

**FRONT COVER**

**CS 178 Final Project**

| **Names** | **Mia Linh Nguyen**  **Jenny Rumondang Pasaribu Rachael Vi Le** |
| --- | --- |
| **IDs** | **56967962, 82024624, 61837377** |
| **Dataset** | **IMDB** |
| **Classification Methods** | **K Nearest Neighbors, Logistic, Feedforward Neural Networks, Multinomial Naive Bayes** |

**Dataset: IMDB**

**Classification Methods:** K Nearest Neighbors, Logistic, Feedforward Neural Networks, Multinomial Naive Bayes

**Authors:** Mia Linh Nguyen, Jenny Rumondang Pasaribu, Rachael Vi Le

**Team Name**: MJR

**June 8th, 2026**

**Collaborators:** None

# **Summary**

For the project, we analyzed the performance of different classifiers on the IMDB Movie Reviews dataset, which contains 50,000 reviews from the Internet Movie Database that are labeled positive or negative. The classification methods that we utilized are K Nearest Neighbor, Logistic, Neural Networks, and multinomial Naive Bayes. From employing a variety of different models, we can conclude that the logistic classifier performed the best in predicting whether a review is positive or negative, with the highest testing accuracy of 84.68%. Multinomial Naive Bayes and Feedforward Neural Networks had high testing accuracies with 82.70% and 83.27%, respectively. The K Nearest Neighbor classifier could not adjust as well to the data and only had a testing accuracy of 73.08%.

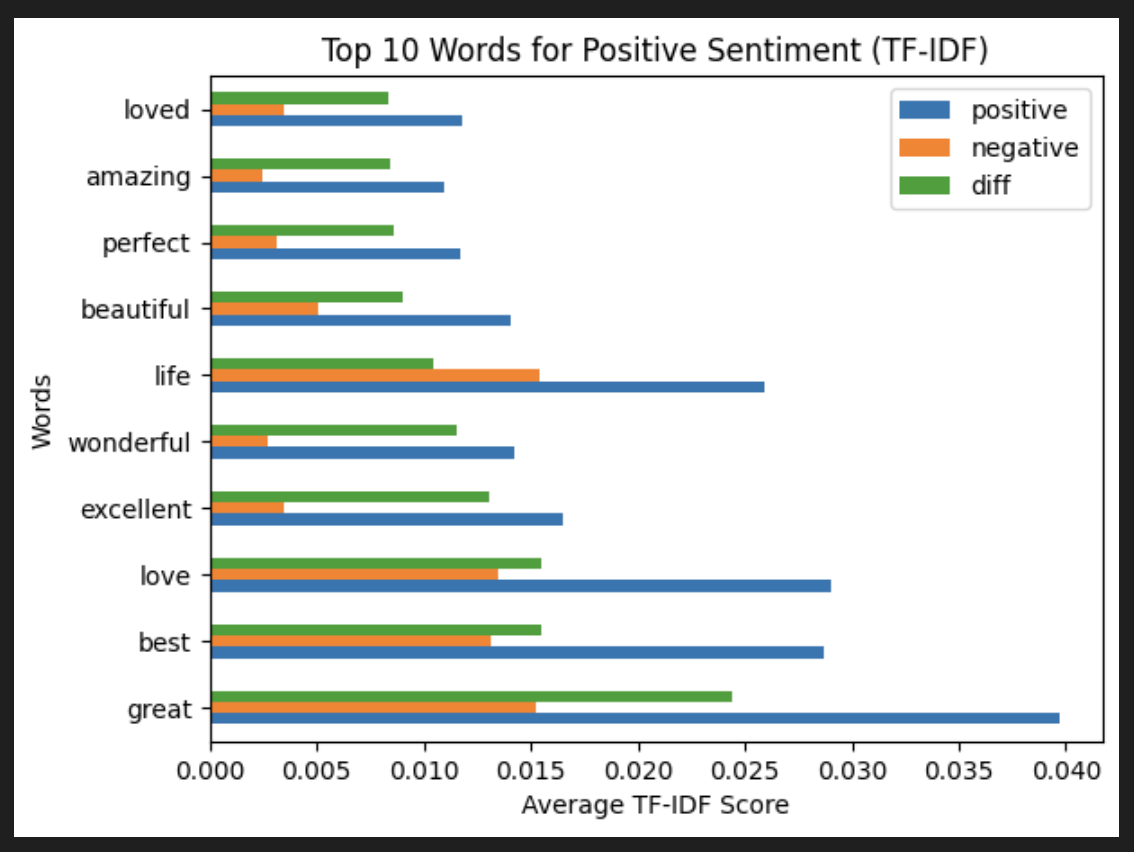
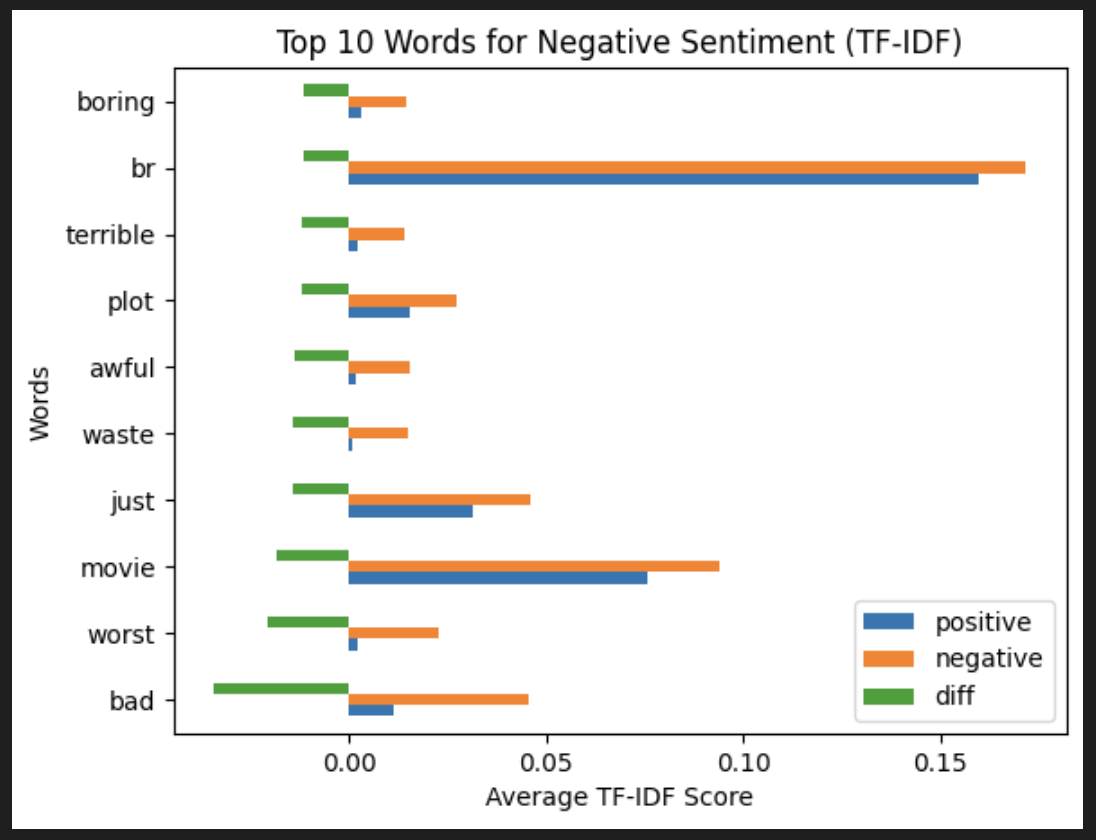
# Data Description

The IMDB Movie Reviews dataset contains 50,000 movie reviews that is widely used for sentiment analysis and text classification tasks. Each review is a text document, and each document is labeled as either positive (1) or negative (0).

