Stats 101C Final Project

Roger Wilson, Wolfe Pickett, Mark Felici,
Shiyu Murashima, Din-Yin Hsieh

December 2023

Abstract

IMDb is an online database with information regarding film and television - including cast and production crews, plot summaries, and most importantly, reviews and ratings. Being able to classify a review's sentiment allows us to determine the consensus of a TV show or movie rating. We will be going in depth on a text analysis of the correlation between the reviews and sentiments using models such as Logistic Regression, K-Nearest Neighbors, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Decision Trees, Random Forest, and Support Vector Machine.

1 Introduction

Many individuals rely on the reviews of others to determine whether a movie is worth watching. In recent years, IMDb has emerged as an immensely popular tool for movie reviews, enabling consumers to peruse thousands of opinions and make informed decisions without the fear of wasting their time. However, the wealth of information poses certain challenges. Short of reading each review individually and manually sorting them, it can be challenging to discern the general consensus. In our project, we were assigned the task of analyzing an IMDb dataset comprising 50,000 reviews, categorized as either "positive" or "negative." Our objective is to accurately and effectively categorize these reviews based solely on the words and sentiment expressed in each review.

2 Data Preprocessing

Our preprocessing steps are as follows:

1. Remove stop words using the nltk package. Stop words are overly common words that don't add specific meaning on their own. However, we opted to leave certain stop words such as 'no', 'not', 'shouldn't, and any other negations, to give more context for Word2Vec.
2. Using a package called Contractions, expand all contracted words (should've, hadn't, etc.)
3. Convert each review to lowercase.
4. Remove all text in square brackets, hyperlinks, and html tags.
5. Remove all punctuation, numbers, and newline tags.

Then, we proceed to perform stemming and lematization on the preprocessed data. Stemming is used to reduce an inflected word down to its word stem (e.g. "programming", "programmer", and "programs" are all "program"). It reduces the number of unique words, which as a result can make the algorithm run faster and more efficiently. However, stemming at times is not as accurate and can reduce inflected words to the same word stem when they are not related.

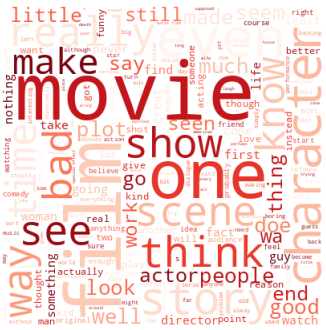
Lemmatization is used to reduce inflected words to their root word, relying on accurately determining the intended part-of-speech and meaning of the word based on the context. Since lemmatization does not cut off words like stemming, it is often more accurate. However, it is also very time-consuming because it involves performing morphological analysis and deriving the meaning of words.

Overall, the stemming approach is much faster than lemmatization, but it can potentially lead to unrecognizable base roots. On the other hand, lemmatization is much more accurate than stemming, and since we are looking to increase our accuracy for our models, we decided to move forward with only lemmatization, and not the stemming. Both did not have much of an effect on the performance of the models, but lemmatization was more consistently slightly better.

By cleaning and standardizing the IMDb reviews, we are improving the quality of the input for our TF-IDF and Word2Vec matrices, and thus our classification models. [1]

(a) **Basic Wordcloud**. The frequency of each word, excluding the stopwords, in the original dataset.

(b) **Positive Wordcloud**. The Frequency of each word in positive reviews, excluding the stopwords.



(c) **Negative Wordcloud**. The frequency of each word in negative reviews, excluding the stopwords.

Figure 1: **Three Wordclouds for the Original Dataset.**

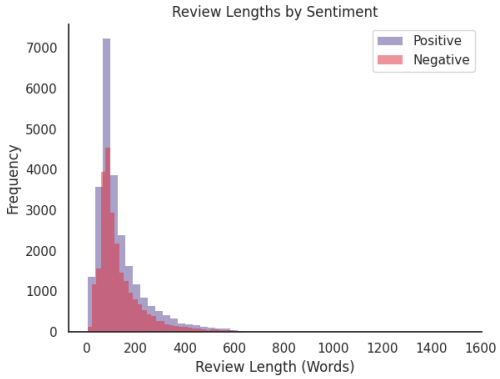


Figure 2: **Review Length**. The length of reviews by sentiment.

2.1 Word2Vec

Word2Vec is a machine learning technique for constructing word embeddings, which is a method of representing words as vectors to capture the semantic and syntax relationships between them. There are two main approaches behind the algorithm: Skip Gram and Common Bag Of Words (CBOW). The CBOW Model takes the contextual information of each word as input and predicts the word that aligns with that context. The Continuous Skip-Gram model predicts words occurring within a specific range both before and following the current word in a given sentence. Both methods have their advantages and disadvantages, but CBOW tends to be faster and generate better representations for more frequent words.

