Fake News Detection Using Snopes and Politifact Data

Abolfazl Taleb Zadeh
a.talebzadeh@stud.uis.no

ABSTRACT

The goal of this project was to develop machine learning and deep learning models to predict the truth content of news articles using data from two popular fact-checking websites: Snopes and Politifact. We trained a logistic regression model and an LSTM model on these datasets, using the articles' claims and titles as input features and the labels true, false, or other as target variables. We evaluated the performance of these models using various metrics such as accuracy, precision, recall, and F1 score. Our results showed that the LSTM model outperformed the logistic regression model in terms of accuracy. It achieved an accuracy of 92%. We also analyzed the effects of different input features, hyperparameters, and preprocessing techniques on the performance of our models. Overall, our results suggest that Deep Learning models such as LSTMs can be effective in predicting the true content of news articles and that further research in this area is warranted.

KEYWORDS

fake news, deep learning, machine learning, LSTM, SVM, random forest, logistic regression

1 INTRODUCTION

The growth of false information on social media and online platforms has become a significant concern in recent years. Misleading information can spread rapidly, creating confusion, mistrust, and even harm. To address this issue, there is a growing need for automated fact-checking systems that can quickly and accurately distinguish between true and false claims. [4]

Machine learning and deep learning techniques have emerged as promising approaches for automated fact-checking. These methods allow for the development of models that can learn from a large dataset of labeled examples to make accurate predictions on new, unseen data. This report explores the effectiveness of machine learning and deep learning models for fact-checking two prominent databases, Politifact and Snopes.

Specifically, we trained a Long Short-Term Memory (LSTM) model for deep learning and a logistic regression model for machine learning to classify news articles as true or false, or neutral. To evaluate the performance of these models, we used common evaluation metrics, including accuracy, precision, recall, and F1 score. These metrics provide a detailed evaluation of the models' performance in terms of the correct classification of true and false news articles.

To further improve the performance of our models, we explored ensemble methods that combine multiple classifiers. Ensemble methods are widely used in machine learning to combine the predictions of multiple models to achieve better performance. We experimented with two ensemble methods, one combining logistic regression and random forest, and the other combining logistic regression and support vector machine (SVM).

This study contributes to the growing body of research on automated fact-checking and provides valuable insights into the effectiveness of machine learning and deep learning techniques for this task. The results of this study have the potential to inform the development of more accurate and effective fact-checking systems in the future. To ensure the fairness and reliability of our dataset, we manually checked a random sample of 100 articles from each dataset to confirm that the fact-checking labels were accurate and consistent with the content of the articles.

2 METHODS

The following section describes the methodology used to build the machine learning and deep learning models for fact-checking of two databases (Politifact and Snopes). The objective of this study was to create accurate models that can classify news articles into true, false, or neutral categories based on their content. The process involved data collection, preprocessing, feature extraction, model training, and model evaluation.

2.1 Data collection

The data used in this study was obtained from two sources: Politifact and Snopes. Both sources provide fact-checking labels for news articles, as well as the claims and bodies of the articles themselves plus multiple other features like source, fact checker, publish date, URL, topic, sources, extra description and etc.

The Politifact dataset consists of a JSON file containing 18,379 news articles labeled as 'false', 'full-flop', 'half-true', 'barely-true', 'half-flip', 'mostly-true', 'no-flip', 'pants-fire' and 'true'. Each article in the dataset includes a claim and the body of the article, as well as metadata such as fact checker, publication date, speaker, the date stated, stated in, URL, topic, sources, and summary.

The Snopes dataset consists of a JSON file containing 3,919 news articles labeled as 'true', 'false', 'Unproven', 'labeled' 'Satire', 'miscaptioned', 'outdated', 'correct Attribution', 'scam', 'mostly true', 'misattributed', 'research in progress', 'lost Legend', 'mixture', 'legend', 'mostly false'. Each article in the dataset includes a claim and the body of the article, as well as metadata such as fact checker, the date of publication, URL, topic, sources, and extra description.