The dataset consists of the following splits:

* Training set: 25,000 labeled reviews (12,500 positive, 12,500 negative)
* Test set: 25,000 labeled reviews (12,500 positive, 12,500 negative)
* Unsupervised set: 50,000 unlabeled reviews

To analyze the IMDb dataset, we need to convert the raw text into numerical representations that can be used by machine learning models. Some techniques that can be employed are bag-of-words, TF-IDF vectorization, and word embeddings. We chose the TF-IDF vectorization technique, as it better weighs the importance of words and typically delivers better results for sentiment analysis compared to BoW, while maintaining low computational cost. Using scikit-learn's TfidfVectorizer, we transformed the text data into a sparse matrix of TF-IDF scores, representing the significance of tokens in each review. Some exploration we did included getting the token count per review, getting the top 20 words overall, and finding the top 10 words for each kind of sentiment (see more in Figures 1 and 2 in the Appendix).



One paper we looked at that also used the IMDb movie review dataset is *Anytime Active Learning* by Ramirez-Loaiza et al. (2020). The authors discuss how it is possible to train classifiers using only part of the input, which helps make faster predictions without needing as much data. This idea influenced our decision to remove stopwords during preprocessing to shrink the feature space and make the model more efficient. It allowed us to focus on the words that actually carry sentiment, which helped keep things simple without hurting performance.

# Classifiers

**K Nearest Neighbors (KNN):** This method calculates the distance from a given point to other points in the training set and assigns the new point the majority label of its K nearest neighbors. The primary hyperparameter is K, which determines the number of neighbors considered when assigning a label. In our experiments, we will test odd values of K in the range from 1 to 121. We will use **scikit-learn** for training and testing the classifier and **matplotlib** for visualization.

**Logistic Classifier:** This method fits a linear decision boundary to separate classes by estimating the probability that an input belongs to a certain class using a logistic function. It is effective for linearly separable, high-dimensional data. For this classifier, we used **scikit-learn's LogisticRegression** class.

Hyperparameters and settings:

* Penalty: investigated the ‘l1’ and ‘l2’ penalty
* Solver: investigated the ‘liblinear’ and ‘saga’ solvers
* Fit\_intercept: investigated setting to true and false
* C: investigated multiple C values from [0, 0.1, 1, 10, 50, 100, 200, 500, 1000]

**Feedforward Neural Networks Classifier (or MLP classifier)**: This method uses a network inspired by the human brain, consisting of layers of neurons that process and transform input data through activation functions. Data flows from the input layer, through hidden layers where patterns are learned, and to the output layer, where predictions (e.g., sentiment classification) are made. The network is trained using backpropagation, adjusting neuron weights to minimize errors. This process repeats until optimal weights are learned. For this classifier, we used **scikit-learn's MLPClassifier**.

Hyperparameters Investigated:

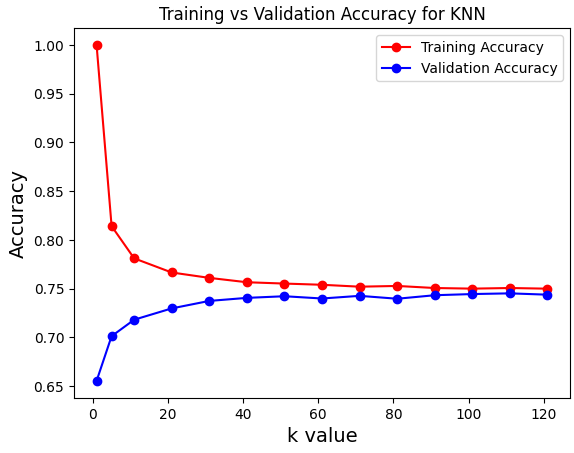
* Hidden Layer Sizes: Tested various architectures, including: (64,), (64, 128), (256, 256), and (64, 64, 64) (for instance, (64, 64, 64) means 3 hidden layers with 64 hidden nodes per layer)
* Activation functions: ReLU, Logistic
* Solvers (optimization techniques): Adam, SGD (Stochastic Gradient Descent)
* Learning rate type: Constant, Adaptive
* Learning rate initializers: 0.001, 0.01, 0.1, and 1.5

**Multinomial Naive Bayes (MNB):** This classifier is widely used for text classification and works well with the tf-idf scores we set up. It assumes word independence and assigns a review to the class with the highest probability based on word counts. The key hyperparameter is alpha, which is tested in the range of 0.01–2. The classifier is implemented and evaluated using **scikit-learn** and visualized with **matplotlib**.

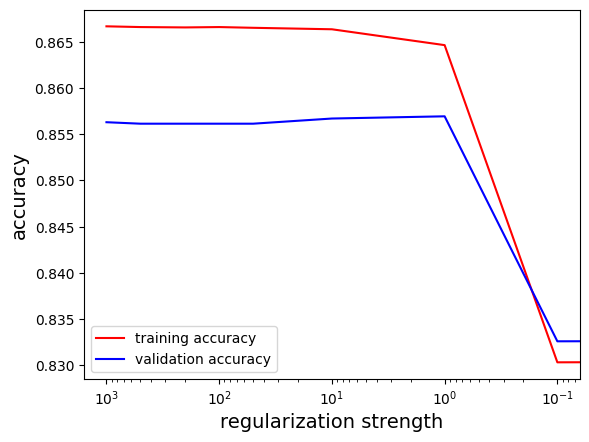
# Experimental Setup

**Experimental methodology:** Each of the classifier models we investigated has specific hyperparameters that can be tuned, and we will explore how those hyperparameters affect the model performance using several metrics: (1) classification accuracy, (2) confusion matrix, and (3) precision and recall.

**Data partitioning:** To ensure the model's performance is evaluated effectively, the original dataset is partitioned into training, validation, and testing sets, with a ratio of 5:2.5:2.5, respectively. We will train our model using the training set, then tune the hyperparameters using the validation set. After the best hyperparameter configuration has been determined, the testing set will be used at the end to verify the model’s performance on unseen data.

