**Project Title: Binary Classification of Fake News and Real News**

Title: Binary Classification of Fake News and Real News

Date: March 14, 2024

Author: Hui Zhang

Abstract:

In this project, the application of various binary classification algorithms to label fake news and real news was explored. The dataset was cleaned, analyzed and preprocessed and new relevant features were created. The method applied for modeling:

1. For numeric variables, use StandardScaler() to scale them.

2. For categorical variables, use OneHotEncoder() to encode them.

3. For the “text” column, use CountVectorizer() to vectorize it.

**1. Introduction:**

**1.1 Background:**

In the era of information overload, the proliferation of fake news poses a significant threat to the integrity of news consumption. This project aims to develop a robust binary classification model to distinguish between fake news and real news articles. By leveraging advanced machine learning techniques, the system will contribute to the identification and mitigation of misinformation, promoting a more reliable and trustworthy news ecosystem.

**1.2 Objectives:**

Build a binary classification model: develop a machine learning model that can accurately label fake news and real news.

**1.3 Datasets:**

*Train Dataset*: 2 csv files from Kaggle

<https://www.kaggle.com/datasets/bhavikjikadara/fake-news-detection>

File for real news:

21417 rows and 5 columns, including information about title, text, subject, date and label.

Refer to this file as real\_news for following discuss.

File for fake news:

23481 rows and 5 columns, including information about title, text, subject, date and label.

Refer to this file as fake\_news for following discuss.

*Real\_world unseen test set*, 1 csv files

Collect 25 pieces of news from internet.

the 13 real news comes from reuters (7 news), cnn (2 news) and npr(4 news),

the 12 real news comes from breitbart (5 news) and thegatewaypundit (7 news).

All news is most recently on 02/26/2024 to 03/01/2024.

**2. Methodology:**

**2.1 Data Collection:**

Download train set csv files from the websites listed above.

Find news from internet and paste to a csv file, remove location, sources and notes to get test set.

**2.2 Data Wrangling:**

Below is an overview of problems and solutions during cleaning the data:

* Problem 1: fake news and real news are in two separate files.

Solution 1:

Combine these two files using pd.concat() method, then shuffle the dataset with df.sample(frac=1, random\_state=42).

* Problem 2: the date column is an object, and have different formats and errors

Solution 2:

First remove all the errors in date column, especially some rows have urls in date column.

Second, check all the formats used in the date column, like February 13, 2017, Feb 13, 2017, and 13-Feb-17, use datetime.strptime() to parse the date with these three formats ('%B %d, %Y', '%b %d, %Y', '%d-%b-%y').

%B means full month name,

%b means abbreviated month name,

%d means day of the month as a zero-padded decimal number,

%Y means year with century as a decimal number,

%y means year without century as a zero-padded decimal number.

* Problem 3: there are duplicates in title, text and date columns.

Solution 3:

First check the number of rows having same title, same text and same date. For these rows, drop duplicates and only keep the first.

Second, check the number of rows only having same title. After that, created a new dataframe grouped by the same title and put all texts with same title into a list. Then transformed the list with TfidfVectorizer() and calculated the cosine similarity.

For rows with only two texts and cosine similarity larger than 0.98, only keep first.

For rows with only two texts and cosine similarity less than 0.98, keep both.

For rows with more than two texts, check manually to decide.

Third, check the number of rows only having same text.

For rows having same text which are empty, drop them.

For rows having same text which are not empty, check manually to decide.

* Problem 4: feature engineering

Solution 4:

Calculate the count of characters in each title, calculate the count of words in each title, calculate the average word length in each title.

Calculate the count of characters in each text, calculate the count of words in each text, calculate the average word length in each text.

Extract year, month, day, and day of week for each date.

Add new row as election day, set value to November 08, 2016, which is election day in 2016, then calculate the days to election day for each date.

**2.3 Exploratory Data Analysis:**

Below is an overview of the main issues I ran into while analyzing the data:

* Figure 1: barplot of fake news and real news proportions

A blue rectangular bars with text

Description automatically generated with medium confidence

Findings 1: the proportions of fake news and real news are 54.5% and 45.5%, just little bit imbalanced.

