Problem Definition

Many people look at movie reviews before going to the movie theaters because most people do not want to waste their time and money on products that do not appeal to the general audience. Even though a review is a subjective concept, it allows us to understand the general attitude of viewers towards the movie. In this paper, we are trying to predict a positive or negative sentiment of the movie review. We will use the Internet Movie Database (IMDb) dataset for Kaggle (N, 2019) to train our models and make correct classifications. The dataset contains 50000 movie reviews, with an equal split of reviews marked as "positive" and "negative." None of the entries contain nan values or empty reviews.

Solution Specification

Since one of the dataset columns has text input, we need to perform data preprocessing and transformation to convert all reviews into a format suitable for the Machine Learning models, which we will describe below. Thus, we begin by defining a method *clean* that performs the following operations:

- Remove HTML tags.
- Fix the contractions (e.g., replace "don't" with "do not").
- Remove punctuation and stop words because it helps decrease the size of the text corpus and is not relevant to further analysis.
- Convert the input to the lower case so that there is no ambiguity in the model understanding words that start with capital letters.
- Tokenize the remaining words and lemmatize them so that all words are reduced to their base form running vocabulary and morphological analyses.

We also convert the dependent variable type into an integer because our models will work with [0, 1] output where 0 is a negative review, and 1 is a positive review.

The last step before initializing the models would be to visualize the most common words in both review categories (see Appendix A). As we can notice from the diagrams, many words are actually neutral. Additionally, we see the word "positive" being frequently used in the negative reviews, which proves that separate words are not helpful to classify a review even if they have a strong connotation.

In this report, we have decided to test the performance of three different machine learning models on the problem introduced above. We have chosen logistic regression to be our base model because it is the simplest one out of the three. Then we run the XGBoost model — a "gradient boosting ensemble method that uses multiple simple decision tree models to achieve a single collated result" (Ming, 2020). We have created pipelines that perform additional review preprocessing before populating both Bag of Words models with training and testing data using *CountVectorizer* from the *scikit-learn* library. Finally, we implement Long short-term memory (LSTM), one of the recurrent neural networks. LSTMs are often used for making predictions based on the time-series data, which is our sequences of words in movie reviews in this case. We have defined the following network architecture for our LSTM implementation:

- Embedding layer (first hidden layer of the network; requires each word in the input to be represented as a unique integer).
- Dropout layer (layer that implements regularization and ignores some fraction of nodes during training).
- LSTM layer (learns long-term dependencies in the input).
- Dense layer (deeply connected layer that uses two different activation functions ("relu" and "sigmoid") in our model; converts the output to a probabilistic form [0, 1]).

Since our problem involves binary classification, we have decided to choose binary cross-entropy as a loss function for the last model. The goal is to minimize the loss function so that two probability distributions (expected and predicted) are as similar as possible.

Testing and Analysis

We suspect that there is a chance of overfitting for all initialized models. This means that our models will learn the training data with all its noise well and will fail to generalize the predictions on the previously unseen (testing) data. Thus, we implement a few approaches for every model, which should help minimize the chances of overfitting. For instance, logistic regression has a *penalty* hyperparameter that chooses a way of controlling the model and avoiding large weights. Another one is *C* — it selects the penalty strength (scikit, n.d.). To customize the model for our dataset and optimize the model hyperparameters, we use GridSearchCV, which defines a search space as a grid with selected hyperparameter values and takes a brute-force approach to evaluate model performance given every hyperparameter combination using cross-validation (*RepeatedStratifiedKFold* in our case). We have discovered that the best results on the hold-out data occur when the *penalty* is "L2" and *C* equals 0.1.

XGBoost model has more hyperparameters to optimize. We are not entirely sure which ones play the biggest role in improving model accuracy or which specific values the algorithm should check. Additionally, *GridSearchCV* takes very long to run since we deal with 384 hyperparameter combinations and repeat stratified K-Fold 3 times. Thus, we have decided to apply *RandomizedSearchCV* in this case. It is great for discovering "hyperparameter combinations that are difficult to guess intuitively" (Brownlee, 2020) because it randomly samples points from the defined search domain and checks the model performance given the sampled hyperparameter values.

In the LSTM model, we have varied the values of batch size (number of patterns a network can keep in memory at a time) and epochs ("number of times an entire dataset is shown to the network during training" (Brownlee, 2020). We have utilized *GridSearchCV* again and discovered that the best results on the hold out data occur when the *batch_size* is set to 100 with 10 *epochs*.

We have measured the performance of all models on the test sets and have noticed that our base model yields the most accurate results (89%), followed by the XGBoost decision tree model (85%) and LSTM neural network (82%). We have also looked at the cross-entropy metric in the last model, which was around 0.4, and hints that there might be room for improvement in network architecture.

We have visualized each model's classification results by plotting confusion matrices (see Appendix B). We can notice a slightly better rate of classification for positive reviews in terms of correctly identified samples. Otherwise, there is no visible skew towards one class (e.g., models classify negative reviews poorly and positive very well). The numbers in confusion matrices prove the conclusions about model accuracy rates presented above.

