Evaluation of Classifiers for Sentiment Analysis of Movie Reviews

Nancy, Ma
ym1581@nyu.edu

Abhilasha, Tandon
at4571@nyu.edu

Alex, Ding
zd727@nyu.edu

Abstract

In recent days, online movie reviews on websites like IMDB, Rotten Tomatoes, and Fandango play a significant role in influencing the public's perception of a film. From amateur bloggers to professional critics, the opinions shared in these reviews can make or break a movie's success. As a result, understanding the sentiment of movie reviews has become imperative for filmmakers, studios, and marketers to produce popular and relevant content. In this paper, we evaluated different approaches to analyzing a set of IMDB movie reviews, containing 50,000 reviews. We explored the different results produced by the following classifiers: random forest, nearest next neighbor, QDA, Naive Bayes, decision tree, and linear SVM. In addition to using a classifier for sentiment classification, we duplicated the analysis with the chosen classifiers and added word embeddings. Lastly, we also wanted to evaluate some other methods such as Latent Dirichlet Allocation, which is popular for sentiment analysis.

1 Introduction

Sentiment analysis is the process of using natural language processing and machine learning techniques to identify and extract subjective information from text. It is used for a wide array of applications in many sectors, such as social media content monitoring, brand engagement, and sentiment monitoring, survey response analysis, Voice of Customer methodology, customer service ticket organization, and more.

The main reason for sentiment analysis's popularity is its ability to automate the processing of huge quantities of data. Without sentiment analysis pipelines, it is often cost-prohibitive or impossible for organizations and companies to analyze thousands of surveys of feedback, causing valuable and actionable data to simply be discarded. Another advantage of using a sentiment analysis algorithm is that it can be deployed in near real-time capacities. For example, brands can use a sentiment analysis algorithm to detect potential PR issues or upset customers. Sentiment analysis gives organizations the ability to become proactive and mitigate crises or complaints before they balloon. Thirdly, another advantage sentiment analysis offers is consistency. Tagging and categorizing text are highly subjective. Thus, using human annotators for sentiment analysis tasks in deployment can often yield inaccurate results, influenced by the annotator's personal experiences and beliefs.

Despite these advantages afforded by algorithmic sentiment analysis, there are still a few challenges that researchers are attempting to mitigate. Irony and sarcasm are tough edge cases in sentiment analysis due to words often taking on the opposite of their usual sentiment. For example: "I'm thrilled" can sometimes indicate a person being excited, or it can indicate exasperation when used ironically. In addition, the definition of "neutral" is also very tricky due to ambiguity.

The problem of sentiment analysis is framed in several different ways. Structured Sentiment Analysis seeks to identify the "holder", "target", "polarity", and "sentiment expression" of a given sentence. In our model we don't use this approach, we assume the "holder" is the speaker, the "target" is the movie being reviewed, the "polarity" is the score, and the "sentiment expression" is the whole review. This is necessarily inaccurate, a review may praise some parts of a movie like the acting and cinematography, and criticize other parts like the writing or the soundtrack. They may also describe the opinions of others in their review, other reviewers, the creators of the movie, etc. However, overall this simplification will still create good results.

2 Related Works

2.1 Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis

This paper explores the issue of contextual polarity, which is the idea that words can be positive or negative depending on their contexts. A word or phrase may be negated, which can be due to a nearby or far away negative word. Modifiers like ”little”, ”less”, and ”more” can be positive or negative depending on the word they are modifying. Their solution to this problem has 2 steps. They started with an annotated corpus with prior polarity, then performed a two-step phrase-level sentiment analysis. First, they assign each phrase (using clues) as neutral or polar. Then in the second step, all the phrases are assigned as polar and disambiguate their contextual polarity (positive, negative, both, or neutral). We do not incorporate the idea of contextual polarity in our systems, the reviews are converted into word vectors that only are concerned with which words appear and how often they appear, which is an issue with our approach.

