**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. Two methods were applied for modeling: Method\_1 using relevant features with different classification models, like logistic regression, random forest classification, gradient boost classification and xgboost classification. Method\_2 vectorizing the news text content and modeling with Logistic Regression.The performance of each model is evaluated using metrics such as accuracy, precision score and recall score.

**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

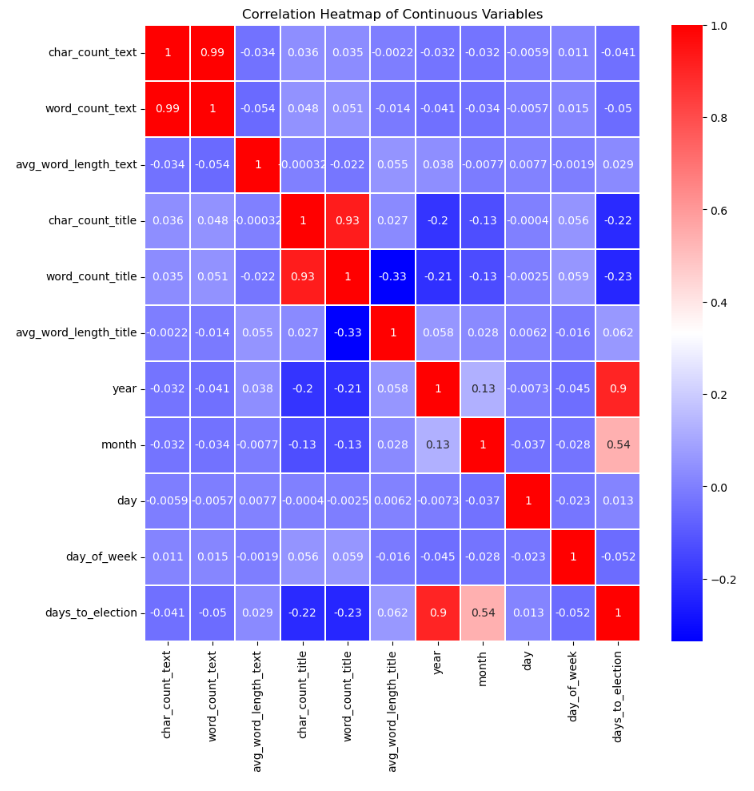
Description automatically generatedA graph of a person's data

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:

Method\_1: use features new created including character count, word count, average word length in title and text, year, month, day, day of week and days to election.

Models: Logistic Regression, RandomForest Classifier, Gradient Boost Classifier and XGBoost Classifier.

This method may not be generalized well because the features used may change for unseen dataset, and the date information may also change. So, the patterns found with this method may not be generalized.

Method\_2: use only the text column and vectorize with CountVectorizer() or TfidfVectorizer() with different parameters like ngram\_range, max\_df and min\_df.

Models: Logistic Regression.

This method captures semantic relationships between words for fake news and real news separately, which makes it better for generalization.

* 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:**

Tabular Comparison for Method\_1:



From the above table:

All these four experiments using Method\_1 have similar results on validation set, accuracy of train set range from 0.89 to 0.919, accuracy of validation set range from 0.897 to 0.924, but on test set, the accuracy range from 0.60 to 0.76, much lower than train set and validation set, which means these models are overfitted.

Tabular Comparison for Method\_2:



From the above table:

Exp\_5, use CountVectorizer(ngram\_range=(1,1)), get 115418 vectors and very high accuracy, 0.995 and 0.993 for both train set, and validation set, the test accuracy is little lower as 0.88, but precision is 1, which means all fake news are correctly labeled. Test recall is 0.769, which means 3 real news are labeled as fake news.

After checked the high coefficient terms, found there are some words like reuters, washington play an important role in the real news determination, and words like via, com play an important role in the fake news determination decided to delete them since they may not be generalized.

Exp\_6, after deleted high coefficients terms, use CountVectorizer(ngram\_range=(1,1)), get little lower accuracy, 0.974 and 0.973 for both train set and validation set, there is 109545 vectors, but the test accuracy 0.88 keeps the same, this makes sense because we already deleted news sources and notes in test set, which means there is no words like reuters, Washington, via, com etc. Test precision still 1, which means all fake news are correctly labeled. Test recall is 0.769, which means 3 real news are labeled as fake news.

Exp\_7, use CountVectorizer(ngram\_range=(1,1), max\_df=0.8) get 109537 vectors , accuracy 0.972 and 0.970 for train set and validation set separately, test accuracy increased to 0.92. Test precision still 1, which means all fake news are correctly labeled. Test recall is 0.846, which means 2 real news are labeled as fake news.

Exp\_8, use CountVectorizer(ngram\_range=(1,1), min\_df=0.1) get 300 vectors, accuracy 0.952 and 0.953 for train set and validation set separately, test accuracy 0.88. Test precision 0.857, which means 2 fake news are labeled as real news. Test recall is 0.923, which means 1 real news are labeled as fake news.

Exp\_9, use CountVectorizer(ngram\_range=(1,1), max\_df=0.8, min\_df=0.1) get 292 vectors, accuracy 0.947 and 0.944 for train set and validation set separately, test accuracy 0.92. Test precision 0.867, which means 2 fake news are labeled as real news. Test recall is 1, which means all real news are labeled correctly. Highest f1 score, 0.929.