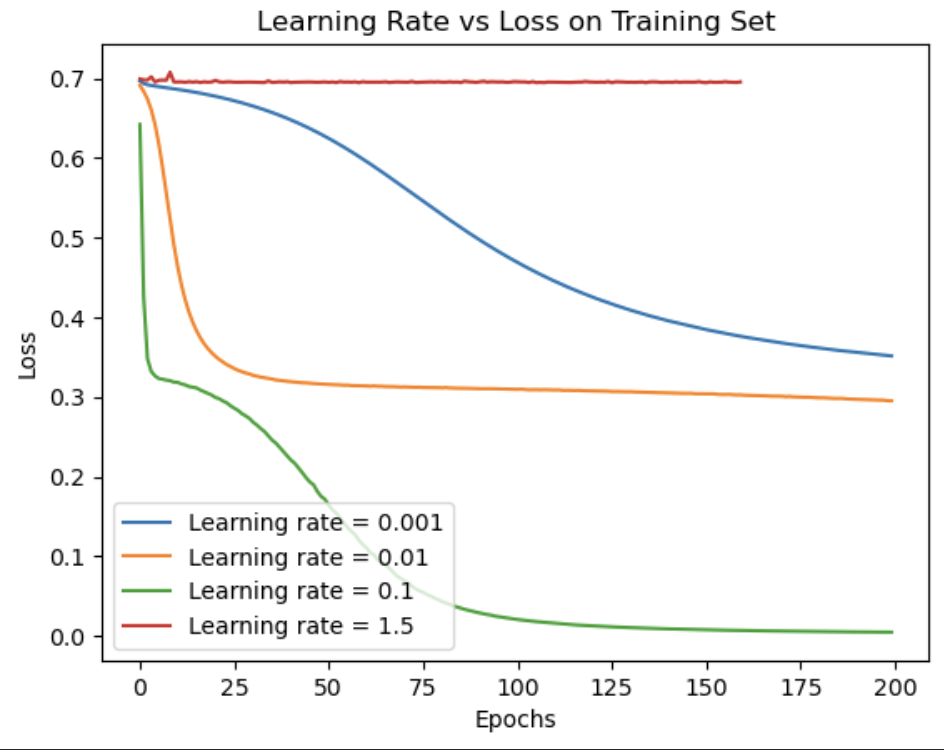
**Hyperparameter Selection:** For each model, we tested how varying training size affects our model performance, and different values of training size were chosen from [50, 500, 2000, 5000, 10000, 20000, 25000]. Additionally, the specific hyperparameter values for each model are tested and selected as below: 

1. **K Nearest Neighbors (KNN)** - Different values of k used to train the model were selected from k\_vals = [1, 5, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111, 121]. Other parameters were kept at default values specified by scikit-learn.



1. **Logistic Classifier** - Different values of the regularization strength parameter C were tested to observe the impact on model performance. Specifically, values were chosen from [0.01, 0.1, 1, 10, 50, 100, 200, 500, 1000]. As you can see in the graph, the accuracy for both the training and validation data plateaus around 50, which is what C is eventually set to. The penalty type was set to 'l1' to promote sparsity in the learned weights, the solver was set to 'liblinear' to support L1 regularization, and fit\_intercept was set to true to include a bias term in the model.



1. **Feedforward Neural Networks Classifier**- 

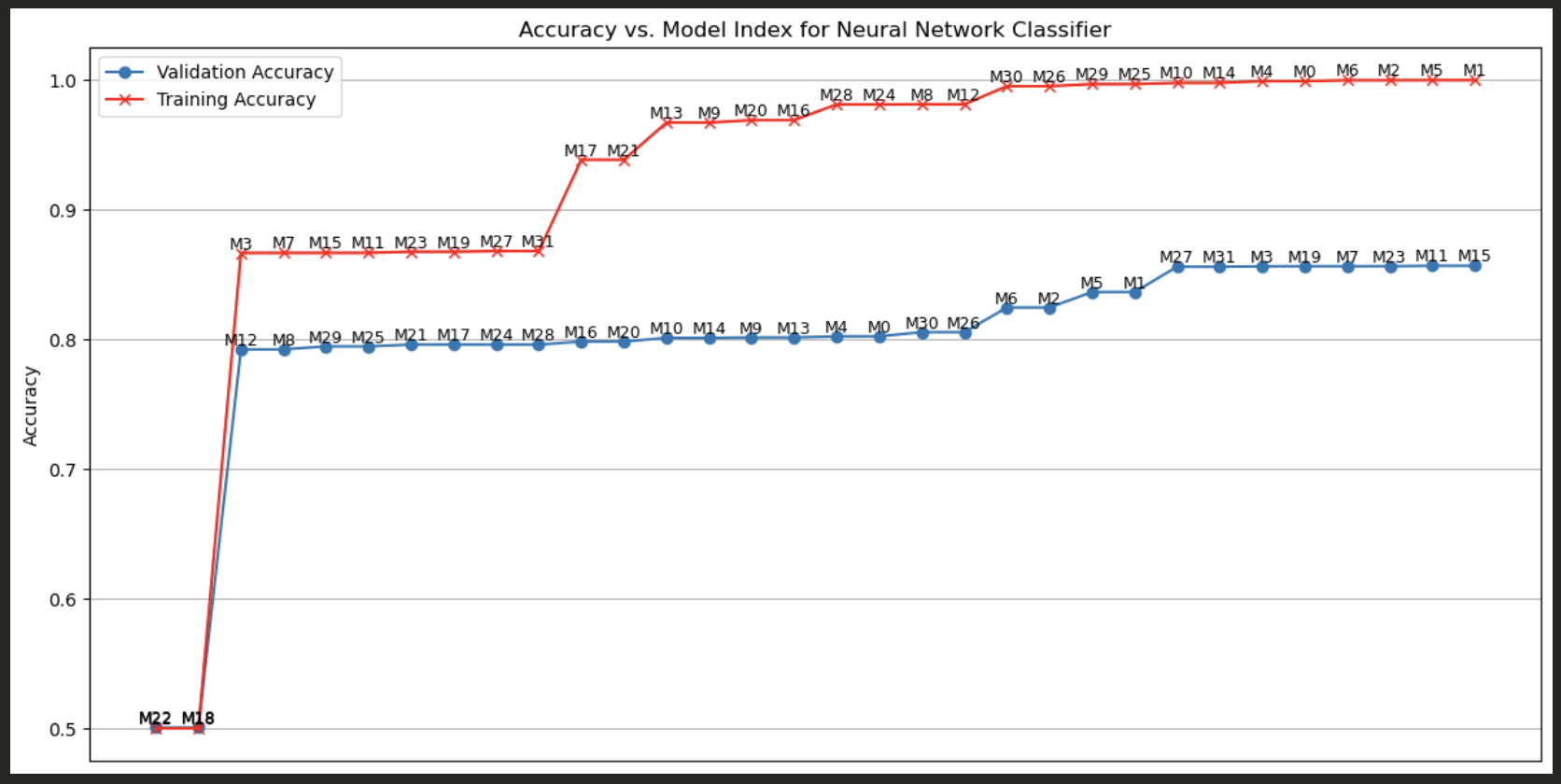
First, the **learning rate** values were tested with values chosen from [0.001, 0.01, 0.1, 1.5] (using a neural network with 1 hidden layer and 64 hidden nodes, ReLU activation function, and stochastic gradient descent (SGD) as a solver). This experiment is shown in the following chart, which indicates that the most suitable learning rate is **0.1**, as it allows the model to converge and results in a low loss.