2.2 Structured Sentiment Analysis as Dependency Graph Parsing

This paper uses a different framing of sentiment analysis as their problem: they focus on structured sentiment analysis which is a problem that seeks to make sentiment analysis more complex by identifying 3 elements of a sentence: a holder (the person/group/idea/etc. holding the sentiment), a target (the person/group/idea/object/etc. receiving the sentiment), and a sentiment expression (“good”, “bad”, “excellent”, “needs work”, etc.). which come together to produce a polarity (positive or negative) that the holder believes about the target. This is a different approach than the one we use, and the increased complexity of their problem illustrates issues with our approach: we assume that all of the reviews have a holder which is the speaker, and a target which is the movie being reviewed, which is not true in practice.

2.3 Learning Word Vectors for Sentiment Analysis

This paper uses a probabilistic model similar to but distinct from Latent Dirichlet Analysis to learn word vectors that capture semantic information. The idea is to overcome the issue where Latent Dirichlet Analysis is good at capturing topics from word co-occurrences, but can’t easily connect those to sentiment. We don’t opt for their model but instead use the simpler LDA approach and use linear regression to find correlations between topics and sentiment. This paper also is the source of the dataset that we use, the Large Movie Review Dataset which is composed of 50000 movie reviews taken from IMDB.

2.4 Language Independent Sentiment Analysis with Sentiment-Specific Word Embeddings

This paper uses a baseline Word2Vec system and uses it as a starting point for training to create word embeddings enhanced with sentiment information. They train their system on a database of French tweets and use emojis as a label for the sentiment (happy faces are positive sentiment, sad faces are negative sentiment). They train their system using deep learning approaches: specifically a convolutional neural network. Our system is quite different in design but we also use word embeddings, though we do not train our own instead use pre-trained ones from OpenAI.

2.5 Sentiment Analysis for Movie Reviews Dataset Using Deep Learning Models

This paper focused on developing a sentiment analysis classification system using deep learning networks and introducing comparative results of different kinds of neural networks. They used Multilayer Perceptron (MLP) as a baseline to evaluate the other networks. They looked at Long Short Term Memory (LSTM), Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), and a hybrid model of LSTM + CNN. We also use a Multilayer Perceptron model in our system but nothing is more advanced in terms of deep learning. They also compare their results to SVM and Naive Bayes approaches, both of which we use. They also build upon the work done in Language Independent Sentiment Analysis with Sentiment-Specific Word Embeddings and preprocess their models with Word2Vec. We use a similar approach but use OpenAI word embeddings instead, and for each model compare the results with and without preprocessing.

3 Motivation

As the demand for language processing and sentiment analysis rises with the drastic increase in digital data, we felt it would be very interesting to understand and examine the different classifiers that are commonly used in Natural Language Processing. NLP is a very broad umbrella so not all the tools that fall under this umbrella will be the most effective at the complex task of determining the sentiment of human text. Below are some of the popular classifiers used in NLP we wanted to explore

1. **Decision Trees**: A classifier that uses a tree-like model of decisions, with decisions represented by branches and the outcome of the decision represented by the node.
2. **Random Forest**: A robust classifier that leverages many randomly-created decision trees.
3. **Naive Bayes**: A simple probabilistic classifier where the presence or absence of a particular feature is independent of the presence or absence of any other feature. This classifier is very computationally light due to its simplicity.
4. **Support Vector Machines (SVMs)**: A flexible classifier that works well on a wide range of data sets using hyperplanes and support vectors. It is particularly effective at finding complex, non-linear relationships in data. Due to its effectiveness with non-linear relationships, it is often used to find decision boundaries in complex datasets.
5. **Neural networks**: Type of classifier that are modeled after by the structure and function of the brain. It is composed of and are composed of interconnected nodes called "neurons" arranged in layers. They can be used to model complex non-linear relationships between input and output.
6. **Nearest neighbor**: An intuitive classifier that assigns training labels by matching it with test data it has the most similarity with. It performs this matching by calculating every distance between each test and training data points (Euclidean or Manhattan distance).

