CS 506 Midterm Writeup

Natalie Cheng

Preprocessing

The data we are dealing with includes text that we need to analyze to help determine a final score. Therefore, I followed Lecture 8 to implement TF-IDF and Word Embeddings, which provide insight into the significance of words in documents. I initially just implemented TF-IDF, which resulted in low accuracy, leading me to additionally implement word embedding to capture the semantic meaning and context of words.

```
# TF-IDF
print("Startng TF-IDF")
vectorizer = TfidfVectorizer(stop_words='english', min_df=4,max_df=0.8)
tfidf_matrix = vectorizer.fit_transform(df['ProcessedText'])
normalized_tfidf = tfidf_matrix / tfidf_matrix.max()
print("Finished TF-IDF")

# centered_dtm = tfidf_matrix - np.mean(tfidf_matrix, axis=0)
# u, s, vt = np.linalg.svd(centered_dtm.toarray(), full_matrices=False)

# Weighted embeddings
print("Starting embeddings")
embeddings = []
for i, text in enumerate(df['ProcessedText']):
    tfidf_vector = normalized_tfidf[i]
    weighted_embedding = get_weighted_embedding(text, tfidf_vector, vectorizer, word_vectors)
    embeddings.append(weighted_embedding)
df['WeightedEmbedding'] = embeddings
print("Completed Embeddings")
```

Just TF-IDF/Word Embeddings, however, still resulted in a low accuracy, which led me to research sentiment analysis and discover the Textblob library (linked below). This library was not discussed in class, however I understand that it is a sentiment analyzer that provides insight on polarity and subjectivity of given text, which would help our models further understand the context and opinions of the "Text" and "Summary" columns. Textblob does not use neural networks/deep learning, and is built on traditional NLP techniques, such as tokenization and simple classification based on Naive Bayes. Some additional research led me to WordNetLemmatizer and RegexpTokenizer, which uses lemmatization instead of stemming, and advanced tokenization. Lemmatization is more effective than stemming because rather than just removing common suffixes, lemmatization gets the root word, allowing the model to understand the content more effectively.

https://www.analyticsvidhya.com/blog/2021/01/sentiment-analysis-vader-or-textblob/

https://neptune.ai/blog/sentiment-analysis-python-textblob-vs-vader-vs-flair

https://www.ibm.com/topics/stemming-lemmatization#:~:text=The%20practical%20distinction%20between%20stemming,be%20found%20in%20the%20dictionary.

```
# Preprocess Text
print("Pre-processing Text")
df['Text'] = df['Text'].fillna('')
df['Summary'] = df['Summary'].fillna('')
df['CombinedText'] = df['Text'] + " " + df['Summary']
df['ProcessedText'] = df['CombinedText'].apply(preprocess_text)
```

I combined the text and summary columns so they could be analyzed together for each review. In hindsight, it may have been better to keep the text and summary columns separated because it might help the sentiment analysis model to understand nuances in the text better, and maybe there are differences in the text vs. summary.

```
# Add sentiment features
df['SentimentPolarity'], df['SentimentSubjectivity'] = zip(*df['ProcessedText'].apply(sentiment_analysis))
```

Final Algorithm

I ran two different classification algorithms- K Nearest Neighbors and Random Forest. After running GridSearchCV to find the best model, I discovered that Random Forest was more accurate than KNN.

Best parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 200}

Best cross-validated score: 0.48915555555555557

Best parameters for KNN: {'algorithm': 'auto', 'n_neighbors': 200, 'weights': 'uniform'}

Best cross-validated score: 0.4669333333333333

Accuracy on testing set = 0.47186666666666666

This score can be improved. I decided to increase the old parameters of Random Forest and KNN since 200 seems to be the current most optimal parameter:

RF: [50, 100, 200] to [100, 200, 300]

KNN: [10, 50, 100, 200] to [100, 200, 300]

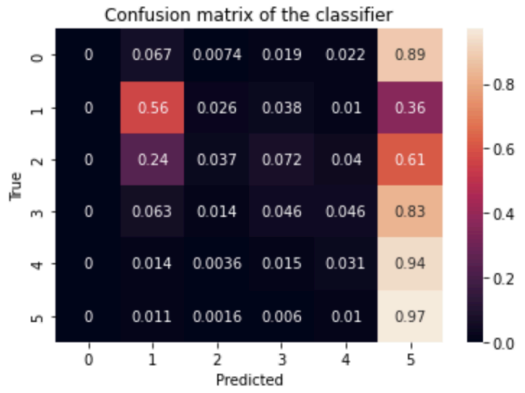
Results:

Best parameters for Random Forest: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}

Best cross-validated score: 0.4893333333333333

After implementing some more changes such as lemmatization and advanced tokenization in the preprocessing, and running the models on the entire dataset instead of a small training subset, my final accuracy is:

Accuracy on testing set = 0.4960860918413512



Looking at the confusion matrix, it seems that the model confuses certain classes and leads to misclassification in the higher classes. Some potential fixes to improve the

accuracy could be to try balancing the classes because it seems like the higher classes are more represented.

Patterns

Working on this project, I noticed that as I decreased nuance in the data (such as lemmatization and stemming the words), the models improved in accuracy. Removing nuances in words allows the models to focus more on meaning of words rather than grammatical differences and trying to differentiate from them. Additionally, after adding methods that allowed more emphasis on meaning, such as weighted embeddings in TF-IDF and sentiment analysis like polarity and subjectivity, my accuracy improved. This shows the importance of determining the most important words that convey sentiment accurately and the importance of cleaning text properly.

Improvement

Ways I could have improved my accuracy could have been adding more/handling the features better. Apparently time was an important factor, as over time the scores would improve so taking a look at that feature could have been helpful in improving. Instead of having time as a single integer, they could have been converted into years/months. Additionally, trying out other sentiment analysis techniques or new models such as Logistic Regression or trying methods to improve Random Forest could have improved my scores. Another way to understand the data better could have been to visualize the data, for example using graphs or charts to see trends in the data which would give me better insight on how to handle the features. The confusion matrix was a helpful tool for visualization but some attempts I made to try to improve upon misclassifications as stated above ended up decreasing my accuracy, but maybe I had changed other things along the way that could have impacted the overall accuracy. Due to the time constraint of the code taking a long time to run, I was not able to get to all of the methods I could have improved upon.