Different combinations of the following hyperparameter values were tested to optimize model performance:

* Hidden Layer Sizes: (64,), (64, 128), (256, 256), and (64, 64, 64)
* Activation Functions: ReLU, Logistic
* Solvers: Adam, SGD (Stochastic Gradient Descent)
* Learning Rates: Constant, Adaptive
* Learning Rate Initializers: 0.1 (chosen from the previous experiment)
* n\_iter\_no\_change=100
* max\_iter=200
* Batch\_size = 256

The results of these configuration combinations are listed in Table 1 (see full version in the appendix). The parameters that gave us the highest validation accuracy are:

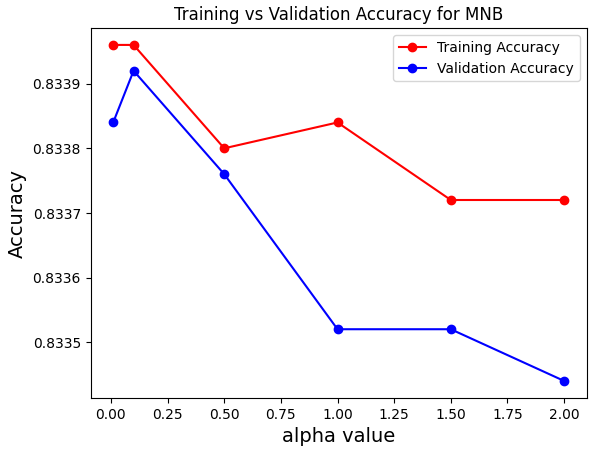
| **index** | **hidden\_layer\_size** | **learning\_rate** | **activation** | **solver** | **train\_accuracy** | **val\_accuracy** |
| --- | --- | --- | --- | --- | --- | --- |
| … | | | | | | |
| M15 | (64, 128) | adaptive | logistic | sgd | 0.8668 | 0.8568 |
| … | | | | | | |

**Table 1 (shortened):** Training and Validation Accuracy Rates for Different MLP Classifier Configurations

In general, we want a model that results in a high validation accuracy to avoid overfitting on the training set. Based on Figure 5, configuration M15 is the best one.

Using **M15** to predict labels for the validation set, we get the following result:

* Validation accuracy = 0.8568
* Precision = 0.8502
* Recall = 0.8665

This result is based on the standard threshold of 0.5 (the cutoff value that determines how predicted probabilities are converted into class labels). As the threshold increases, precision increases and recall decreases (see **Figures 3** and **4** in the Appendix). Based on the context of sentiment classification, we wanted to prioritize precision, so we increased the threshold to 0.7. The result of this will be shown in the Experimental Results section.

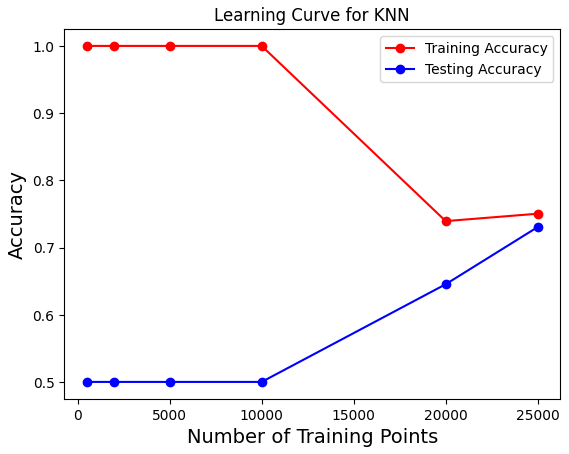
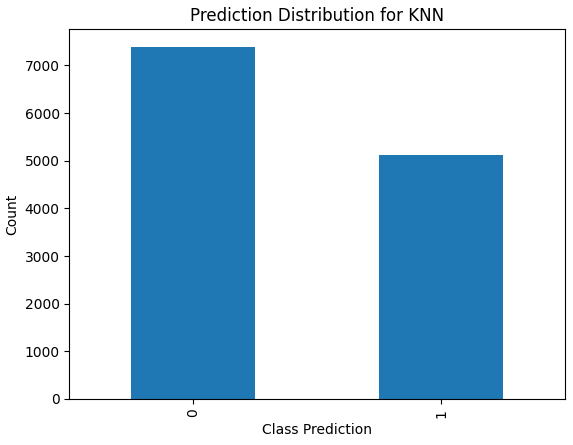
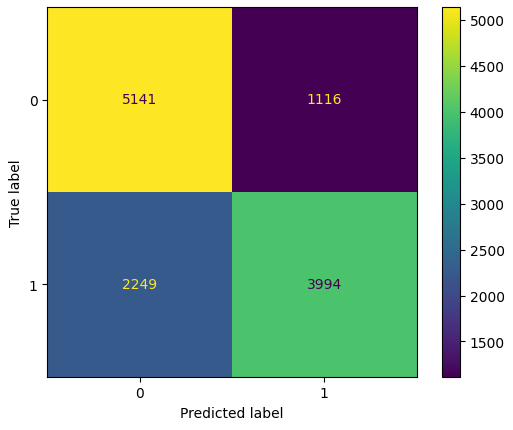
1. **Multinomial Naive Bayes (MNB):** Values of alpha used to train the model were selected from alpha\_vals = [.01, .1, .5, 1, 1.50, 2]. Other parameters were kept at default values specified by scikit-learn.

# 

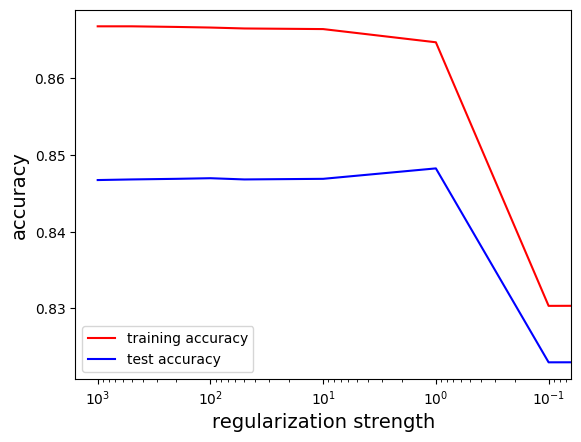
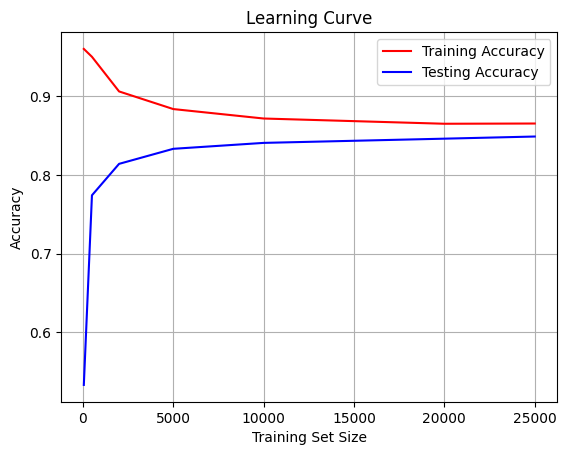
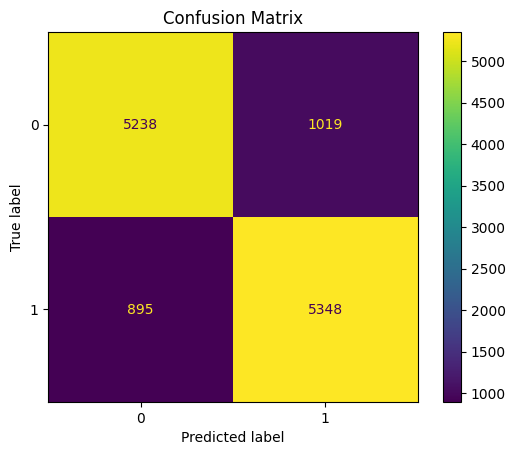
# 