4 Data

The data used for this research paper was *Large Movie Review Dataset*. We obtained the data from the authors of *Learning Word Vectors for Sentiment Analysis* from Stanford AI Lab. The authors of *Learning Word Vectors for Sentiment Analysis* compiled 50,000 IMDB movie reviews from IMDB, with 30 reviews or less per movie to prevent the more popular movies from skewing the data. This constructed dataset was scored on a scale of 1-10, with [1,4] being negative and [7,10] being positive. This dataset contained the roughly same amount of positive and negative reviews, so an algorithm guessing at random would result in 50% accuracy.

Each record in the dataset contains the numerical ranking as an integer, then the review as a string. The authors of the paper split half of the data into a testing set (25,000 reviews), and a half into a training set(25,000 reviews). We split the training set in the source into 2 datasets: one training set (12,500 reviews) and one development set (12,500 reviews). In testing, we used cross-validation by combining our training (12,500 reviews) and test (25,000 reviews) sets and then re-splitting them into a training set of (22,500 reviews) and a test set (15,000 reviews). We will opt to redistribute the test train split, which we will describe in the Experiment section.

5 Methodology

To perform sentiment analysis, the reviews must be transformed from text to meaningful numerical representations. The most popular option is Count Vectorizer and TF-IDF Vectorizer. Count Vectorizer uses a bag-of-words approach to count the number of times a word occurs. TF-IDF goes one step further by providing both frequencies of words in the corpus and the importance of the words. By providing us with the importance of the words, we can reduce the dimensionality For this reason, we decided to use TF-IDF Vectorizer and Transformer as our baseline.

For comparison, we also wanted to experiment with OpenAI's new state-of-the-art GPT-3 embeddings. We chose to use *text-embedding-ada-002* even though Davinci is OpenAI's most powerful model, however it is exponentially slower and more expensive. Since ADA is the fastest, best-performing, and cheapest, we decided to use it to compare against the free embeddings achieved by the scikit-learn TF-IDF package.

We implemented our evaluation of the following different classifiers in scikit-learn

1. Random Forest
2. QDA
3. Naive Bayes
4. Decision Tree
5. Nearest Neighbor
6. Linear SVM
7. Gaussian Process
8. AdaBoost
9. Gradient Boosting
10. Neural Net
11. LDA (Latent Dirichlet Analysis)

In addition to experimenting with scikit-learn models, we also wanted to attempt to implement our own LDA classifier. Our LDA system was unique compared to our other systems in its implementation and incorporates Linear Regression on the topic weights for each document, to find topics that correlate with sentiment.

6 Experiments

When we obtained the source data, it was split 50/50 between training and testing data. We decided we'd like a higher ratio of training data to testing data. Thus, we joined the input test data and train data and used scikit-learn *test_train_split()* to achieve a 75/25 split. We wanted our model to be able to train on more data to improve the classifications it makes.

By doing our test_train_split, we also had flexibility over the splitting of test and train datasets. During the development phase, we set *random_seed=0* to have a fair comparison between all the classifier's classification reports. However, we were then able to remove the *random_seed* parameter to perform cross-validation.

In addition to evaluating the different classifiers' effectiveness for sentiment analysis, we wanted to see the effect of hyperparameter tuning. We used *GridSearchCV* to find the optimal input for the following parameters

1. n_estimator: [10, 100, 1000]
2. max_depth: [5, 10, 50, 100]
3. min_impurity_decrease: [0, 0.1, 1]
4. max_features: [1, 10, 100, 1000, None]

It took over 10 hours for the *GridSearchCV* to compute for *RandomForestClassifier()* due to the size of the dataset and the number of permutations. With the parameters produced by *GridSearchCV*, we were able to use them to tune the classifier and model for comparison.

After transforming the training data with TF-IDF, we ran it with an LDA system with 500 topics. We then used linear regression on the output to find which topics correlated with the sentiment.

Our own LDA classifier uses a few custom functions to transform the words and the following custom functions:

1. get_tf_idfs(doclist): calculates TF-IDF scores for each doc, returns those, and a list of words for all docs.
2. lda(tf_idfs, topics, test_tf_idfs): performs Latent Dirichlet Analysis on TF-IDF scores to output topics
3. find_polarity_topics(doc_topics, review_ polarities): find the polarity of topics by performing linear regression with an array of topic scores (X) and polarities (Y)
4. predict_polarity(coeffs, intercept, doc_topics): predicts polarity by using the dot product of coefficient with the topic score, then adding the intercept.

