GENERAL ASSEMBLY

DSI - 7 SF

PROJECT FOUR: DISASTER CLASSIFICATION
GROUP: Bill Yu, Evelyn Li, Kun Guo, Manu Kalia

PROBLEM STATEMENT

Can a classification model successfully classify news article posts into one of two categories… DISASTER or NOT-DISASTER? If so, which estimator performs best in terms of accuracy and compute resources?

DATA COLLECTION

1. The number of news articles in the training/test dataset was about 40,000. We collected data utilizing the News API. We created and deployed a function to collect 250 articles at a time, each download based on a keyword from our own list. Finally, we identified 5,551 articles related to disasters.

2. Next, we manually reviewed and labeled disaster events (labeled as 1) and non-disaster events (labeled as 0). This is the key element to implementing supervised machine learning. It is the only way for models to "learn" which words are relevant to the classification problem. Reading the titles of 40,000 articles and classifying as disaster or not-disaster took about 56 person-hours, and was the limiting factor determining the size of the train/test dataset.

DATA INSPECTION and EDA

Missing values

Since there were only a small number of rows with missing values in the Title, Content, and Description fields, which is significantly fewer than the total number of rows (11,110), filled them with a blank space “ ”.

Decided to include the 'Content' column in the NLP classification process.

We dropped the non-relevant columns such as URL, author, etc.

Are yes_disasters classes *balanced* (1's vs. 0's)? To ensure balanced classes, we randomly selected yes_disaster=0 rows equal in number to yes_disaster=0 (5,551 of each).

Count vectorization

In order to check the frequencies of the top words, we built a function to vectorize the 'Words' column into numbers, counting how many times each word appeared in the Words column, and then sorted by frequency.

- Most frequently-appearing words in yes_disaster=1
- Most frequently-appearing words in yes_disaster=0
- With and without stopwords

Removed all special characters such as “,”, numbers, “<br>”, “$”... etc., then applied a customized stopwords list which includes location words, country names, regions, unique words, city names...etc. to the count, vectorizer to be able to filter unrelated words from showing up in visualizations.

Word frequency bar charts

- Top 20, 50, 100, 200 words

Most common words, overlap (in words and in percentages)

- Word appearing in *both* lists (yes_disaster = 1 and 0) of top 20-50-100-200 counts
- 52.6% (top 20 words), 36.7% (top 50), 45.5% (top 100), 56.8% (top 200)
- The overlap rates are relatively high, indicating that classification methods may have a difficult time distinguishing between a disaster condition and a non-disaster condition.

CUSTOM STOP WORDS

Employed English stop words, with additional location words added to prevent the models from training on locations instead of disaster-related words. Training the classification models on location words would not generalize well to future disasters, but rather to any future news that occurs in locations at which disasters occurred in the training set (e.g. news occurrences that occur in Paradise, CA the site of one of 2018's wildfire disasters.

2,490 total stopwords:

- 318 starting stopwords list
- 193 non-US countries
- 1,000 US cities
- 51 US states + D.C.
- 928 Tsunami, river-flood, storm-surge, and earthquake-prone regions

CLASSIFICATION MODELS

Pre-processing:

1. Count Vectorizer
2. TFIDF (min, max, n-grams, and stopwords)
3. Doc2Vec

To train the supervised machine learning models, our group defined a list of 178 keywords pretended to natural disasters. We generated 40,000 news articles via the News API. After the selection process, we manually reviewed these articles and labeled the disaster event as 1 and non-disaster events as 0 for the target value. By screening each article for 5-10 seconds, our group dedicated approximate 56 labor hours to review all 40,000 articles. After dropping duplicates and balancing the classes, a total of 11,100 articles were remained for our training material to classify news that are either disaster or non-disaster event.

We explored a combination of text transformation techniques and feature engineering to achieve our modeling goal as below:

1. Text transformation techniques: CountVectorizer, and TFIDF, and Doc2vec
2. Feature engineering: Content only, Content + Description feature
3. Model parameters: Using the customized stop words, n-grams: (1,2) and (1,3), TFIDF with min, max.

For round 1 model evaluation, we implemented a total of nine regression and classification model such as Logistic Regression, KNN, and Gradient Boosting. We then compared and evaluate accuracy, recall, and train-test score among these models. TFIDF provided a relatively higher score than Count Vectorizer and Doc2vec after running models. Sequentially, Logistic Regression and Naive Bayes were chosen as our final models due to the consideration of computing power, fit time, interpretability, and overall performance. We consider that a large volume of news needs to be classified when the model is deployed. Therefore, Logistic Regression and Naive Bayes could handle classification without sacrificing computation resources and minimizing accuracy.

As for the feature engineering in round 2, we decided to use the 'Content' column only, instead of combining Content + Description + Title columns. Although blending three columns would provide a higher score than content columns. We deduced that the overlapping text content would bring more noise, creating an overfitted model. Therefore, we decided to use Content as our X input.