Exp\_10, use CountVectorizer(ngram\_range=(2,2), max\_df=0.8, min\_df=0.1) get 91 vectors, accuracy 0.901 and 0.895 for train set and validation set separately, test accuracy only 0.60. Test precision 0.60, which means 6 fake news are labeled as real news. Test recall is 0.692, which means 4 real news are labeled as fake news. This is test performs worst.

Exp\_11, use CountVectorizer(ngram\_range=(1,2), max\_df=0.8, min\_df=0.1) get 383 vectors, accuracy 0.96 and 0.958 for train set and validation set separately, test accuracy 0.84. Test precision 0.846, which means 2 fake news are labeled as real news. Test recall is 0.846, which means 2 real news are labeled as fake news.

Exp\_12, use CountVectorizer(ngram\_range=(1,2), max\_df=0.8, min\_df=0.2) get 143 vectors, accuracy 0.932 and 0.932 for train set and validation set separately, test accuracy 0.80. Test precision 0.833, which means 2 fake news are labeled as real news. Test recall is 0.769, which means 3 real news are labeled as fake news.

Exp\_13, use TfidfVectorizer(ngram\_range=(1,2), max\_df=0.8, min\_df=0.1) get 383 vectors, accuracy 0.96 and 0.958 for train set and validation set separately, test accuracy 0.84. Test precision 0.846, which means 2 fake news are labeled as real news. Test recall is 0.846, which means 2 real news are labeled as fake news. Same metrics compared with Exp\_11.

Exp\_14, use TfidfVectorizer(ngram\_range=(1,1), max\_df=0.8, min\_df=0.1) get 292 vectors, accuracy 0.951 and 0.944 for train set and validation set separately, test accuracy 0.92. Test precision 0.923, which means 1 fake news are labeled as real news. Test recall is 0.923, which means 1 real news are labeled as fake news. Second highest f1 score, 0.923, just little lower than Test\_9.

Exp\_15, use TfidfVectorizer(ngram\_range=(1,1), min\_df=0.1) get 300 vectors, accuracy 0.956 and 0.952 for train set and validation set separately, test accuracy 0.88. Test precision 0.917, which means 1 fake news are labeled as real news. Test recall is 0.846, which means 2 real news are labeled as fake news.

Exp\_16, use TfidfVectorizer(ngram\_range=(1,1), max\_df=0.8) get 109537 vectors, accuracy 0.972 and 0.971 for train set and validation set separately, test accuracy 0.92. Test precision 1, which means no fake news are labeled as real news. Test recall is 0.846, which means 2 real news are labeled as fake news, f1 score 0.917.

Exp\_17, use TfidfVectorizer(ngram\_range=(1,1)) get 109545 vectors, accuracy 0.973 and 0.971 for train set and validation set separately, test accuracy 0.92. Test precision 1, which means no fake news are labeled as real news. Test recall is 0.846, which means 2 real news are labeled as fake news. Same metrics as Exp\_16.

**3. Conclusion:**

Two modeling methods were used for this project. Method\_2 performs better than Method\_1 overall, which makes sense, means natural language processing method can be generalized better.

With Method\_1, the four models perform similar on train set. On unseen test set, the XGBoost Classifier performs best with accuracy 0.76, precision score 0.889, recall score 0.615, all higher than others 3 models. Compared with metrics of validation set, these models using Method\_1 are overfitting seriously.

With Method\_2, only tested Logistic Regression model with different parameters in CountVectorizer and TfidfVectorizer.

Exp\_9 and Exp\_14 using CountVectorizer and TfidfVectorizer with same parameters. Get f1 score as 0.929 and 0.923 separately, both using only 292 vectors. But Exp\_9 has lower precision score 0.867, Exp\_14 is more balanced, test accuracy 0.923 and test precision 0.923. TfidfVectorizer works better?

Exp\_7, Exp\_16 using CountVectorizer and TfidfVectorizer with same parameters. Get same test accuracy score as 0.92, these two tests all get test precision score 1, which means all fake news are correctly labelled.

Exp\_8 and Exp\_15 using CountVectorizer and TfidfVectorizer with same parameters. Their results are similar, CountVectorizer even better?

Exp\_5 and Exp\_6 using CountVectorizer before and after deleting high coefficient terms, their results are same, which mean deleting high coefficient terms makes no difference on test set.

But these two are all not as good as Exp\_17, which use TfidfVectorizer with same parameters. TfidfVectorizer works better?

Exp\_11 and Exp\_13 using same parameters CountVectorizer and TfidfVectorizer separately are two models get moderate results. The results of these two are same, which means using same parameters, these two vectorize methods performs similar for test set.

Based on these findings, it looks like more vectors model gets higher precision score and less vectors model gets higher recall score. Although these models using Method\_2 also overfitting comparing the validation metrics and test metrics, but much less overfitted, especially Exp\_9 and Exp\_14, almost no overfitting. It is very difficult to decide better one from CountVectorizer and TfidfVectorizer.

For the high coefficient vectors comparison, the top 6 vectors for determining real news are: Wednesday, Thursday, Tuesday, Friday, Monday, said.

The top 10 vectors for determining fake news are: just, this, these, even, Hillary, us.

Using CountVectorizer and TfidfVectorizer with different parameters didn’t change these vectors much.

**4. Future Work:**

Try other vectorize methods using packages like spacy, genism 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.