For evaluation, we use the built-in *classification_report* from scikit-learn to obtain the accuracy, precision, recall, and F1 score for each model.

7 Results and Discussion

Our findings have been summarized in Tables 1-5. We compared of our Discrimination Analysis classifiers, LDA and QDA. Our LDA classifier performed very poorly with predictions achieving 67% precision, 69% recall, and 68% accuracy. Given that the baseline is roughly 50% when the algorithm guesses randomly, the predictions made by our LDA classifier is very undesirable. QDA(Quadratic Discrimination Analysis) with TF-IDF was able to accomplish 84% precision, 89% recall, and 87% accuracy. The main difference between LDA and QDA is that LDA assumes that the data follows a Gaussian distribution and that the classes have equal covariance matrices, whereas QDA does not. This generally means that QDA can model more complex relationships

Table 1: Accuracy for classifiers tested

Classifier	Embedding	Accuracy
LDA (500 topics)	N/A	0.68
Random Forest	TF-IDF	0.86
	OpenAI	0.85
Random Forest (Hy-pertuend)	TF-IDF	0.91
	OpenAI	0.90
QDA	TF-IDF	0.87
	OpenAI	0.54
Decision Tree	TF-IDF	0.88
	OpenAI	0.76
Nearest Neighbor	TF-IDF	0.73
	OpenAI	0.85
Linear SVM	TF-IDF	0.81
	OpenAI	0.91
Neural net	TF-IDF	0.85
	OpenAI	0.91
Gaussian Process	TF-IDF	0.87
	OpenAI	0.91
Ada Boost	TF-IDF	0.91
	OpenAI	0.85
Gradient Boosting	TF-IDF	0.81
	OpenAI	0.85
Naive Bayes	TF-IDF	0.88
	OpenAI	0.89

between the features and the classes than LDA. These results make sense and can be corroborated by other research. Given that we know our data is not Gaussian distributed, LDA assumes a Gaussian distribution while QDA does not, it is logical that QDA outperforms LDA. ***Sentiment analysis using Latent Dirichlet Allocation and topic polarity wordcloud visualization*** indicated that their LDA sentiment analysis system scored a F-measure of 61%[6] (Our LDA had F-measure of 68%) Comparatively, our LDA classifier performed similarity so it is perhaps confirmation that LDA is not suitable for complex sentiment analysis tasks. It is worth further noting that QDA classifier performed significantly worse when using OpenAI's GPT-3 ada embedding instead of Scikit-Learn's TF-IDF.

Another pair of classifier that can be discussed in conjunction are AdaBoost(Adaptive Boosting) and Gradient Boosting classifiers as they are ensemble learning method. We observe that our AdaBoost classifier outperforms over Gradient boost consistently with both embeddings. To understand why, we need to think of how the two classifiers differ. Ensemble learning methods create multiple models to combine them in order to achieve improved results. Typically, the weak model that requires boosting are decision trees with small number of branches. Due to the multi-model setup of ensemble learning, both AdaBoost and Gradient Boosting are well ideal for data that is noisy or has complex relationships between features and target. One key difference between AdaBoost and gradient boosting is the way in which the weak models are boosted. In AdaBoost, the weights of the training examples are adjusted so that the next weak learner focuses more on the examples that were misclassified by the previous weak model. For gradient boosting, boosting the week model is done by training the weak learners sequentially, with each weak modeled trained to correct the misclassifications made by the previous weak model. This sequential training is achieved through regressing the weak models to the negative gradient of the loss function associated with the previous weak model. In fact, it is valid to consider AdaBoost a special case of Gradient Boost. Due to above differences, Gradient boosting does not penalize misclassifications, but use loss function instead. This informs us that the weak models contain a lot of misclassifications so by penalizing the mistakes, AdaBoost was able to produce better results.