* Figure 2: countplots of fake news and real news.

A graph of blue and orange bars

Description automatically generated

A graph of a bar chart

Description automatically generated with medium confidenceA graph of a number of bars

Description automatically generated with medium confidence

A graph of blue and orange bars

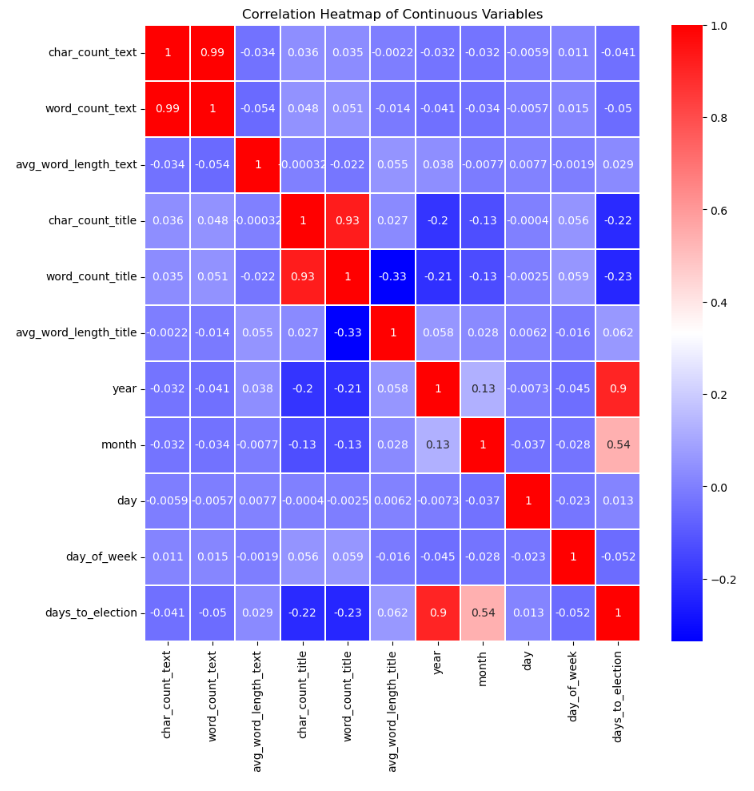
Description automatically generatedA graph of a person's data

Description automatically generated with medium confidence

Findings 2: there is obvious pattern of news distribution by subjects, but the subject column is not suitable for the modeling because it has obvious bias which means that it cannot be generalized.

There are also some patterns of news distribution by years, months, day of week and days to election.

* Figure 3: heatmap



Findings 3: based on the heatmap, there is no obvious correlation between variables, except word count and character count, year and days to election. This observance makes sense.

**2.4 Data Training and Modeling:**

Below is an overview of problems and solutions during data training and modeling:

* Problem 1:data preprocessing before modeling

Solution 1:

First, convert some numeric features to categorical, like month, change 1 to ‘Jan etc, like day\_of\_week, change 0 to ‘Monday’.

Second, use pd.get\_dummies() to encode categorical variables, month and day\_of\_week.

Third, use train\_test\_split(test\_size=0.3, stratify=y) to get train set and validation set.

* Problem 2: features selection and modeling

Solution 2:

1. For numeric variables, use StandardScaler() to scale them.

2. For categorical variables, use OneHotEncoder() to encode them.

3. For the “text” column, use CountVectorizer() to vectorize it.

Model: Logistic Regression, Random Forest, Gradient Boost, and XGBoost.

* Problem 3: how to avoid overfitting

Solution 3: use cross validation in scikit-learn with cv=5.

* Problem 4: metrics used for model evaluation

Solution 4: monitor accuracy of train set, validation set and test set, precision score and recall score of validation set and test set.