# Experimental Results

**kNN Classifier**



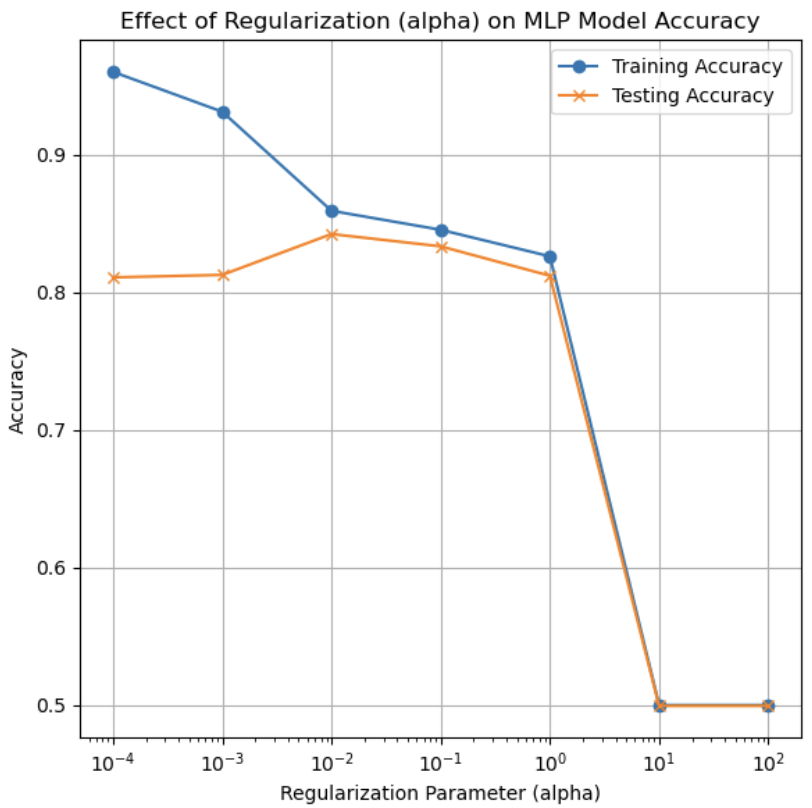
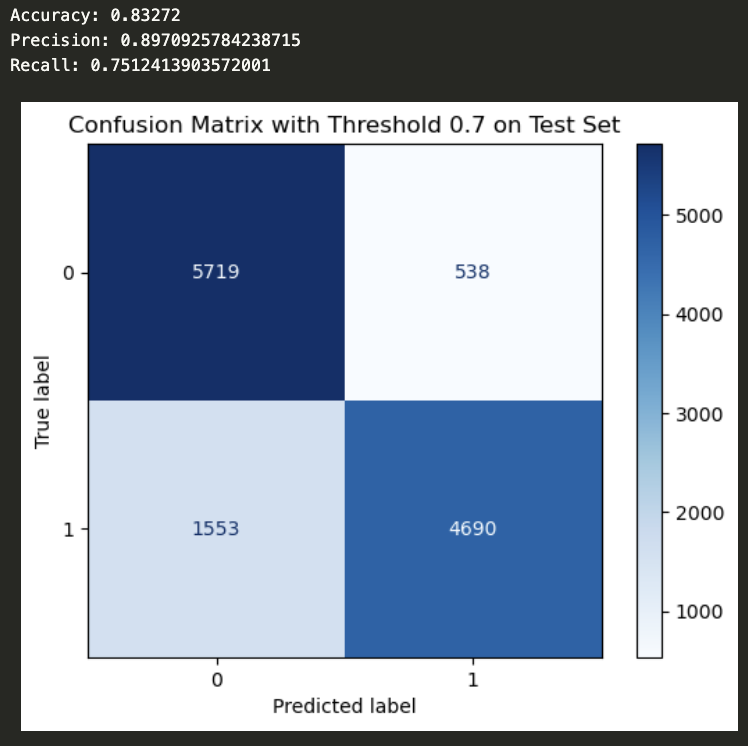
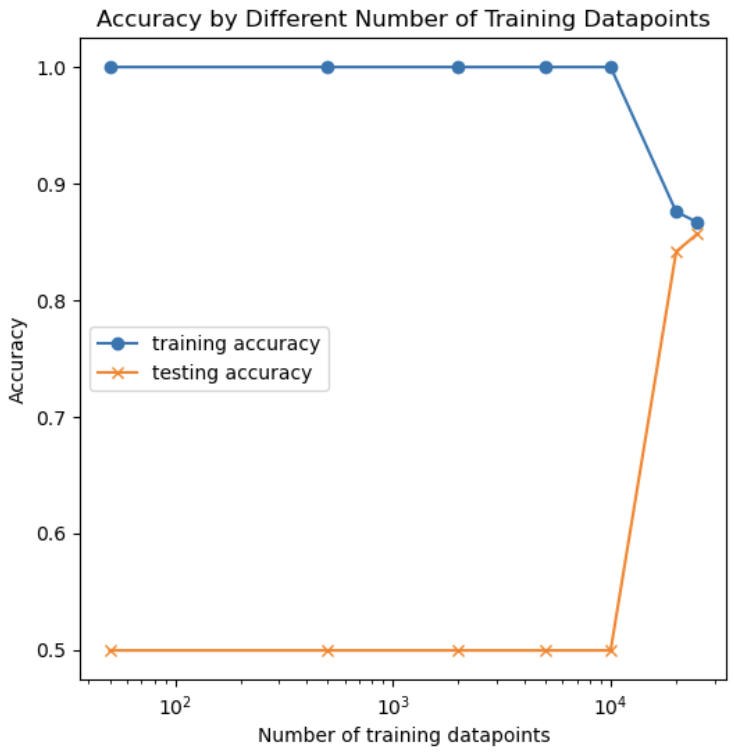
The K Nearest Neighbor classifier had significant drops in training accuracy as K increased, while validation accuracy went up. However, the accuracies for the two sets similarly flattened out at K=60. The K value that produced the best validation accuracy and that we used to produce these results is K = 111. For the learning curve, it can be seen that the more training data points allow for the test accuracy to be higher, while the train accuracy starts to decrease. An interesting aspect to note is that this classifier overpredicted the label 0 as the difference between the amount of predicted label 0 and predicted label 1 is 2,280. This shows that the classifier had the tendency to predict label 0 more than label 1.