Next, we'd like to examine the very interesting results from Nearest Neighbor classifier. When using TF-IDF to make positive predictions, the precision score (82%) is significantly different from the recall score(60%). However, when using GPT-3 ada-embedding, there is negligible difference between the precision(86%) and recall(84%). Similarily for when making negative predictions with TF-IDF, the precision (68%) and recall(87%) differed greatly while the precision(85%) and recall(87%) when using GPT-3 ada-embedding. Since we know our data contain roughly equal amounts of positive and negative data, the distribution of positive and negative training/testing data is not the cause. Let us remind ourselves the difference between precision and

Table 2: random parameters for random forest vs. random forest with hyper tuned parameters; Positive

Parameters	Embedding	Precision	Recall	F1
Random	TF-IDF	0.90	0.87	0.88
Random	OpenAI	0.87	0.93	0.90
Hypertuned	TF-IDF	0.90	0.87	0.88
Hypertuned	OpenAI	0.87	0.93	0.90

Table 3: random parameters for random forest vs. random forest with hyper tuned parameters; Negative

Parameters	Embedding	Precision	Recall	F1
Random	TF-IDF	0.89	0.83	0.86
Random	OpenAI	0.85	0.85	0.85
Hypertuned	TF-IDF	0.91	0.93	0.92
Hypertuned	OpenAI	0.93	0.87	0.90

recall in order to analyze this interesting outcome. Recall is dividing true positives by all events that **should** have been predicted as positive, whereas precision is true positives divided by all positive predictions. When the precision is higher than the recall, it can be interpreted that the classifier is more confident about its positive predictions is less likely to predict false positives, which would increase the precision. However, this could also lead to missed positive cases, which would decrease the recall. Looking at our result, we can infer that when using TF-IDF, the classifier made a lot of irrelevant negative predictions, and failed to return many of the relevant predictions when making positive predictions. However, we see that GPT-3 ada-embedding mitigated this entirely. One key difference between TF-IDF and GPT-3 is how they represent words. TF-IDF represents words as a numeric value based on their frequency and relevance in the corpus, while GPT-3 represents words as vectors in a high-dimensional space (word embeddings). These vectors capture the meaning and context of the words, allowing GPT-3 to understand the relationships between words and to perform various language tasks. As mentioned with the challenges of sentiment analysis, it is possible that the reviews were very complex with irony (very common with movie reviews) and contextual polarity with large distances. These could cause the word frequency or relevance to be insufficient to capture the nuance in the reviews. Given that nearest neighbor relies on distance between the words, word embeddings are more suitable as they represent relationships between words by capturing context and meaning.

Neural Networks is the next frontier in all things Machine Learning and Artificial Intelligence, so we wanted to examine a neural network classifier. Since Scikit-Learn has several NN-models, we chose to use MLPClassifier due to its simplicity(minimal data pre-processing) and speed. We see the classifier performed very well using either TF-IDF and GPT-3 ada-embeddings, with word embeddings with a slight (5%) advantage. This is not surprising as the different layers and neurons enable neural networks to be able to capture contextual information and dependencies between words. An example, ”that was fun” is usually a positive statement as ”fun” is a positive word. However, consider the phrase with the context of ”We were all caught in the rain without an umbrella and a taxi splashed us. That was fun”. In this scenario, the context informs that the phrase is actually negative(irony, one of the challenges stated earlier). Due to its complex design a NN can use this contextual dependencies it just learned about irony to be able to detect irony better the next time ”that was fun” is used sarcastically. For this reason, our neural network classifier using word embeddings is tied for one of the 3 best-performing predictors.

Finally, we'd like to discuss our Random Forest classifier and Decision Tree classifier in conjunction as they are based on the same concept of tree-like models. In addition, we also performed hyperparameter tuning with Grid Search method for Random Forest. Tuning with GridSearch involves specifying a grid of hyperparameter values to test, and evaluating the model for each combination of values. Since Random Forest already is computationally intensive, the tuning process took very long. With the optimized parameters, we saw a small but still meaningful increase in the performance of our classifier (5%). However, it was very surprising to see that there is no large improvement in prediction quality between random forest (untuned for more fair comparison) and decision tree. It is usually expected for random forest to outperform decision tree as random forest is another ensemble method which leverages many decision trees. One idea for why the decision tree performed so similarly to random forest is that the data is high-dimensional data. With high-dimensional data, decision trees may be more effective as they can handle more features without the need for feature selection.