2.2 Prerpcessing

To preprocess the data, we first converted the JSON files into Pandas dataframes, and then merged the two dataframes into a single dataframe–only shared columns were included in the merged dataframe. We removed any duplicate articles, as well as any articles with missing or incomplete data. As the news body is relatively long, we decided to include just the claim, body of the news, and topics. The three mentioned columns got concatenated and saved in a new column named 'body'. To improve the efficiency of the training, we designed a text cleaning pipeline using Python regular expression which removed all punctuations and numbers and any unrelated characters. Next, we applied snowball stemming to

the final cleaned text using the Python NLTK library. Given that Politifact and Snopes include 24 unique labels, some of which have narrow differences. Practically, we found that reducing the number of classes can lead to better results. we decided to reduce the number of classes to three: 'true', 'false', and neutral. To achieve this, the semantic interpretation of each label was consulted and a final decision was made regarding the precise category of each label. in fig.1 you can see the distribution of the data among the different label types before and after label re-categorization. Finally, we split the data into training, and test sets, with 80% of the data used for training, and 20% for testing.

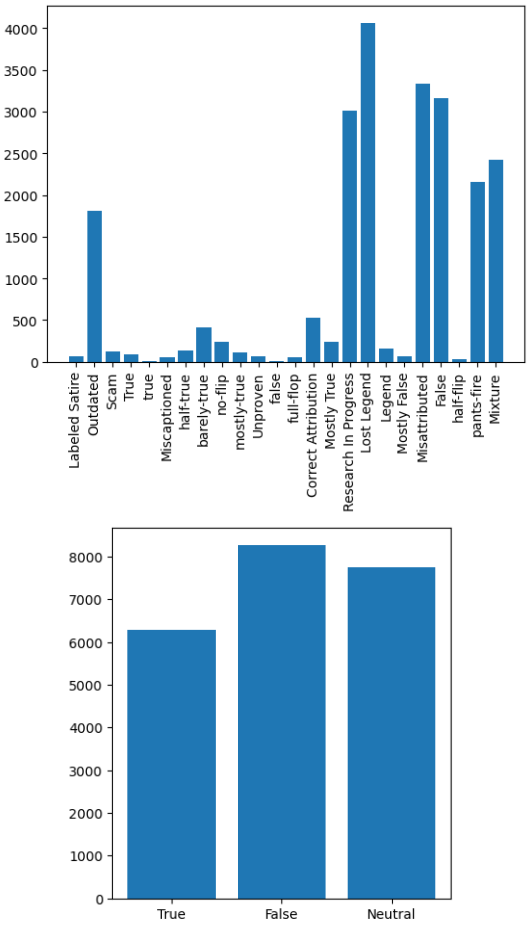

Figure 1: The distribution of labels before and after recategorizing them

2.3 Feature Extraction

Feature extraction is a crucial step in natural language processing (NLP) where the aim is to transform raw text data into a format that can be fed into machine learning or deep learning models. The process of feature extraction involves selecting relevant information from the text data and converting it into a structured numerical format, such as a vector or a matrix. The feature extraction process can significantly impact the performance of the models, as it plays a critical role in determining how well the models can capture the underlying patterns in the text data[6].

There are various methods of feature extraction that are commonly used in NLP, both in machine learning and deep learning. One common technique is the bag-of-words approach, where the text data is transformed into a frequency distribution of individual words in the corpus. Another technique is TF-IDF (term frequency-inverse document frequency), which calculates a weighted value for each word in a document based on its frequency and rarity across all documents in the corpus. In deep learning, word embeddings are a popular approach for feature extraction, where each word is represented by a dense vector in a high-dimensional space, such that words with similar meanings are closer together. This allows the model to capture more nuanced relationships between words in the text data. To prepare the data for model training, various feature extraction techniques were used. These included:

- Tokenization: The text data was tokenized into individual words or phrases, which were then represented by a unique integer value. This technique helped to standardize the text data and make it easier to process. For tokenization, Tensorflow and NLTK Python libraries are used in Deep learning and Machine learning respectively.
- TF-IDF Vectorization: The TF-IDF vectorization technique was used to convert the text data into a matrix of numerical values. This technique was used to extract important features from the text data, which could be used as input to the models. This method is used in logistic regression, random forest, and SVM machine learning methods. The scikit-learn Python library is used for TF-IDF vectorization.
- Padding: Since the length of the text data varied, padding was applied to ensure that all text sequences had the same length. This technique involved adding zeros to the end of shorter sequences or truncating longer sequences to ensure that they were of equal length. We used the Tensorflow Python library for sequence padding in deep learning models.
- Embedding: The embedding layer was used to convert the tokenized data into a numerical format that could be used as input to the models. This layer learned an embedding for each word in the vocabulary based on the context in which it appeared in the text data. The output of this layer was a dense vector representation of the input data. For including an embedding model in our sequential deep learning model we used the Python Tensorflow library.