For a final round of tuning, to prevent our model training on location names instead of disaster-relevant words, we included a customized the list of stop words containing names of over 2,000 countries, cities, states, and regions.

Doc2vec contextual vectorization:

Since the information we are analyzing is news articles, there is an underlying assumption we had that the words within the articles relating to disasters have some relationship with each other. For example, `California wildfire` is a two-word phrase that we should not assume independence between words since these two words are pointing to a specific event. With this assumption in mind, we decided to use Doc2vec as one of the text vectorization methods to transform data.

So how does Doc2vec pick up the relationship between words within an article? Doc2vec studies the relationship between words and the article. First, the algorithm assumes all words are independent of one another. To teach the algorithm about the relationship between words and documents, we first need to build a dictionary for the model with tags indicating whether an article is relating to disaster or not.

Doc2vec picks up the relationship between words using continuous bag of words which creates a sliding window around the current word, to predict it from “context” — the surrounding words. Each word is represented as a feature vector with 100-dimension. This 100-dimension vector comes from learning words in each article that we label as 1(disaster-related articles). At the end of the training, each article will now contain 100-dimension of vectors that represent each word and its relationship with other words. From there, we use the dictionary to transform our train and test data and train the model. Doc2vec now then take the information learned from the train data to estimate whether or not the test data is relating to disaster events based on article and words position in space.

Although the idea of Doc2vec sounds promising, the result of the prediction with the tuned model is not significantly higher than using Count Vectorizer or TFIDF. Here are a couple of reasons that we have concluded regarding the context of the article. First, the structure of the text does not matter that much. Since all we are collecting are news articles, they all follow similar structures and uses similar words. Therefore, understanding the structure of the text does not give the model an advantage over other vectorization methods. Secondly, what we are interested in is disaster news. Usually, disaster news has clear indicators such as words 'disaster', 'fire,' 'flood' and so on, as long as the model can pick up these keywords, classification model might not need complicated indications.

Model testing:

After establishing the model, we must understand how the model is performing with new data. To exam the model, we query two sets of new data from the NewsApi with two keywords. One is the word 'flood' which is relevant to the disaster news, and the other one is the word 'today.' From the two keywords, we expect the model to classify news article related to disaster with a 78% accuracy rate. After we feed the newly collect data into the model and collect the prediction, we manually exam the result to understand how well the model has performed.

From looking at the result from the data collected using the keyword 'flood,' the accuracy rate was about 60%. We understand the model needs improvement since it is still having some difficulty classifying borderline articles such as 'how to help people during a disaster.' However, when examining the result from the data collected using the keyword 'today,' which contains no articles about disasters that are currently happening, the model picks up six pieces that are semi-related to disaster. From evaluating both results, we can conclude that the model is performing well at ignoring non-relevant articles. However, the model still has difficulty distinguishing between articles that talk about disasters in general, versus articles that are reporting an unfolding disaster event.

Classification estimators: 9 different models deployed, 3 vectorization methods, 2 sets of features (Content and Content+Description)

Hyperparameter Tuning:

1. Stop words and customized stop words (remove locations names, like countries, regions, states, cities)
2. Tokenize and Lemmatize
3. Features (content only, and content + description)
4. Grid Search

Final model selection

1. TFIDF + custom location stop words + Logistic Regression
2. Pull out key parameters (top feature words & coefficients)
3. Compare the coefficients and feature name before and after the customized stop words

RECOMMENDATIONS (using the project output)

1. Rank-ordered feature words: final model creates a set of important words and their associated coefficients, the client can apply the top words to classify all ingested news articles, and display "disaster=yes" items on a webpage.
   A. Keywords to use: top coefficient words from the final logistic regression model

   B. Set up a scheduled script (as in CRON) to ingest all news posts (and/or tweets) 1-4 times per day. Use this downloaded set of articles as a new 'test' dataset for our trained and fitted model.

   C. Display all the news items that are returned as positive for the disaster class. There will be some false positives that are posts containing disaster words but are not necessarily about a disaster in progress.

2. Rerun the model & update the key features list. Periodically (2-3 times per year) use the top 50 or so key features to download more articles for training the classifier(s). The update cycle will collect news articles, and check for duplicates to articles in the existing training dataset. Re-run the Naive-Bayes and Logistic Regression models to keep the keywords set updated.

3. Continuously grow the classified training dataset. Collect large new sets of articles and characterize each one as "disaster" or not-disaster" by human inspection. This is a time and labor-intensive task, so it is important to minimize the labor expense whenever possible...

a. Local contests
b. Internships
c. Disaster assistance volunteer days

4. Continuously grow the dataset also by flagging all news articles on known disaster days, and store for future incorporation. The articles on the day of a disaster (and perhaps for a few days after) will have a much higher number of 'disaster=yes' articles, which are very useful for the training set.

5. Future model enhancement
   a. Secondarily classify the disaster types (wildfires, storms, floods, etc.)
   b. Identify a "Disaster Condition" using the number of articles classified as 'yes_disaster' during a certain time-period (perhaps one day or several hours)