In the model used, we implemented Word2Vec with CBOW to mathematically vectorize the word embeddings. We essentially train a Word2Vec model on the tokenized review data, and then compute document vectors by averaging the word vectors for each review. An example use case is being able to predict and find similar words. For example, the word "film" which appeared frequently in the data, is most similar to movie, picture, documentary, and cinema, in decreasing order. Based on the Word2Vec model, we partitioned the data into training and testing sets, employing them for prediction tasks such as logistic regression, KNN, Decision Tree, and more.

2.2 Term Frequency - Inverse Document Frequency (TF-IDF) Matrix

A TF-IDF matrix measures the importance of a word to a document in a corpus, adjusted for the fact that some words appear more frequently in general. Term frequency, $\text{tf}(t,d)$ is the relative frequency of term t within document d divided by the number of terms in the document $\text{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$, where $f_{t,d}$ is the frequency. The inverse document frequency is a measure of how much information the word t provides across all documents D, so $\text{idf}(t,D) = \log \frac{|D|}{|\{d \in D : t \in D\}|}$, or the log of the total number of documents in the corpus divided by the number of documents where the term t appears. [2]

TF-IDF matrices are useful for text processing because they measure word importance, are easy to interpret, and are widely used. On the other hand, they fail to recognize word context, and as our IMDb reviews are very long, the resulting matrix is very sparse.

When we created a TF-IDF matrix of each IMDb review, we decided to use all words across every review as the terms. The result is a 50,000 x 200,000 sparse matrix. Given that an IDF score represents how much information a word provides, we decided to filter the matrix for low and high IDF scores, because low scores are overly common words, which present no new information in training, and high scores correspond to infre-

Statistic	Value
Count	201783
Mean	10.58
Standard Deviation	1.125
Minimum	1.23
25% Quartile	10.43
Median (50% Quartile)	11.127
75% Quartile	11.127
Maximum	11.127

Table 1: **Distribution of IDF Scores**. A thing to note is that the data before filtering words with an idf score of between 2 and 10 has a dimension of $\mathbb{R}^{50000\times201783}$.

quent words, which could lead to overfitting in training. By iterating through possible IDF score ranges, we found that limiting scores to between 2 and 10 not only maximized accuracy, but also reduced the enormity of our feature space to around 36,000 features. We also added a L2 regularization term in the creation of our TF-IDF matrix, so the sum of squares of vector elements is 1, and the cosine similarity between two vectors is their dot product when the L2 norm has been applied.

2.3 Dimension Reduction

Our TF-IDF matrix still had 36,000 features after IDF filtering and was significantly hindering the efficiency of some of our models such as KNN that don't perform well in high-dimensional spaces. To remedy this, we employed Principal component analysis (PCA), a dimension reduction technique that creates a new coordinate system in which the new variables are independent of one another and to reduce the number of features. Increasing the number of components captures more information but also increases the risk of overfitting and is computationally expensive. We chose n = 500 principal components to still preserve a fair amount of the TF-IDF variance, which had little effect on model accuracies but greatly improved runtime.

3 Experiments

In this section, we give some brief explanations about how each models we have decided to use theoretically work, as well as the models' performance across two preprocessed datasets (Word2Vec and TF-IDF) using three different metrics.

3.1 Logistic Regression

Logistic Regression serves as a versatile extension of Linear Regression, gaining popularity for predicting continuous variables. However, its distinct strength lies in its application to binary classification tasks. This technique estimates the conditional probability of a binary event based on given data. The logistic function, often referred to as the sigmoid function, transforms any real-valued number into a value between 0 and 1, making it apt for categorizing outcomes as "positive" or "negative." In the context of logistic regression, features are assigned weights, and the weighted sum is passed through the logistic function to calculate the probability of belonging to the positive class. The model is trained by adjusting these weights through an optimization process, aligning predicted probabilities with actual class labels in the training data. Throughout our exploration, we considered the incorporation of different regularization methods, including L1 and L2 regularization. Regularization is employed to counter overfitting by constraining large coefficients in the model. Despite its potential benefits, we ultimately decided against implementing regularization. Surprisingly, even without regularization, our model exhibited remarkably high accuracy, prompting the decision to forego additional regularization measures [3, 4, 5, 6, 7].

For the Word2Vec preprocessed data, we achieved an accuracy of 86.79%, F1 score of 0.8686, and AUC of 0.9395.