In general, our findings support the idea that random forest is a flexible and well-performing method across several different problems despite taking more time to complete. Nearest neighbors likely performed more poorly than other systems because our data has very high dimensionality, which means distance in a multidimensional space (i.e. ”nearness”) isn’t a good metric for document similarity. This is exacerbated by the fact that most words occur in a small fraction of documents, meaning that each dimension is only non-zero for a small fraction of documents. LDA likely performed poorly for the same reason, along with the relatively small number of topics used in our system.

However, most of our classifiers performed well overall, having scored in the 80% to 90% range for precision, recall, and accuracy. Despite the dominance of deep learning approaches for sentiment analysis (as well as most problems in machine learning), our neural net system wasn’t significantly better than our other systems, likely since it had a relatively small number of neurons and layers, it doesn’t incorporate advanced neural network designs like LSTMs and CNNs, and that many of our other systems already performed very well.

8 Future Works

8.1 GPC

The Gaussian Process Classifier a classifier that performed very solid in our test, consistent across precision, recall, and both positive and negative predictions. This was surprising to us as our data does not follow a Gaussian distribution, thus did not expect a classifier using a latent function (Gaussian process) would perform well. We would love to explore further in the future to understand why.

8.2 Hyperparameter Tuning

We saw the benefit of hyperpararmeter tuning with our RandomForestClassifier(). However, due to the heavy computation, it took incredibly long to run and we were unable to perfect parameter tuning for the other classifiers. In the future, we may search for optimal parameters for the other classifiers. Perhaps the optimal parameters will have a significant impact on how the they classificiers’ predictions will rank.

9 State of the Art

Latent Dirichlet Allocation (LDA) is a widely used topic modeling algorithm that is particularly popular in the field of natural language processing (NLP). It is a generative model that represents each document as a mixture of a fixed number of topics, and each word in the document is assumed to be generated from one of those topics. LDA is an unsupervised learning method, which means it does not require labeled training data.

Latent Dirichlet Allocation works under the assumption that every document in a set is a distribution of “topics”: each document is assigned a weight for each topic. Each topic is defined as a distribution of words: each topic is assigned a weight for each word. Say we want to discriminate between documents about politics versus documents about sports. We might have one topic that refers to negative movie reviews with high weights for words like: “worst”, “bad”, “awful”, or “boring”, and one topic that refers to positive movie reviews with high weights for words like: “loved”, “fun”, “wonderful”. These topics would have high weights on each side of the review documents.

In this application we are looking for topics that correlate with positive sentiment, for example having high weights for words like ”impressive”, “well-done”, and “beautiful”, which correlate with positive statements, and words like ”trashy”, ”cheap”, ”cliche” for negative sentiments.

Other common systems for sentiment analysis are Vector Space Models (VSMs) and various deep-learning approaches. Vector space models work by converting a document to a vector where each index represents a meaningful parameter related to the document. One such method is Latent Semantic Analysis which analyzes the values in a term-document frequency matrix (typically weighted by tf-idf scores) and produces a lower-rank matrix that collapses words with similar distributions together. This is similar to LDA in that it will convert a word vector to a topic vector.

Deep learning approaches are valuable because they are usually the highest performing but are also more opaque in terms of how they work. A fully trained LDA model is understandable by a human since they can look at the topics and see which words correlate with each other, while the weights and biases of a neural network are not interpretable.

10 Team Contributions

Nancy Ma: Building pipelines, working with OpenAI API, result analysis, paper presentation

Abhilasha Tandon: LDA prediction system, research, found data source, paper, presentation

Alex Ding: Testing, debugging, and running coding, paper, presentation

11 Reference

[1] Wilson, T., Wiebe, J.,; Hoffmann, P. (2009). Recognizing contextual polarity: An exploration of features for phrase-level sentiment analysis. Computational Linguistics, 35(3), 399–433.

