Sentiment Analysis

Myriam Kapon - aid24008 22/04/24

Approach

I prioritized developing straightforward code to facilitate the implementation of various options for sentiment analysis. This included offering different preprocessing techniques, the flexibility to choose vector models and parameters for fine-tuning, and the option to select different classifiers. The objective was to provide a method that allows users to effortlessly experiment with different combinations, enabling them to identify the optimal choice and analyze and interpret the results effectively.

Data Exploration

Before starting the analysis, some information about the dataset was collected to see if the classes are balanced.

Number of positive reviews	1000
Number of negative reviews	1000
Total words	1583820
Number of words in pos	832564
Number of words in neg	751256
Most popular words in pos	(',', 42448), ('the', 41471), ('.', 33714)
Most popular words in neg	(',', 35269), ('the', 35058), ('.', 32162)

The dataset is balanced with positive reviews having a bit more words. Looking at the most popular words, it's clear that some word preprocessing must be done.

Preprocessing

The preprocessing pipeline was designed to provide flexibility in enabling or disabling different preprocessing steps easily. Here are the smaller functions implemented as part of the pipeline:

- **Tokenization** (default): Splits the review into individual tokens.
- **Lowercasing**: Converts all characters to lowercase to ensure consistency.
- **Removal of punctuation**: Important for removing common punctuation marks like commas and periods.
- **Removal of non-alphabetic characters**: This step is more aggressive than simple punctuation removal, ensuring that only alphabetic characters remain.
- **Lemmatization**: Reduces words to their base or root form based on grammar rules. Works well with models like Word2Vec.

- **Stemming**: Reduces words to their root forms by removing prefixes and suffixes. However, it may produce non-standard or "butchered" words, making it less suitable for pretrained Word2Vec models.
- **Removal of stopwords**: Eliminates common words like "and", "the", and "a", which often carry little semantic meaning.
- **Removal of short words**: Filters out single-character words that may not contribute much to the overall meaning of the text.

Following a rule of thumb, more aggressive preprocessing techniques were applied to the model that was trained from scratch, while less aggressive preprocessing was used for the pretrained model, which already possessed an understanding of normal words and their variations.

Original Review
the happy bastard's quick movie review damn that y2k bug . it's got a head start in this movie starring jamie lee curtis and another baldwin brother (william this time) in a story regarding a crew of a tugboat that comes across a deserted russian tech ship that has a strangeness to it when they kick the power back on . little do they know the power within . . . going for the gore and bringing on a few action sequences here and there , virus still feels very empty , like a movie going for all flash and no substance . we don't know why the crew was really out in the middle of nowhere , we don't know the origin of what took over the ship (just that a big pink flashy thing hit the mir) , and , of course , we don't know why donald sutherland is stumbling around drunkenly throughout . here , it's just " hey , let's chase these people around with some robots " . the acting is below average , even from the likes of curtis . you're more likely to get a kick out of her work in halloween h20 . sutherland is wasted and baldwin , well , he's acting like a baldwin , of course . the real star here are stan winston's robot design , some schnazzy cgi , and the occasional good gore shot , like picking into someone's brain . so , if robots and body parts really turn you on , here's your movie . otherwise , it's pretty much a sunken ship of a movie .

Preprocessed Review
the happy bastard quick movie review damn that y2k bug got head start this movie starring jamie lee curtis and another baldwin brother william this time story regarding crew tugboat that come across deserted russian tech ship that strangeness when they kick the power back little they know the power within going for the gore and bringing few action sequence here and there virus still feel very empty like movie going for all flash and substance know why the crew really out the middle nowhere know the origin what took over the ship just that big pink flashy thing hit the mir and course know why donald sutherland stumbling around drunkenly throughout here just hey let chase these people around with some robot the acting below average even from the like curtis you more likely get kick out her work halloween h20 sutherland wasted and baldwin well acting like baldwin course the real star here are stan winston robot design some schnazzy cgi and the occasional good gore shot like picking into someone brain robot and body part really turn you here your movie otherwise pretty much sunken ship movie

Used Here: Lowercase, Remove Punctuation, Lemmatize, Remove Short Words (len =2)

Word Vectors

The **Word2Vec** model from the Gensim library was employed to convert words into vectors. This versatile tool allows users to either train a model from scratch or utilize a pre-existing one. Both methodologies were thoroughly examined and evaluated for their effectiveness.

However, using Word2Vec for creating word and document vectors presents a significant drawback in sentiment analysis. While it captures word meanings based on their co-occurrence in text, it overlooks semantic relationships. For instance, "good" and "bad" may be opposites, but Word2Vec fails to recognize this. Instead, it focuses solely on contextual usage, leading to the antonymy problem, where words with opposing meanings appear similar if they share contexts.

Custom Trained Model

The model was configured with a vector size of 256, a power of 2 chosen deliberately to circumvent potential bottlenecks. This selection aimed to encapsulate a sufficiently large dimensionality where distinctions between contrasting concepts like "good" and "bad" could potentially be discerned more effectively.