**Logistic Classifier**

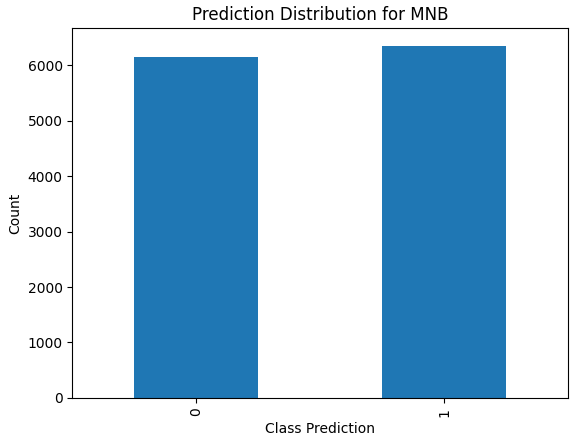
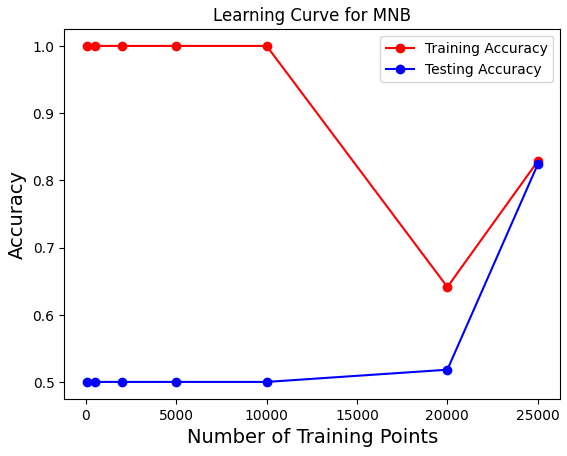
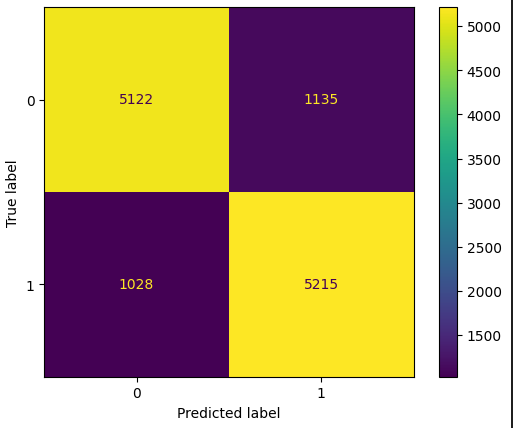




The logistic classifier performs well on the test data and achieves solid precision and recall. We can see from the learning curve that both training and test accuracy improve with more data, suggesting the model benefits from larger datasets. The confusion matrix confirms that the classifier maintains good balance in identifying both classes, although there is room for improvement in reducing false positives and false negatives. Additionally, the regularization experiment highlights the importance of tuning C to avoid underfitting or overfitting in both the training and test data as less regularization leads to a dramatic decrease in accuracy.

**Feedforward Neural Networks Classifier**

The neural network classifier performs well on the test data, achieving the highest precision score among all models. The learning curve reveals significant overfitting when the training data size is less than 20,000. To mitigate overfitting, we can increase the training size, but care must be taken not to increase it excessively, as this could introduce higher variance. Additionally, our regularization experiment shows how adjusting the regularization parameter (alpha) can help reduce overfitting by increasing bias slightly, leading to a better bias-variance tradeoff. The confusion matrix shows that our model slightly favors correctly predicting negative sentiment reviews over positive ones. This aligns with our decision to increase the threshold for classifying a review as positive, which increases confidence in the accuracy of positive predictions. This threshold adjustment can be fine-tuned based on the specific goals of the model.

**Multinomial Naive Bayes Classifier**



From testing, we noticed that it was better to keep alpha values low and relatively close to zero. The alpha value that gave us the best validation accuracy was 0.1, and we used that value to calculate the other results. For the learning curve, we can see that training accuracy was flat but then dropped drastically. On the other hand, the testing accuracy increased while the training accuracy decreased. An interesting thing to note would be that the two accuracies meet when the data amount for training reaches 25,000, as training accuracy begins to increase again. The classifier pretty evenly predicted the labels, with only a difference of 200.

# Insights

This project has provided valuable experience in transforming text documents into usable representations for various models. From previous classwork, we’ve used the Bag Of Words method, but we decided to choose a new approach to capture more nuances in each review, aiming for more accurate results. Through this process, we can see how preprocessing the data plays an essential role in affecting the results we get.

**KNN Classifier**

Through experimenting with the kNN classifier, we found that training and validation accuracies behave oppositely as the K-value changes. A higher K-value results in lower training accuracy but higher validation accuracy, eventually stabilizing. This highlights how certain models can overfit the training data depending on the chosen hyperparameters. It is crucial to focus on testing accuracy rather than just improving training accuracy. We also observed that the kNN classifier had low recall but high precision, indicating that it struggles to identify positive reviews, mistakenly classifying them as negative. Additionally, increasing the training set size helped reduce overfitting. In a real-world setting, such as the entire IMDb review database, kNN would be computationally expensive due to the need to calculate distances for every point and would likely fail as the feature space expands, suffering from the curse of dimensionality.

**Logistic Classifier**

The project showed how logistic regression works as a basic classifier with 86.65% accuracy on the training data, 84.68% accuracy on the test data, precision at 0.8398, and recall at 0.8567. Initially, the learning curve indicated overfitting with small datasets, but both training and test accuracies improved with larger datasets. Varying the regularization strength showed that too little regularization hurt performance, while too much regularization around 50 plateaued. An error analysis showed a balanced trade-off between false positives and false negatives, especially with data containing common or weak features, which even humans might misclassify, so linear models had trouble predicting them well. Logistic regression is a useful tool in practical settings like medical diagnosis or spam filtering due to its simplicity and interpretability. However, it may struggle with more complex, nonlinear problems where more advanced models are needed for better performance.

**Feedforward Neural Networks Classifier**

Through the neural network experiments, we learned how hyperparameters like learning rate, hidden layer sizes, and activation functions significantly affect model performance, with more hyperparameters providing flexibility but requiring careful tuning to avoid overfitting or underfitting. A surprising finding was that the testing accuracy remained around 50% when the training set size was under 20,000, suggesting significant overfitting with insufficient data. I also discovered that adjusting the decision threshold can impact precision and recall, allowing for a trade-off based on the desired model outcome. Additionally, the importance of regularization and data size became evident in ensuring the model generalizes well, emphasizing the need for a balanced approach to both model complexity and data quality. These insights emphasize the importance of sufficient data and careful hyperparameter tuning to optimize the model's effectiveness in real-world applications.

**Multinomial Naive Bayes Classifier**

This project provided valuable experience with the Naive Bayes classifier and its performance in text classification. This model can leverage word frequencies to generate probabilities for determining if a review is positive or negative. It significantly benefits from having a larger training set, as the testing accuracy improved drastically when the training set size reached 20,000 data points. Interestingly, the model predicts positive and negative reviews with a relatively even distribution, showing no tendency to overpredict one class. Furthermore, the model achieved almost equal precision and recall, which is interesting as other models typically have to compromise on one to benefit the other. For a real-world application, the strengths of this model are that it does not suffer from the curse of dimensionality that other classifiers struggle with and performs well on TF-IDF, which weighs the importance of words based on their frequency. Additionally, the model is quick to train, as it has fewer hyperparameters to tune.