For the TF-IDF preprocessed data, we achieved an accuracy of 87.60%, F1 score of 0.8760, and AUC of 0.9482.

3.2 KNN

K nearest neighbors, commonly referred to as KNN, is a supervised learning method that classifies data points based on their proximity to other data points, typically defined through Euclidean distance or a similar metric. An inherent advantage of KNN lies in its non-parametric classification, affording the model considerable flexibility, particularly advantageous when dealing with extensive datasets. Given that the user denotes the value for K, we conducted tests for K = 1, 3, ..., 9, aiming to identify the optimal K. Notably, testing exclusively involved odd numbers, a practice rooted in the demonstrated poor outcomes associated with even values for K. Following the training of our model across various K values, the resulting metrics were tabulated as follows: For the Word2Vec preprocessed data, we achieved an accuracy of 79.98%, F1 score of 0.7961, and AUC of 0.8738.

For the TF-IDF preprocessed data, we achieved an accuracy of 54.37%, F1 score of 0.212, and AUC of

0.6375. Strikingly, in comparison to alternative models, our KNN model trained using the TF-IDF preprocessed data proved to be the least accurate. This outcome contradicts the typical expectation that the flexibility of KNN contributes to higher accuracy. Upon careful consideration, we discerned that the observed low accuracy can be attributed to the curse of dimensionality. With a vast number of variables in our dataset, most points likely share similar distances to one another. Consequently, irrespective of the K value, a diverse selection of values is anticipated in a confined area. The Word2Vec data likely performed better as Word2Vec allows for more context to be given to words, hence creating more of a distinction between data points. Once we performed PCA on the data preprocessed with TF-IDF, we saw a decent improvement in accuracy.

We achieved an accuracy of 68.73%, F1 score of 0.6863, and AUC of 0.7527.

3.3 Decision Tree

A tree based model that we have adopted is the decision tree model. Decision tree model works by continuously splitting subsets of the data into two parts, each with half of the size of its parent subset. At each split, an impurity function is used to calculate the quality of the split, or the goodness of fit. Finally, the model assigns a label to each terminal subset, and subsequently all data points in that specific subset.

For the training process, we set the impurity function as the Gini impurity

$$\sum_k p_{tk}(1 - p_{tk}) \tag{1}$$

where t is the node, k the label of the data, and p_{tk} the probability of probability of label in node t, and the max depth as 'None' [8, 9, 10, 11].

For the Word2Vec preprocessed data, we achieved an accuracy of 73.27%, F1 score of 0.7326, and AUC of 0.7327.

For the TF-IDF preprocessed data, we achieved an accuracy of 70.71%, F1 score of 0.7071, and AUC of 0.7070.

3.4 Random Forest

Another tree-based model we utilized was a random forest. As mentioned earlier, there are various advantages to employing decision trees for classification. However, the most significant drawback of decision trees is their pronounced susceptibility to overfitting. Random forests aim to mitigate this issue while maintaining the effectiveness of decision trees. The strength of random forests lies in processing the data through a collection of decision trees and making predictions based on the class that receives the most votes. Our random forest comprised 100 trees, each with a maximum depth of 5. We decided to limit the depth of each tree as computational costs for random forests can be quite expensive if not regulated.

For the Word2Vec preprocessed data, we achieved an accuracy of 81.35%, F1 score of 0.8154, and AUC of 0.8947.

For the TF-IDF preprocessed data, we achieved an accuracy of 78.50%, F1 score of 0.7848, and AUC of 0.8582.

Note: We were unable to run the Random Forest Model on the TF-IDF data that did not undergo PCA as our Google Colab would crash.

3.5 SVM

The SVM (support vector machine) model works by defining a hyperplane that separates different data points into two classes, which is really suitable for the IMDb dataset. For our specific case, we utilized `LinearSVC` from Scikit-Learn that specifically uses a linear kernel to find the hyperplane [3, 12, 13, 14, 15].

For the training process, we set the following parameters:

- penalty: l2
- loss: squared_hinge
- C: 1.0

Since the original `LinearSVC` model can not output prediction probabilities, we also utilized the `CalibratedClassifierCV` to output the probabilities and, subsequently, calculate the ROC AUC.

For the Word2Vec preprocessed data, we achieved an accuracy of 86.93%, F1 score of 0.8700, and AUC of 0.9401.

For the TF-IDF preprocessed data, we achieved an accuracy of 87.69%, F1 score of 0.8769, and AUC of 0.9483.

3.6 LDA

One of the generative models that we decided to adopt is the LDA (Linear Discriminant Analysis). Compared to discriminative models, even though some conditions have to be satisfied before using generative models, generative models generally deliver better performances than discriminative models [16].