We believe there are a few explanations of why our base model is superior to the other two, which are more complex. Neural networks have higher flexibility in learning, which makes them more challenging to train and more prone to overfitting. Additionally, the loss function of LSTM can be very "bumpy" because many hyperparameters are added to it. Thus, no matter where the learning starts, stochastic gradient ascent might be stuck in the local optima. LSTM and XGBoost might also be more demanding in terms of data volumes and "richness" (van der Ploeg et al., 2014), which might also be the reason why a simple logistic regression is sufficient in our case. It is easier to interpret, train and it is less likely to overfit in low-dimensional datasets like the one we have introduced.

After analyzing the obtained outcomes, we think that the models still have some weaknesses that could be mitigated. Those are mostly related to hyperparameter tuning for all three models. A more extensive search space might positively impact the model accuracy because the algorithm picked some hyperparameters (e.g., *gamma* and *max_depth* in XGBoost) with the highest values in the specified domain. It means that if we expanded the domain, *RandomSearchCV*/*GridSearchCV* would potentially find hyperparameters that result in better accuracy for the given scenario. We also recommend changing LSTM architecture in terms of

the number of layers and their types. After varying the number of layers a few times, we have discovered that having many of them adds complexity which does not positively affect the LSTM prediction accuracy. However, we believe it is still a subject of further research and experiments.

References

Brownlee, J. (2020). *Hyperparameter optimization with random search and grid search*. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

Ming, J. (2020). *Sentiment classification using XGBoost*. Medium. Retrieved from https://ai.plainenglish.io/sentiment-classification-using-xgboost-7abdaf4771f9

N, L. (2019). *IMDB dataset of 50K movie reviews*. Kaggle. Retrieved from https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

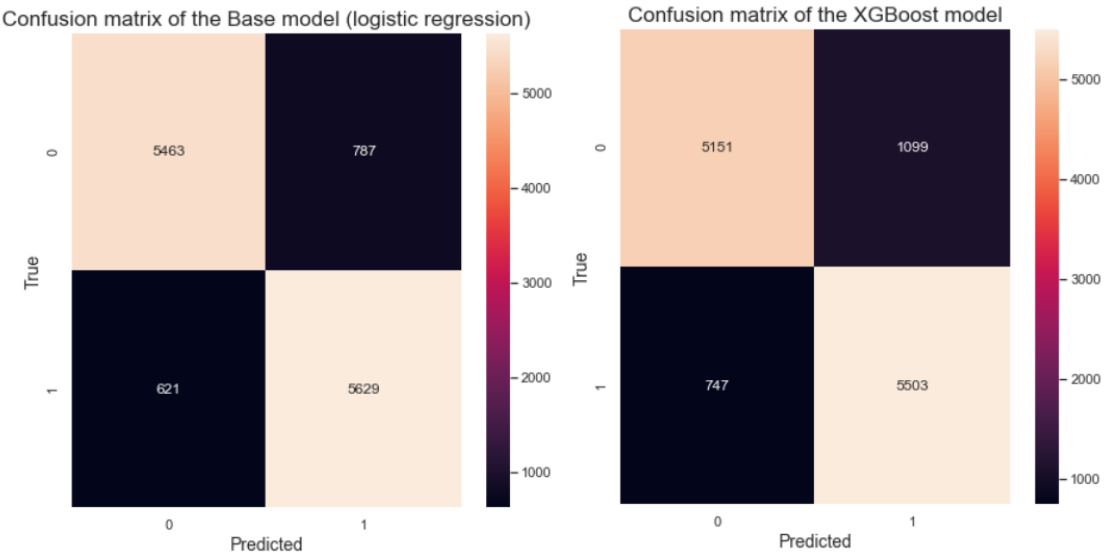
scikit. (n.d.). *Sklearn.linear_model.logisticregression*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

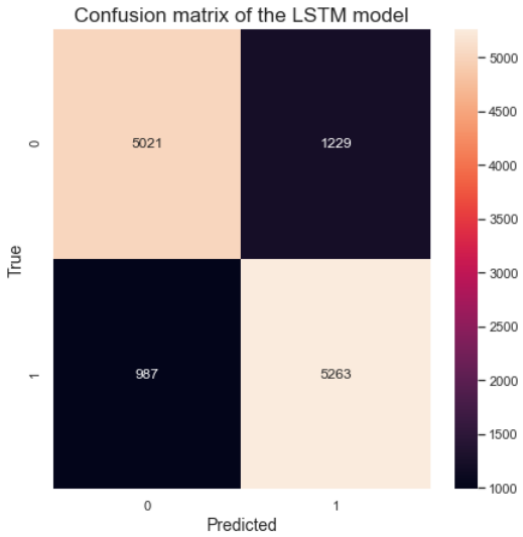
van der Ploeg, T., Austin, P. C., & Steyerberg, E. W. (2014). *Modern modelling techniques are data hungry: A simulation study for predicting dichotomous endpoints - BMC Medical Research methodology*. BioMed Central. Retrieved from https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/1471-2288-14-137

Appendices

Appendix A: WordClouds of words most used in positive and negative reviews.

Appendix B: Confusion matrices for each model




Confusion matrix of the LSTM model
5021
1229
987
5263
True
0
1
Predicted
0
1
5000
4500
4000
3500
3000
2500
2000
1500
1000