* Problem 5: the importance of unseen test set

Solution 5: An unseen dataset helps evaluate how well a model generalizes to new, unseen examples. If a model performs well on the training data but poorly on unseen data, it may indicate overfitting, where the model has memorized the training set rather than learning the underlying patterns.

**2.5 Results and Discussion:**

2.5.1 Hybrid Model Performance Comparison:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exp | Model\_Name | Features | Scale | Vectorizer | Train\_Accuracy | Val\_Accuracy | Val\_Recall | Val\_Precision | Val\_F1 |
| 1\_Hybrid | Logistic Regression (Pipeline) | Text+Meta | StandardScaler | CountVectorizer | 0.999 | 0.981 | 0.976 | 0.982 | 0.979 |
| 2\_Hybrid | Random Forest (Pipeline) | Text+Meta | StandardScaler | CountVectorizer | 1 | 0.972 | 0.957 | 0.98 | 0.968 |
| 3\_Hybrid | Gradient Boost (Pipeline) | Text+Meta | StandardScaler | CountVectorizer | 0.976 | 0.971 | 0.957 | 0.978 | 0.967 |
| 4\_Hybrid | XGboost (Pipeline) | Text+Meta | StandardScaler | CountVectorizer | 0.999 | 0.987 | 0.983 | 0.989 | 0.986 |

The initial experiments compared four different classifiers (Logistic Regression, Random Forest, Gradient Boost, and XGBoost) using a hybrid approach that combined engineered metadata features with a **CountVectorizer** representation of the text. As shown in the comparison table, all four models achieved excellent performance on the validation set, with validation accuracies ranging from 0.971 to 0.987. The **XGBoost Classifier was the clear top performer**, achieving the highest validation accuracy (0.987), recall (0.983), precision (0.989), and F1-score (0.986).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Exp | Model\_Name | Features | Scale | Vectorizer | Test\_Accuracy | Test\_Recall | Test\_Precision | Test\_F1 | Vectorizer\_Params |
| 4\_Hybrid | XGboost (Pipeline) | Text+Meta | StandardScaler | CountVectorizer | 0.84 | 0.917 | 0.786 | 0.846 | max\_features=None, ngram\_range=(1,1), max\_df=1, min\_df=1 |
| 5\_Hybrid | XGboost (Pipeline) | Text+Meta | StandardScaler | CountVectorizer | 0.84 | 0.917 | 0.786 | 0.846 | max\_features=2000, ngram\_range=(1,3), max\_df=0.8, min\_df=5 |

When this top-performing XGBoost model was evaluated on the unseen test set, it achieved a test accuracy of 0.84 and an F1-score of 0.846. The significant drop from a validation accuracy of ~0.99 to a test accuracy of 0.84 indicates that the model is **overfitting to the training data** and does not generalize perfectly. Further investigation into tuning the **CountVectorizer** parameters (changing **max\_features**, **ngram\_range**, **max\_df**, and **min\_df**) yielded no change in the test set performance, suggesting that the model's predictive power is heavily reliant on the non-text metadata features rather than the nuances of the text itself.

2.5.2 Feature Importance Analysis:

A graph of a graph showing different colors

AI-generated content may be incorrect.

A graph of a number of words

AI-generated content may be incorrect.

Analysis of the feature importances for the XGBoost model provides a clear explanation for its behavior. The most influential features are not traditional keywords but rather **structural and stylistic artifacts**. As seen in the feature importance plot, terms related to social media, such as **"twitter com"**, **"pic"**, **"httpsis"**, and the user handle **"realdonaldtrump"**, are highly dominant. This indicates the model has learned a powerful shortcut: **the presence of embedded tweets or social media links is a strong indicator of an article's class.**

Furthermore, politically charged terms like **"gop"**, **"president donald trump"**, and **"president barack"** are also highly ranked, suggesting the model is keying on political polarization. The high importance of informal or conversational words like **"just"**, **"said"**, and **"like"** points to the model identifying a stylistic difference between formal news reporting and the more opinionated or conversational tone often found in fake news.