Then, by assuming the following conditions: both groups follow a multivariate normal distribution and have the same covariance vectors but different mean

	Accuracy		F1 Score		ROC AUC	
	W2V	TF-IDF	W2V	TF-IDF	W2V	TF-IDF
Logistic Regression	0.8679	<u>0.8760</u>	0.8687	<u>0.8760</u>	0.9395	<u>0.9482</u>
KNN	0.7998	0.6873	0.7962	0.6863	0.8739	0.7527
LDA	**0.8719**	0.8711	**0.8733**	0.8710	**0.9414**	0.9445
QDA	0.7853	0.8139	0.7741	0.8137	0.8701	0.8848
Decision Tree	0.7327	0.7071	0.7326	0.7071	0.7327	0.7070
Random Forest	0.8135	0.7850	0.8154	0.7848	0.8948	0.8582
SVM	<u>0.8693</u>	**0.8769**	<u>0.8700</u>	**0.8769**	<u>0.9401</u>	**0.9483**

Table 2: **Comparison of Performance Across All Models Based on Word2Vec and TF-IDF Preprocessing (After PCA). Bold indicates best performing, underline indicates second best performing.** For Word2Vec preprocessing, LDA achieves the best results among all models, while SVM achieves the second best results. For TF-IDF preprocessing, SVM achieves the best results, while Logistic Regression achieves the second best.

vectors, we train the LDA model to find the linear decision boundary that separates the two groups.

For the Word2Vec preprocessed data, we achieved an accuracy of around 87.19%, F1 score of around 0.8733, and AUC of 0.9414. The results, to our surprise, are higher than all other models that we used.

For the TF-IDF preprocessed data, LDA also achieves a decent result of an accuracy of 87.11%, F1 score of 0.8710, and AUC of 0.9445.

3.7 QDA

QDA (Quadratic Discriminant Analysis) closely replicates LDA in that the measurements are assumed to be normally distributed, but for QDA, we do not assume that the mean and covariance of each of the classes are equal. Thus, we must calculate it separately.

It is worth noting that LDA is a lot less flexible than QDA so it results in a lower variance. However, due to bias-variance tradeoff, LDA can also suffer from a higher bias at the same time. Therefore, LDA tends to be better when there is a smaller training data set, while QDA performs better with a larger training data set [17].

For the Word2Vec preprocessed data, we achieved an accuracy of 78.53%, F1 score of 0.7741, and AUC of 0.8701. The results were surprisingly lower than LDA, considering that we had a rather large training data set.

For the TF-IDF preprocessed data, we achieved an accuracy of 81.39%, F1 score of 0.8137, and AUC of 0.8848.

4 Results and Final Remarks

Our comprehensive text analysis of IMDb reviews aimed to understand the correlation between movie or TV show reviews and their corresponding sentiment.

The preprocessing steps significantly improved the input quality for TF-IDF and Word2Vec vectorizers. We explored the impact of stemming and lemmatization on model performance and opted for lemmatization because of its ability to group synonyms, which is needed for both Word2Vec's contextualization and TF-IDF's vocabulary.

We employed two techniques for vectorizing the reviews: Word2Vec and TF-IDF matrices. Word2Vec is great for semantic understanding and similarity measurements, but is computationally expensive when encoding large pieces of text and can be difficult to interpret. On the other hand, a TF-IDF matrix represents term importance across documents and is easy to interpret, but can struggle with larger vocabularies and understanding term contextualization.

Seeing as a TF-IDF matrix had 32,537 features even after L2 regularization and idf score filtering, we also applied principal component analysis for dimension reduction. While it did not have much impact on model performance, it greatly improved the runtime for models that struggle in high-dimensional spaces, such as KNN.

For Word2Vec preprocessing, LDA achieves the best results among all models, while SVM achieves the second best results. For TF-IDF preprocessing, SVM achieves the best results, while Logistic Regression achieves the second best. Overall, we believe that SVM model, specifically with linear kernels, is the most suit-

able for the IMDb Dataset. Another interesting point that we would like to note is the high performance of linear models in the downstream task of classifying review sentiments. This might be because linear models introduce the least variance and thus prevents overfitting. Furthermore, we highlight the weak performance of KNN, where its performance is greatly affected by the high dimensions (500 columns for both W2V and TF-IDF) of our data. For more information, refer to table 2 and appendix section A.