# Contributions

* Jenny Pasaribu:
  + Performed experiments on the dataset using the logistic regression model
  + Analyzed and summarized a published paper related to the IMDB dataset
* Mia Linh Nguyen:
  + Perform data exploration on the IMDB movie review dataset
  + Perform experiments on the dataset using the feedforward neural network model
* Rachael Le:
  + Performed experiments on the dataset using the K Nearest Neighbor and Multinomial Naive Bayes classifiers

# Appendix

**Citations**

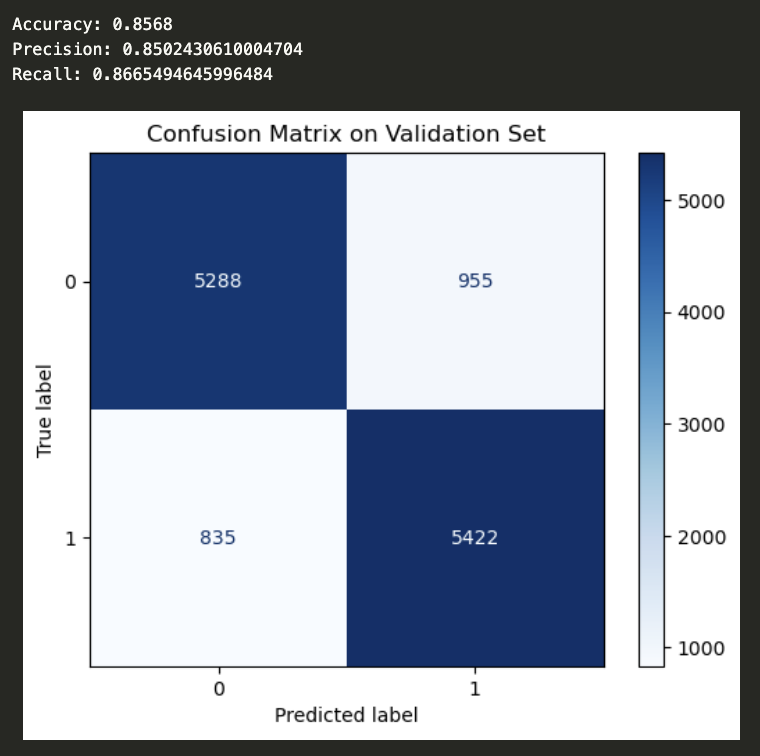
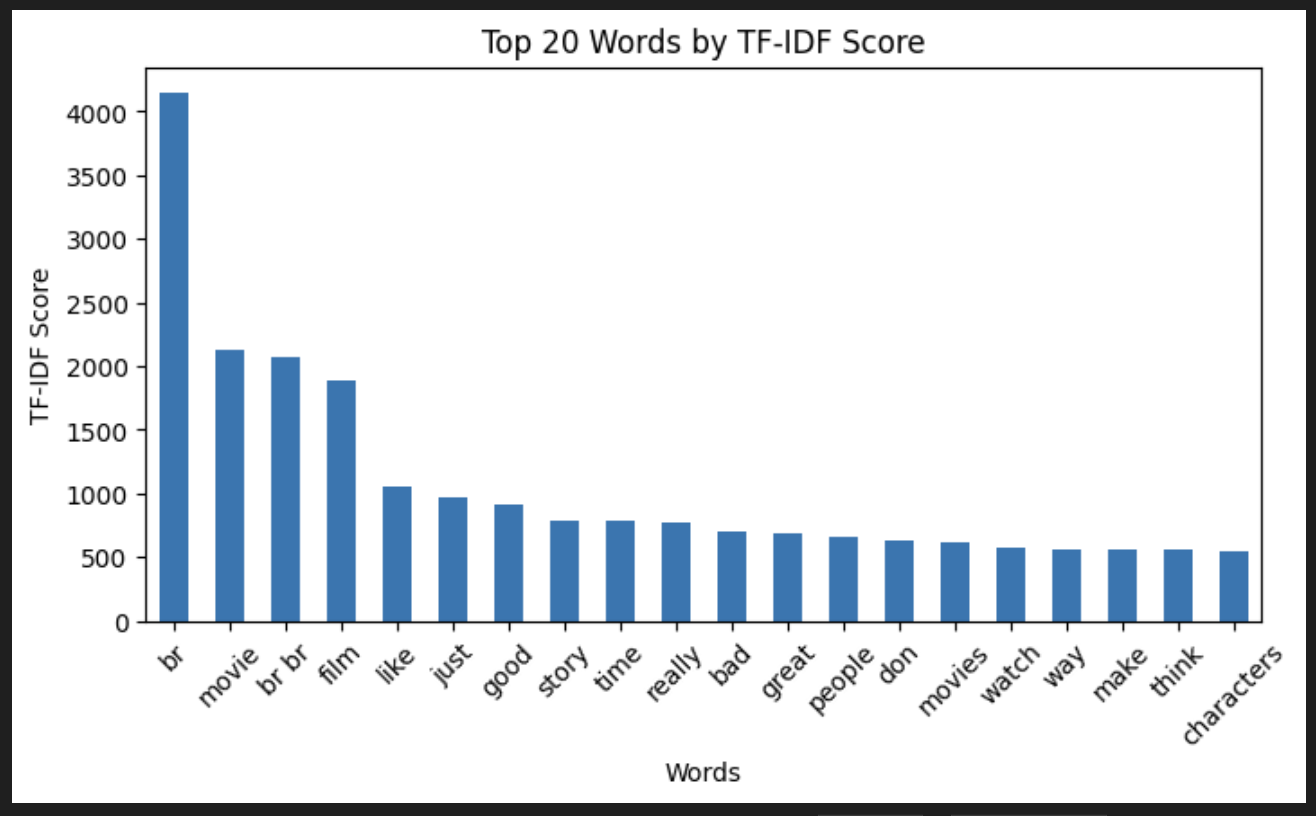
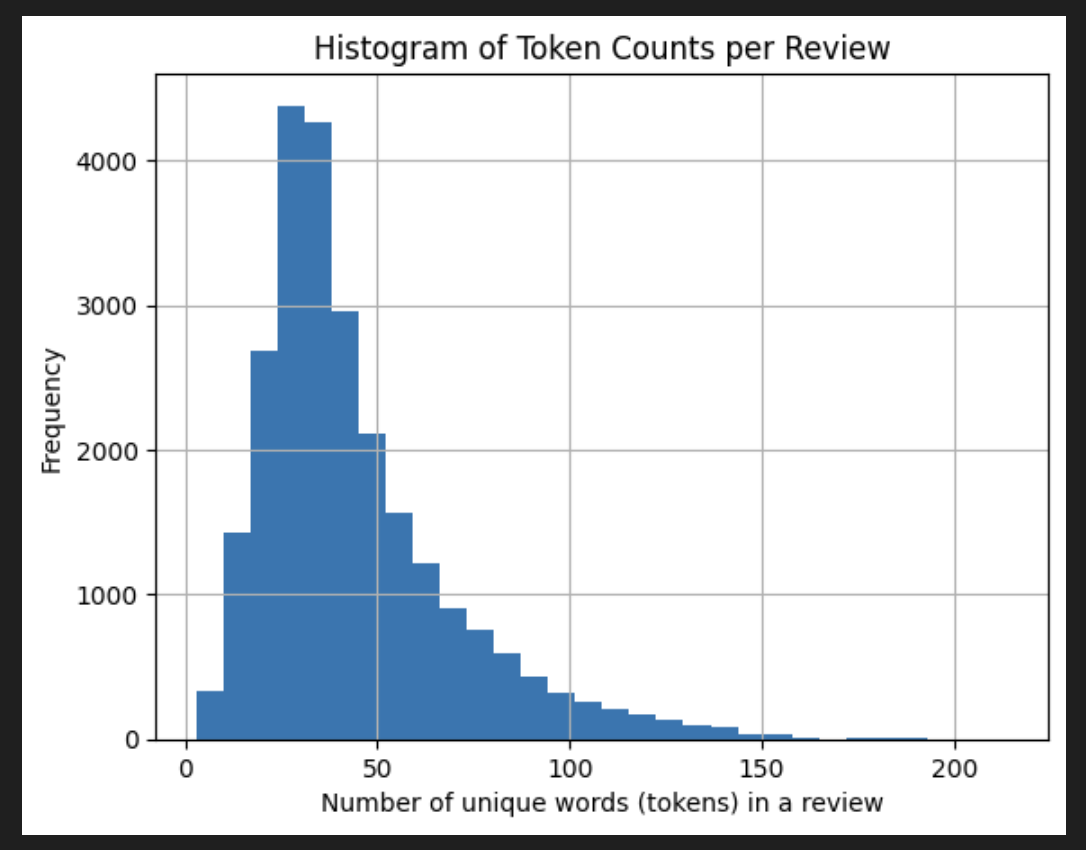
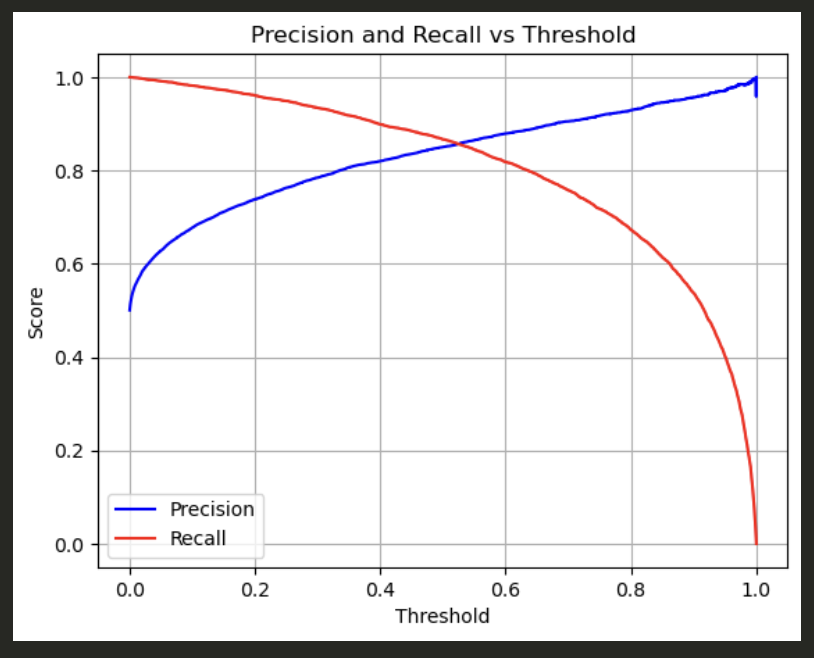
Ramirez-Loaiza, Maria E., Aron Culotta, and Mustafa Bilgic. *Anytime Active Learning*.