Barnes, J., Kurtz, R., Oepen, S., Øvrelid, L.,; Velldal, E. (2021). Structured sentiment analysis as dependency graph parsing. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers).

Lan, M., Zhang, Z., Lu, Y., ; Wu, J. (2016). Three convolutional neural network-based models for learning sentiment word vectors towards sentiment analysis. 2016 International Joint Conference on Neural Networks (IJCNN).

Saroufim, C., Almatarky, A.,; Abdel Hady, M. (2018). Language independent sentiment analysis with sentiment-specific word embeddings. Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis.

] Mohamed Ali, N., El Hamid, M. M.,; Youssif, A. (2019). Sentiment Analysis for movies reviews dataset using Deep Learning Models. International Journal of Data Mining &amp; Knowledge Management Process, 09(03), 19–27.

Mohammad F. A. Bashri, Retno Kusumaningrum (2017). Sentiment analysis using Latent Dirichlet Allocation and topic polarity wordcloud visualization. Conference: 2017 5th International Conference on Information and Communication Technology (ICoIC7)

Table 4: Classification report for classifiers tested; Positive outcomes

Classifier	Embedding	Precision	Recall	F1
LDA (500 topics)	N/A	0.67	0.69	0.68
Random Forest	TF-IDF	0.83	0.9	0.86
	OpenAI	0.85	0.85	0.85
Random Forest (Hyptertuned)	TF-IDF	0.90	0.87	0.88
	OpenAI	0.87	0.93	0.90
QDA	TF-IDF	0.84	0.89	0.87
	OpenAI	0.52	0.79	0.63
Decision Tree	TF-IDF	0.87	0.9	0.89
	OpenAI	0.73	0.77	0.75
Nearest Neighbor	TF-IDF	0.82	0.6	0.7
	OpenAI	0.86	0.84	0.85
Linear SVM	TF-IDF	0.77	0.89	0.83
	OpenAI	0.9	0.87	0.88
Neural net	TF-IDF	0.85	0.87	0.86
	OpenAI	0.91	0.91	0.91
Gaussian Process	TF-IDF	0.85	0.9	0.87
	OpenAI	0.91	0.91	0.91
AdaBoost	TF-IDF	0.9	0.87	0.88
	OpenAI	0.86	0.84	0.85
Gradient Boosting	TF-IDF	0.78	0.85	0.82
	OpenAI	0.83	0.88	0.85
Naive Bayes	TF-IDF	0.86	0.89	0.88
	OpenAI	0.87	0.93	0.9

Table 5: Classification report for classifiers tested; Negative outcomes

Classifier	Embedding	Precision	Recall	F1
LDA (500 topics)	N/A	0.69	0.67	0.68
Random Forest	TF-IDF	0.89	0.83	0.86
	OpenAI	0.85	0.85	0.85
Random Forest (Hyptertuned)	TF-IDF	0.91	0.93	0.92
	OpenAI	0.93	0.87	0.90
QDA	TF-IDF	0.89	0.83	0.86
	OpenAI	0.59	0.29	0.39
Decision Tree	TF-IDF	0.9	0.87	0.88
	OpenAI	0.78	0.75	0.76
Nearest Neighbor	TF-IDF	0.68	0.87	0.76
	OpenAI	0.85	0.87	0.86
Linear SVM	TF-IDF	0.87	0.73	0.79
	OpenAI	0.91	0.93	0.92
Neural net	TF-IDF	0.85	0.84	0.85
	OpenAI	0.91	0.91	0.91
Gaussian Process	TF-IDF	0.89	0.84	0.86
	OpenAI	0.91	0.91	0.91
AdaBoost	TF-IDF	0.91	0.93	0.92
	OpenAI	0.83	0.85	0.84
Gradient Boosting	TF-IDF	0.84	0.76	0.80
	OpenAI	0.88	0.83	0.86
Naive Bayes	TF-IDF	0.89	0.86	0.87
	OpenAI	0.92	0.85	0.88