Our model is successful at determining whether an IMDb review is positive or negative, helping IMDb and users understand the consensus on a movie or TV show. And this isn't just about sentiment; our model also helps IMDb in recommending media to its users. With so many reviews pouring in every day, our model stands out by picking up on important words in reviews (TF-IDF) and spotting similarities between them (Word2Vec). It's a practical tool for IMDb, giving them a closer look at what people are saying and helping users find the right stuff to watch.

References

[1] Kurtis Pykes. Stemming and lemmatization in python, Feb 2023.

[2] Mark Hasegawa-Johnson. Lecture 38 –tf/idfand information retrieval.

[3] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

[4] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23:550–560, 1997.

[5] Mark W. Schmidt, Nicolas Le Roux, and Francis R. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162:83 – 112, 2013.

[6] Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Neural Information Processing Systems*, 2014.

[7] Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85:41–75, 2011.

[8] Wikipedia. Decision tree learning — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Decision%20tree%20learning&oldid=1187062781`, 2023. [Online; accessed 08-December-2023].

[9] L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. Classification and regression trees. *Biometrics*, 40:874, 1984.

[10] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[11] Trevor J. Hastie, Robert Tibshirani, and Jerome H. Friedman. The elements of statistical learning. 2001.

[12] Bianca Zadrozny and Charles Peter Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *International Conference on Machine Learning*, 2001.

[13] Bianca Zadrozny and Charles Peter Elkan. Transforming classifier scores into accurate multiclass probability estimates. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[14] John Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. 1999.

[15] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. *Proceedings of the 22nd international conference on Machine learning*, 2005.

[16] Wikipedia. Linear discriminant analysis — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Linear%20discriminant%20analysis&oldid=1188796231`, 2023. [Online; accessed 06-December-2023].

[17] Linear and quadratic discriminant analysis · afit data science lab r programming guide.

A ROC Curves for all 7 Models

Left indicates the ROC curve for model trained with Word2Vec preprocessed data, right indicates model trained with TF-IDF preprocessed data (after PCA).