Proceedings of the 28th ACM International Conference on Artificial Intelligence, 2020, pp. 2048-2054.<https://paperswithcode.com/paper/anytime-active-learning>.

**Extra Diagrams:**

| **index** | **hidden\_layer\_size** | **learning\_rate** | **activation** | **solver** | **train\_accuracy** | **val\_accuracy** |
| --- | --- | --- | --- | --- | --- | --- |
| M0 | (64,) | constant | relu | adam | 0.99924 | 0.8024 |
| M1 | (64,) | constant | relu | sgd | 1 | 0.83656 |
| M2 | (64,) | constant | logistic | adam | 0.99992 | 0.82464 |
| M3 | (64,) | constant | logistic | sgd | 0.86672 | 0.8564 |
| M4 | (64,) | adaptive | relu | adam | 0.99924 | 0.8024 |
| M5 | (64,) | adaptive | relu | sgd | 1 | 0.83656 |
| M6 | (64,) | adaptive | logistic | adam | 0.99992 | 0.82464 |
| M7 | (64,) | adaptive | logistic | sgd | 0.86676 | 0.85648 |
| M8 | (64, 128) | constant | relu | adam | 0.9814 | 0.79232 |
| M9 | (64, 128) | constant | relu | sgd | 0.96724 | 0.80152 |
| M10 | (64, 128) | constant | logistic | adam | 0.99788 | 0.8012 |
| M11 | (64, 128) | constant | logistic | sgd | 0.8668 | 0.8568 |
| M12 | (64, 128) | adaptive | relu | adam | 0.9814 | 0.79232 |
| M13 | (64, 128) | adaptive | relu | sgd | 0.96724 | 0.80152 |
| M14 | (64, 128) | adaptive | logistic | adam | 0.99788 | 0.8012 |
| M15 | (64, 128) | adaptive | logistic | sgd | 0.8668 | 0.8568 |
| M16 | (256, 256) | constant | relu | adam | 0.96916 | 0.7984 |
| M17 | (256, 256) | constant | relu | sgd | 0.93864 | 0.796 |
| M18 | (256, 256) | constant | logistic | adam | 0.5 | 0.50056 |
| M19 | (256, 256) | constant | logistic | sgd | 0.86764 | 0.85648 |
| M20 | (256, 256) | adaptive | relu | adam | 0.96916 | 0.7984 |
| M21 | (256, 256) | adaptive | relu | sgd | 0.93864 | 0.796 |
| M22 | (256, 256) | adaptive | logistic | adam | 0.5 | 0.50056 |
| M23 | (256, 256) | adaptive | logistic | sgd | 0.86764 | 0.85648 |
| M24 | (64, 64, 64) | constant | relu | adam | 0.98128 | 0.796 |
| M25 | (64, 64, 64) | constant | relu | sgd | 0.99696 | 0.79464 |
| M26 | (64, 64, 64) | constant | logistic | adam | 0.99532 | 0.8056 |
| M27 | (64, 64, 64) | constant | logistic | sgd | 0.86816 | 0.85608 |
| M28 | (64, 64, 64) | adaptive | relu | adam | 0.98128 | 0.796 |
| M29 | (64, 64, 64) | adaptive | relu | sgd | 0.99696 | 0.79464 |
| M30 | (64, 64, 64) | adaptive | logistic | adam | 0.99532 | 0.8056 |
| M31 | (64, 64, 64) | adaptive | logistic | sgd | 0.86816 | 0.85608 |

**Table 1**: Training and Validation Accuracy Rates for Different MLP Classifier Configurations, with M15 is the configuration with the highest validation accuracy





**Link to Github repository**: <https://github.com/MiaNguyen912/IMDb-movie-reviews-classification>

(Note: each classifier is experimented with in a different GitHub branch)