In pursuit of enhancing accuracy, several strategies were experimented with:

1. **Not-word:** This approach aimed to address negation by amalgamating words prefixed with "not" into a single token. For instance, "not good" would become "not-good". However, this technique couldn't be applied to the pretrained model, as it wouldn't recognize these merged tokens.
2. **Default Vector for Unknown:** To handle unknown words, a default vector of zeroes was assigned. While this strategy effectively dealt with unknown words, it inadvertently impacted the word averaging process, resulting in a decline in accuracy.
3. **Labels as Words:** In an effort to provide additional context, the labels "positive" and "negative" were appended after each sentence, with the aim of better contextualizing the words.

Unfortunately, none of the strategies yielded improved results. Nevertheless, documenting these endeavors serves as a source of inspiration for future iterations.

Word Similarity			
Words most similar to “**good**”		Words most similar to “**bad**”	
bad	0.8809	good	0.8809
nice	0.8603	funny	0.8283
funny	0.8308	nice	0.8037
great	0.823	stupid	0.8006

decent	0.7794
perfect	0.7729
quite	0.757
simply	0.7536
fine	0.74332
smart	0.7400

dumb	0.7770
quite	0.7690
simply	0.7675
scary	0.7643
cool	0.7602
boring	0.7507

It’s easy to see the antonimity problem here: “good” is most similar to “bad” and vice versa!

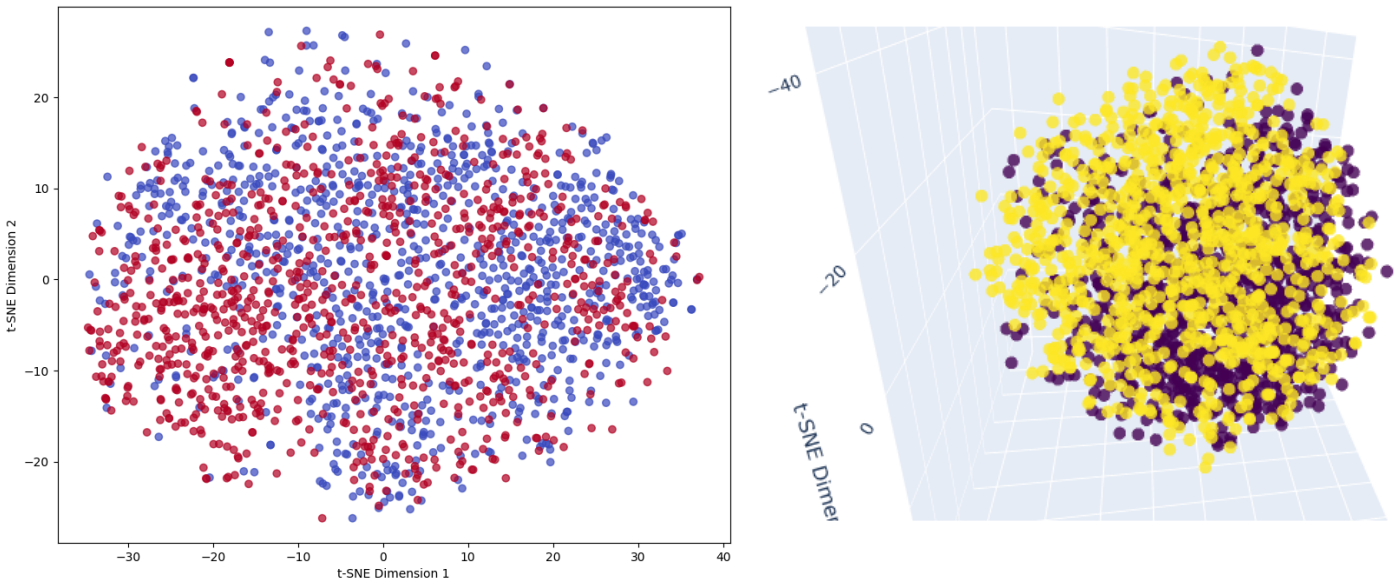
Pretrained Model

A model trained on Twitter data was obtained, considering its similarity to the conversational style often found in movie reviews. Surprisingly, this model didn't yield significantly better results compared to training a model from scratch.

Conversely, the "Sample" pretrained model, trained on the Google News Dataset, delivered remarkable performance, achieving an impressive accuracy of 83%. This outcome marked the highest accuracy observed throughout the entire analysis.

Vector Evaluation

To create visualizations of the word vectors, both 2D and 3D plots were generated using t-SNE (t-distributed Stochastic Neighbor Embedding). T-SNE is a dimensionality reduction technique used to visualize high-dimensional data in lower dimensions, typically 2D. It aims to preserve the local structure of the data points, making it particularly useful for visualizing relationships and clusters within the data.