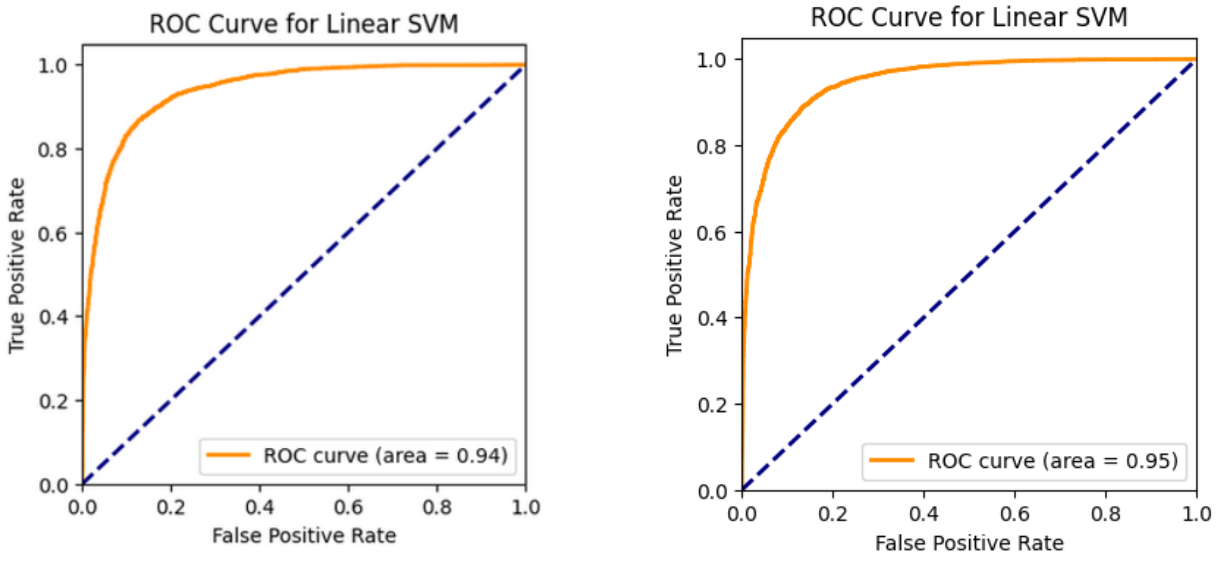


Figure 3: **ROC Curve for Linear SVM.**

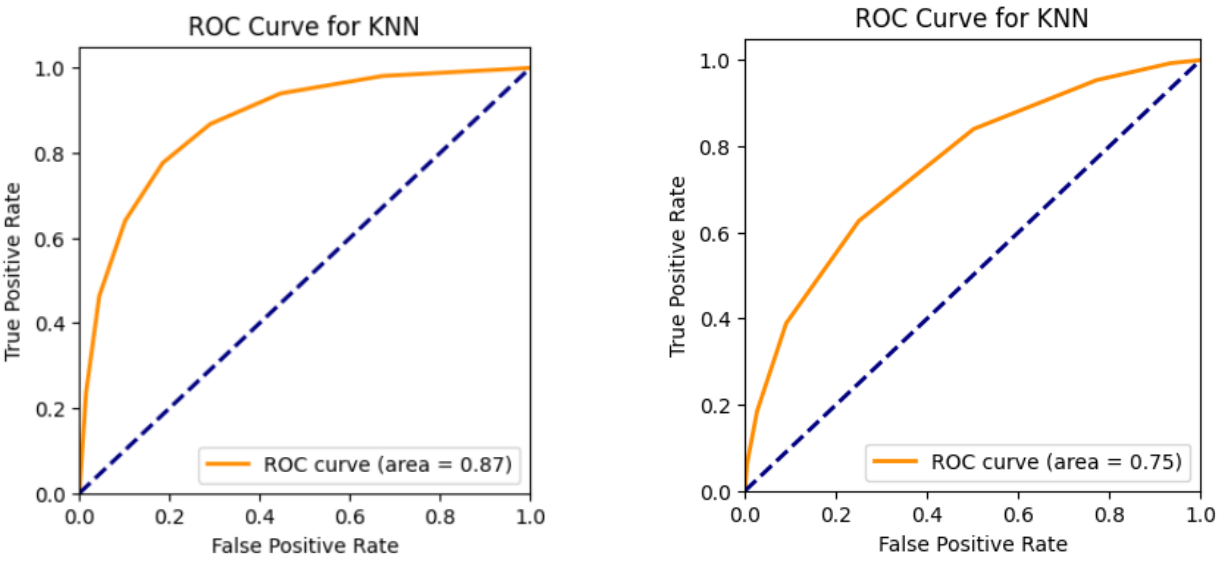


Figure 4: **ROC Curve for KNN.**

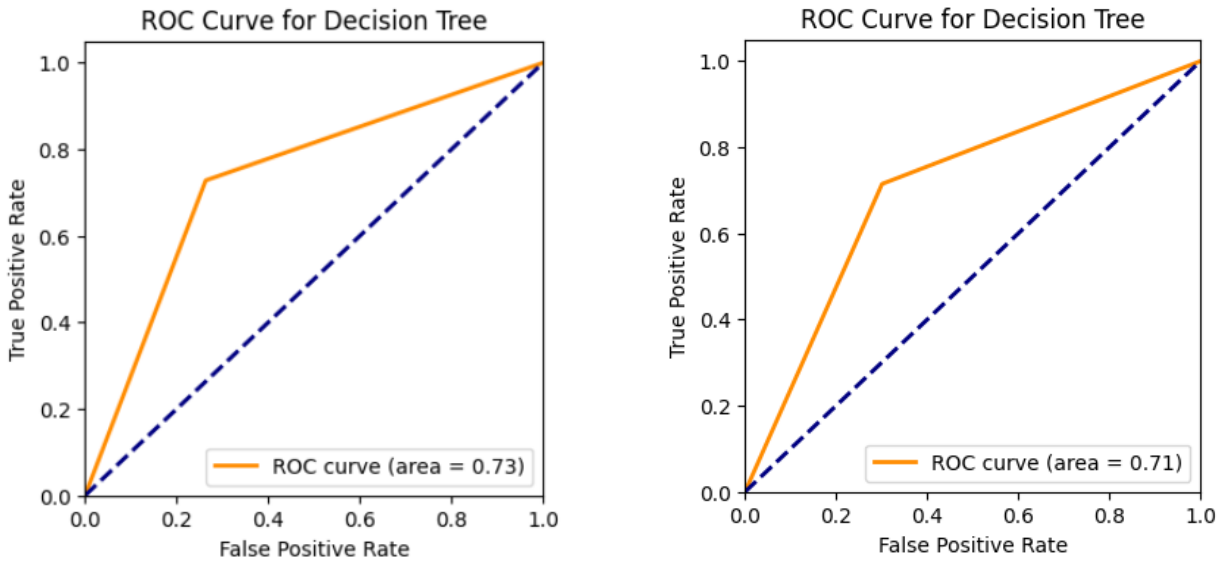


