ds3 reviews

April 20, 2019

1 Google Local - Negative Sentiment Analysis

1.0.1 1) Problem Statement

We wanted to tackle what the general problems that businesses should strive to solve in order to prevent negative reviews.

1.0.2 2) Initial data analysis and findings

We used a sample of ~500,000 reviews. Our initial data analysis looked into problems regarding the different languages in the review data and how tf-idf can be used to create accurate models. We also experimented with what size of n-gram would produce the most accurate model. We thought using larger sized n-grams as well as stop words would increase the accuracy of our model, however it turned out it did not. 1-3 words were the best way to split a review text and including the stop words also increased the accuracy. It is hard to see what effect the foreign languages had on our analysis because we were forced to leave many in the text, but the model was still able to perform reasonably well. We also tried other models such as multinomial naive bayes, and decision tree classifiers, but linear SVC performed the best.

1.0.3 3) Hypothesis testing and results

Hypothesis: Reviews will have words that reflect the rating.

To test this, we first located the text reviews that were 3-stars or lower and analyzed if the reviews were reflective of the rating. We created a classifier based on a linear SVC model and n-gram features to predict whether a review would be negative or not which could achieve around 88% accuracy on test data. This led us to conclude that the reviews did contain information that were relevant to the negative rating.

1.0.4 4) Modelling / Summarizing insights

With this knowledge, we took the negative reviews and created text summaries of length 4 to 5 words based on TF-IDF for each review. We then took these summaries to create a word cloud of all negative summaries. The most relevant words were "food", "minute", and "service." From this, we can see that most issues stem from these general themes of food, time, and service.

Members: Derrick Liu, Pete Sheurpukdi

```
In [1]: import sklearn
        import pandas as pd
        from collections import defaultdict
        import numpy as np
        from sklearn.pipeline import Pipeline
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.svm import LinearSVC
        from sklearn.model_selection import train_test_split
        import matplotlib
        import matplotlib.pyplot as plt
        from IPython.display import Image
        %matplotlib inline

In [2]: data = pd.read_json('sample.reviews.json')

In [3]: def create_classifier_binary(X, y):
            """
            """

            from sklearn.feature_extraction.text import TfidfVectorizer
            from sklearn.svm import LinearSVC
            from sklearn.tree import DecisionTreeClassifier
            from sklearn.naive_bayes import MultinomialNB
            from sklearn.model_selection import train_test_split

            classifier = Pipeline([
                ('tfidf', TfidfVectorizer(ngram_range=(1,3))), #including stop words because i
                ('lin', LinearSVC())
            ])

            classifier.fit(X, y)

            return classifier

In [5]: import string
        df = pd.DataFrame(data)

        df = df.dropna(subset=['reviewText', 'rating']).sample(100000)
        df['rating'] = df['rating'].apply(lambda x: 1 if x in [4,5] else 0)

        df['reviewText'] = df['reviewText'].apply(lambda x: str.lower(x).strip().translate(str
        X_train, X_test, y_train, y_test = train_test_split(df['reviewText'], df['rating'], tes
        pl = create_classifier_binary(X_train, y_train)
        pl

        pl.score(X_test, y_test) #validate, see if accuracy is reasonable.

Out[5]: 0.86825

In [6]: df['predicted sentiment'] = pl.predict(df['reviewText'])
        df['predicted sentiment'].head()
```

```
Out[6]: 241338    0
        326769    0
        287719    1
        441866    1
        268857    1
        Name: predicted sentiment, dtype: int64
```

```
In [ ]: # get only the negative reviews in the dataframe
        sentiment = df

        d = sentiment.dropna(subset=['reviewText', 'rating'])
        # d['rating'] = d['rating'].apply(lambda x: 1 if x in [4,5] else 0)
        # d['reviewText'] = d['reviewText'].apply(lambda x: str.lower(x).strip().translate(str

        vectorizer = TfidfVectorizer(ngram_range=(4,5),stop_words='english')

        X = vectorizer.fit_transform(d['reviewText'])

        m = {v: k for (k, v) in vectorizer.vocabulary_.items()}

        ngram_summary = [m[t] for t in np.array(np.argmax(X, axis=1)).flatten()]

        d['ngram summary'] = ngram_summary

        d = d.reset_index()

        for i in range(len(d)):
            if len(d.loc[i, 'reviewText'].split()) < 4:
                d.loc[i, 'ngram summary'] = d.loc[i, 'reviewText']
```

The table below gives the negative reviews along with the generated n-gram tf-idf summary.

```
In [ ]: d[['rating', 'reviewText', 'predicted sentiment', 'ngram summary']]
```

We changed machines, and the second didn't have wordcloud installed so we just uploaded the original image.

```
In [ ]: # from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
        # import langdetect
        # df = d.copy()

        # df = df.dropna(subset=['reviewText', 'rating'])

        # df = df.loc[df['predicted sentiment'].astype(int) == 0]#['reviewText']

        # text = " ".join(review for review in df['ngram summary'])

        # wordcloud = WordCloud().generate(text)
        # # Display the generated image:
        # plt.imshow(wordcloud, interpolation='bilinear')
        # plt.title('Word Cloud of Sampled Reviews Classified as Negative')
        # plt.show()

        from IPython.display import Image
        Image(filename='wordcloud.png')
```

```
In [ ]: #Language filtering TAKES TOO LONG TO RUN, SO WE WONT DO IT

        # from polyglot.detect import Detector
        # df = pd.DataFrame(data)

        # df = df.dropna(subset=['reviewText', 'rating'])

        # df['reviewText'] = df['reviewText'].apply(lambda x: str.lower(x).translate(str.maket

        # df['reviewText'] = df['reviewText'].apply(lambda x: bytes(x, 'utf-8').decode('utf-8'
        # df = df.loc[df['reviewText'] != ''].reset_index()


        # df['lang'] = df['reviewText'].apply(lambda x: Detector(x, quiet=True).languages[0].c

        # #df1.loc[df1['Language'] == 'en']
```