([[322, 448],
       [230, 765]], dtype=int64)
```

```
print(classification_report(y_test, y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.58      0.42      0.49       770
           1       0.63      0.77      0.69       995

    accuracy                           0.62      1765
   macro avg       0.61      0.59      0.59      1765
weighted avg       0.61      0.62      0.60      1765
```

The primary goal of the project was not just to improve accuracy, but to gain insights into which words were more associated with true or false labels. Additionally, 5-fold cross-validation was performed for each model, confirming consistent results within the same accuracy range. To further refine model performance, hyperparameter tuning using GridSearchCV was conducted. For LogisticRegression, the optimal parameters were found to be {'C': 1, 'max_iter': 100, 'penalty': 'l2'} — though improvements remained modest. For RandomForestClassifier, the best parameters were {'bootstrap': True, 'n_estimators': 200}, resulting in a slightly improved accuracy of 0.610198.

6. Model

After evaluating and fine-tuning various models, the final selected model was Logistic Regression. The model's performance was primarily evaluated using the F-beta score, which is valuable because it allows control over the trade-off between precision and recall through the β parameter. In this case, the F-beta score achieved was 0.601. To better understand the model's behavior at different decision thresholds, the precision_recall_curve function was used to compute precision, recall, and the corresponding thresholds based on the predicted probabilities.

- Recall is the proportion of actual positive cases correctly identified.

- Precision is the proportion of positive predictions that are actually correct.

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.57      0.36      0.44       770
           1       0.61      0.79      0.69       995

    accuracy                           0.60      1765
   macro avg       0.59      0.57      0.57      1765
weighted avg       0.59      0.60      0.58      1765
```

```
# classification report which shows the affect of decearsin
print(classification_report(y_test, np.array(yprob) > 0.3))
```

```
              precision    recall  f1-score   support

           0       0.62      0.10      0.18       770
           1       0.58      0.95      0.72       995

    accuracy                           0.58      1765
   macro avg       0.60      0.53      0.45      1765
weighted avg       0.60      0.58      0.48      1765
```

```
# classification report which shows the affect of incerasin
print(classification_report(y_test, np.array(yprob) > 0.7))
```

```
              precision    recall  f1-score   support

           0       0.50      0.70      0.58       770
           1       0.66      0.46      0.54       995

    accuracy                           0.56      1765
   macro avg       0.58      0.58      0.56      1765
weighted avg       0.59      0.56      0.56      1765
```

By adjusting the threshold:

- Lowering the threshold (below 0.5) increases recall but decreases precision.

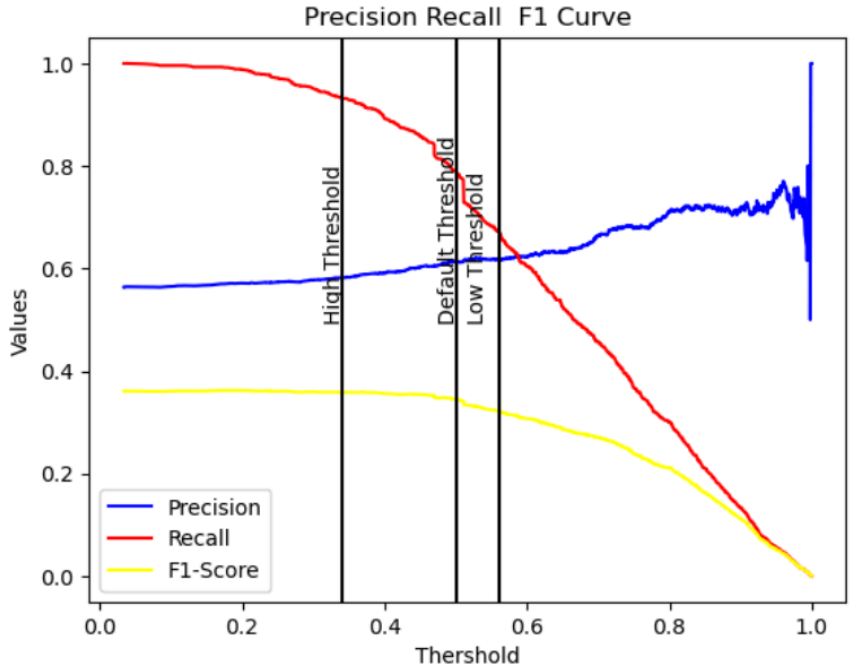
- Raising the threshold (above 0.5) increases precision but decreases recall.

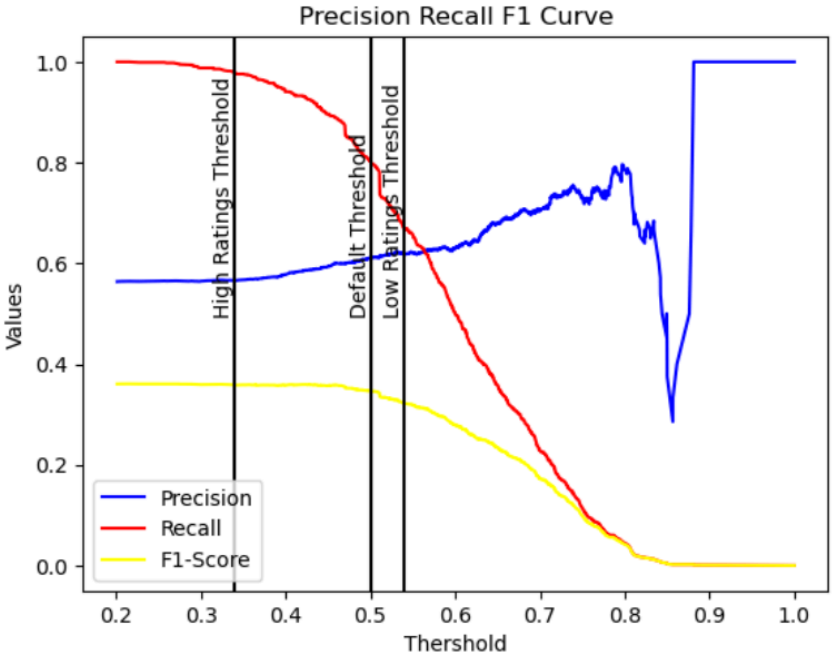
A plot of recall, precision, and F1 scores against thresholds revealed that as thresholds increase:

- Recall and F1 score decrease.

- Precision increases.

CountVectorizer with LogisticRegression





TfidfVectorizer with LogisticRegression

The optimal balance depends on the specific business case. For example: In a scenario focused on identifying false news, it is more critical to maximize precision, as it would be more damaging for the system to classify false news as true than to occasionally miss a false news case. Additionally, the ROC curve was plotted to visualize how the True Positive Rate (TPR) and

False Positive Rate (FPR) change across thresholds. The area under the ROC Curve (AUC) provides a single metric to summarize overall model performance. The ROC curve is generated by evaluating the model's performance at different threshold values, which determine how a model classifies data points. A probability score above the threshold is classified as positive, below is negative.