**3. Conclusion:**

This project demonstrated that a hybrid modeling approach combining metadata and text features can be highly effective for fake news detection, with the **XGBoost classifier proving to be the most successful model**, achieving a final test F1-score of 0.846.

However, the key insight from this analysis is not just the final score, but the **mechanism by which the model achieves it.** The model's performance is heavily influenced by overfitting, and the feature importance analysis reveals that it relies more on identifying **structural and stylistic shortcuts** than on a deep semantic understanding of the news content. It effectively learns to be a "format detector"—identifying embedded tweets, political polarization, and informal language—which are strong characteristics of the fake news within this specific dataset.

While this strategy is effective here, it highlights a potential limitation in generalizability. The model's success is tied to the specific style of the fake news articles it was trained on, and it may be less effective against different formats of misinformation that do not share these same stylistic artifacts.

**4. Future Work:**

Try other vectorize methods using packages like spacy, genism, sentence\_transformer etc.

The following is some discussion about vectorizing with spacy and genism separately.

First, spacy method:

nlp = spacy.load('en\_core\_web\_md')

spacy\_X\_train = X\_train.apply(lambda x: nlp(x).vector)

spacy\_X\_test = X\_test.apply(lambda x: nlp(x).vector)

Use np.vstack() change to 2d arrays with 300 vectors:

spacy\_X\_train = np.vstack(spacy\_X\_train)

spacy\_X\_test = np.vstack(spacy\_X\_test)

After getting spacy\_X\_train and spacy\_X\_test, can modeling with LogisticRegression.

Possible problem: after load 'en\_core\_web\_md', the model was pre-trained, which may not be suitable for the specific project, that means it may not perform well.

Second, word2vec method:

tokens = word\_tokenize(text)

sentences\_train = [[‘token1’, ‘token2’], [‘token1’, ‘token2’, ‘token3’]……]

model\_wv = Word2Vec(sentences\_train, vector\_size=300)

After getting the trained word2vec model with 300 vectors, map the tokens in train set and test set using it.

vectorized\_text = [model\_wv.wv[word] for word in text if word in model\_wv.wv)]

Then, average along the rows for each news to get an averaged vector to represent each document. Finally use np.vstack() change to 2d arrays with 300 vectors, which can be used for modeling with LogisticRegression.

Since word2vec using custom trained model, it may perform better than spacy’s pre-trained model.

Third, doc2vec method:

tagged\_data\_train = [TaggedDocument(words=doc)]

model\_dv = Doc2Vec(tagged\_data\_train, vector\_size=300)

After getting the trained doc2vec model with 300 vectors, can use the infer\_vector () method on tagged test data to get vectors for test set, which can be used for modeling with LogisticRegression.

doc\_vectors\_test = [model\_dv.infer\_vector(tagged\_data.words) for tagged\_data in tagged\_data\_test]

Doc2Vec learns to predict words in a similar manner to Word2Vec but includes a unique vector for each document.

Fourth, BERT method:

model = SentenceTransformer("all-MiniLM-L6-v2")

model.encode(X\_train.tolist())

model.encode(X\_test.tolist())

First load pre-trained model and then use model.encode() to get the encoded sentences.

This is an easy method to use BERT to encode sentences, but the users have less control over underlying architecture and tokenization process. There is one more complicated method to get BERT embeddings using BERT model and tokenizer which give users more control over tokenization process, which can be found in the following links about this method and its application with pytorch for fake news detection.

[Detecting Fake News — with a BERT Model | by Skillcate AI | Medium](https://medium.com/@skillcate/detecting-fake-news-with-a-bert-model-9c666e3cdd9b)

All these four methods work differently as CountVectorizer and TfidfVectorizer. In CountVectorizer and TfidfVectorizer, each vector corresponds to a unique word in the vocabulary. Therefore, interpreting the logistic regression coefficients is straightforward. However, with spacy, word2vec, doc2vec and BERT embeddings, each vector captures complex relationships and context-dependent information from the text. Interpreting the coefficients becomes more challenging because each dimension does not directly correspond to a word.