Plots: t-SNE 2D (Left) and 3D (Right) Visualizations for Sample Trained Model.

When projected onto a two-dimensional space, the data appears mostly scrambled with some degree of separation to the left and right.However, in the 3D representation, there is separation noticeable on the y-axis.

Classification

The 7 classifiers tested were Linear Descriminant Analysis (LDA), Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors (kNN), Naive Bayes and Support Vector Machines (SVM).

To decide on the most suitable classifier, I revisited a Classifier Comparison framework developed together with Evgenia Argyriadi during a previous assignment in the "Machine Learning and Computer Vision" class with Professor Protopapadakis.With only minor adjustments to tailor it to the current assignment, I'm pleased with the opportunity to apply and build upon past work in this way.

Specifically, the code implements a framework for comparing the performance of different classifiers using various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. It conducts cross-validation, calculates these metrics for each fold, and aggregates the results. Additionally, it visualizes confusion matrices and identifies the best classifier based on a combined score derived from the evaluation metrics.

Normalization was ommitted to allow the vector values to remain far apart.

Results

This is the final setup that achieved the best results:

- **Preprocessing**: Lowercase, Remove Punctuation, Lemmatize, Remove Short Words (len =2).
- **Model**: Sample Pretrained Word2Vec Model.

Classifier	Accuracy	Precision	Recall	F1	Roc-Auc	Sensitivity	Specificity
Decision Tree	0.655504	0.651172	0.668087	0.658327	0.655552	0.668087	0.643017
LDA	0.837992	0.834672	0.842984	0.838797	0.837988	0.842984	0.832992
Logistic Regression	0.678000	0.682243	0.667011	0.674333	0.678000	0.667011	0.688988
Naive Bayes	0.685002	0.692934	0.665012	0.678525	0.684993	0.665012	0.704974
Random Forest	0.713499	0.720779	0.698000	0.709009	0.713484	0.698000	0.728968
SVM	0.668499	0.663818	0.688053	0.674433	0.668552	0.688053	0.649050
kNN	0.654517	0.656102	0.652014	0.653895	0.654526	0.652014	0.657037

Table 1: Classifier Comparison

Based on the average scores across 3 folds, **LDA** emerged as the top-performing classifier in terms of accuracy and F1 score. With a best accuracy and F1 score of 0.84, it meets the baseline accuracy range of 80% to 85%, which is typically considered on par with human analysts' accuracy.

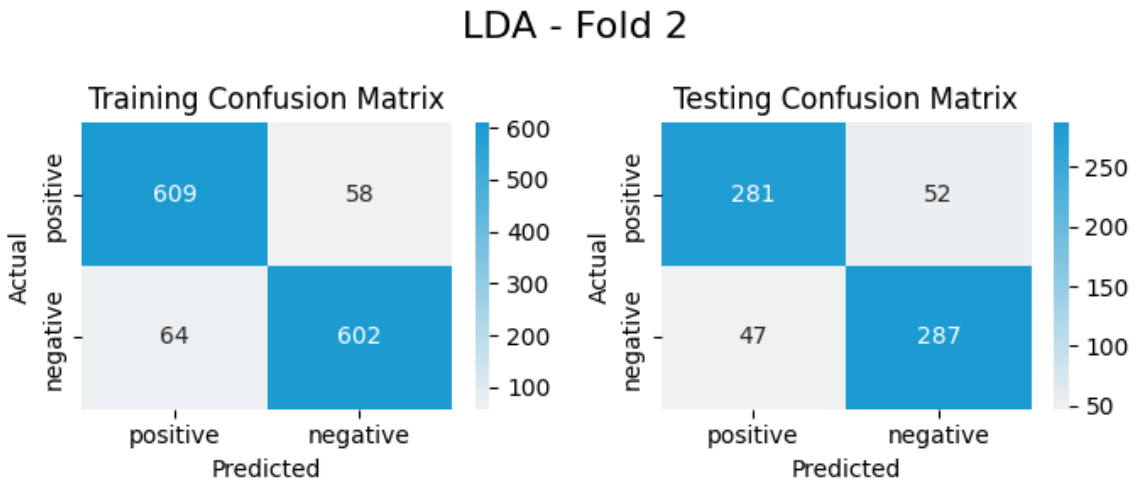


Figure 3: Confusion Matrix for LDA

Even when training a Word2Vec model from scratch, LDA still emerged as the top-performing classifier, albeit with lower accuracy and F1 scores of 0.72.

Thoughts

Preprocessing played a crucial role in the result's accuracy, but the most important part was using a pretrained model. Despite various attempts, it was challenging to prevent "good" and "bad" from being considered similar to each other due to the inherent limitations of word embeddings. There remains ample opportunity for experimentation and parameter tuning, such as exploring different preprocessing techniques, deciding whether to train the model or use pretrained ones, adjusting vector sizes, and fine-tuning classifier parameters. It's possible that there exists a more optimal combination of these factors that was not explored in this analysis.