Figure 5: **ROC Curve for Decision Tree.**

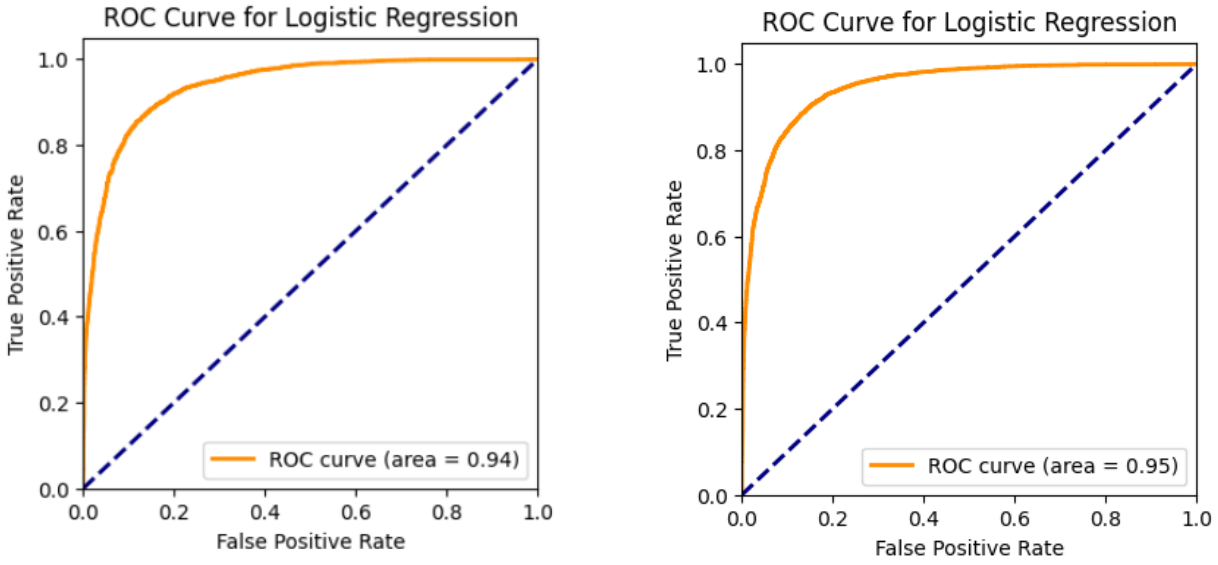


Figure 6: **ROC Curve for Logistic Regression.**

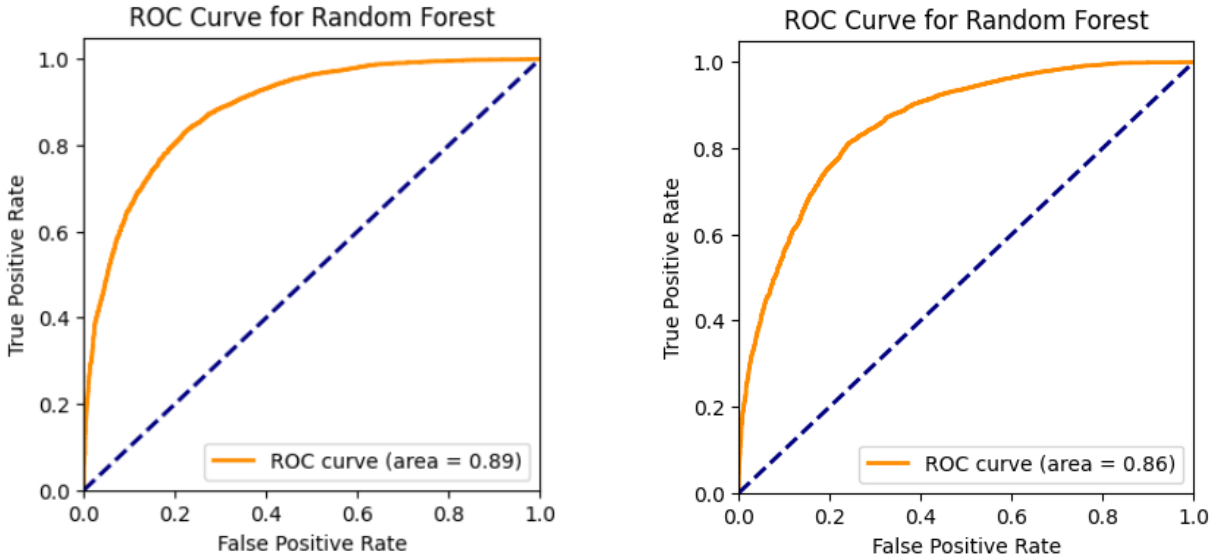


Figure 7: **ROC Curve for Random Forest.**

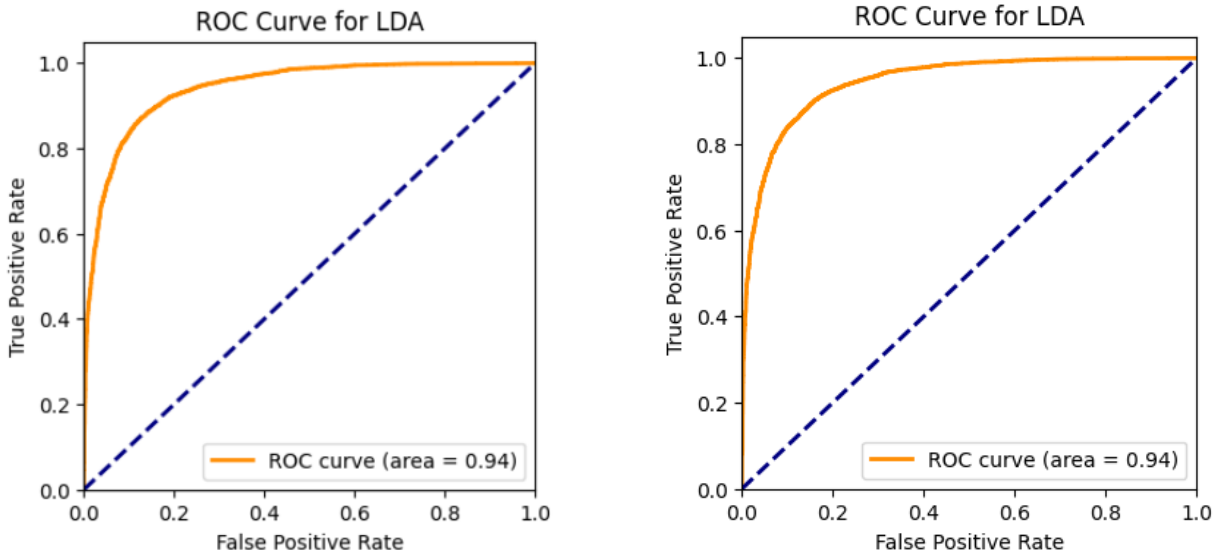


Figure 8: **ROC Curve for LDA.**

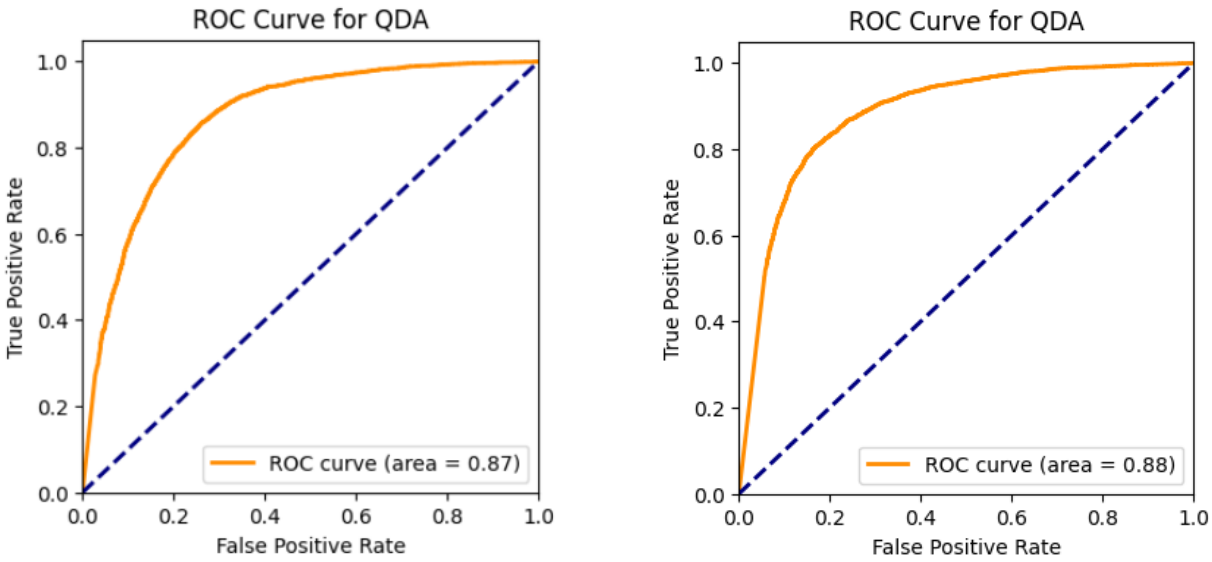


Figure 9: **ROC Curve for QDA.**