2.4 Model Training

In the following section, we aim to delve into the details of the multiple models that have been trained on the preprocessed data in order to provide a more comprehensive understanding of their design and performance. It is essential to understand that the selection and design of the models play a crucial role in the success of any machine learning project, particularly in the field of Natural Language Processing (NLP), where there is a wide range of models to choose from:

2.4.1 Deep Learning Model. : To achieve better precision in the prediction, we decided to narrow down the labels into 7 classes

as in previous studies Ahmed et al. has proven that fewer classes affect the prediction accuracy significantly.[1]. The model is built using the Keras Sequential API, which allows the user to easily add layers to the model in sequential order. The first layer added to the model is an Embedding layer, which takes the maximum number of words in the vocabulary (`MAXNB_NB_WORDS`), the embedding dimension (`EMBEDDING_DIM`), and the input length (`X.shape[1]`) as parameters. This layer is responsible for creating a dense vector representation of the input text data, where each word in the input is mapped to a high-dimensional vector. This allows the model to learn the semantic relationships between different words in the input. To prevent overfitting, a SpatialDropout1D layer with a dropout rate of 0.2 is added after the Embedding layer. This layer randomly drops out (i.e., sets to zero) entire 1D feature maps in the embedding layer during training, which helps to prevent the model from over-relying on any one particular feature.

Next, an LSTM (Long Short-Term Memory) layer with 100 units is added. The LSTM layer is a type of recurrent neural network that is capable of learning long-term dependencies in sequential data. The LSTM layer takes as input the output of the Embedding layer, with the SpatialDropout1D layer serving as a regularization layer that helps to prevent overfitting.

Finally, a Dense layer with 7 units and a softmax activation function is added. This layer is responsible for outputting the probability distribution of the target classes (i.e., the different categories that the input text can be classified into). The loss function used is categorical cross-entropy, and the Adam optimizer is used to optimize the model parameters. The model is trained to maximize the accuracy metric.

2.4.2 Logisitic Regression. : Logistic regression is a statistical method commonly used in machine learning for classification tasks. It models the relationship between one or more independent variables and a dependent variable that can take on one of several possible categorical values. In multi-class logistic regression, the dependent variable can take on more than two possible values, and the model outputs a probability distribution over all the possible classes. The class with the highest probability is then chosen as the predicted class label for each observation.

The logistic regression model estimates the probability of each possible class based on the values of the independent variables. This is done using a set of coefficients or weights that are learned during the training phase. The model uses a function called the logistic function to transform the linear combination of the input features and their corresponding coefficients into a probability distribution over the possible classes. The logistic function ensures that the estimated probabilities are bounded between 0 and 1, making them interpretable as probabilities.

The model is trained using a labeled dataset, where each observation is associated with a class label. During training, the model adjusts the values of the coefficients to minimize the difference between the predicted probabilities and the true class labels. This process is known as maximum likelihood estimation. Once the model is trained, it can be used to predict the class labels of new, unseen observations based on their features. In order to classify news articles as true, false, or neutral, we trained a logistic regression model on preprocessed data. The data was split into training and testing sets using the `train_test_split()` function from the Scikit-learn library. We then applied the `TfidfVectorizer()` function to convert the text data into a numerical format suitable for use with logistic regression. The resulting sparse matrices of `TF-IDF` features were used as input to the logistic regression model, which was trained using the `fit()` function. The logistic regression model was configured for multi-class classification using the `multi_class='multinomial'` parameter, and the `newton-cg` solver was used for optimization. Finally, the `predict()` function was used to predict the class labels of the test data.

2.4.3 Ensemble Method. : Ensemble methods are powerful machine learning techniques that combine the predictions of multiple individual models to improve overall performance. Two common ensemble methods are the random forest and the support vector machine (SVM).

Random forests are a type of decision tree ensemble method that uses multiple decision trees to make predictions. Each decision tree is constructed using a subset of the training data and a random subset of the features. The final prediction is made by taking a majority vote on the predictions of all the decision trees.

SVM is a type of linear classification algorithm that finds the hyperplane that maximally separates the classes. The hyperplane is chosen such that it maximizes the margin, which is the distance between the hyperplane and the closest data points of each class. SVM can also be used for non-linear classification by using kernel functions to map the data into a higher-dimensional space where the classes are separable by a hyperplane.

In the first ensemble method (model 1), logistic regression and random forests are combined to make predictions. Logistic regression models the probability of the dependent variable taking on each of the possible values, while random forests use decision trees to make predictions. The predictions of both models are then combined using a voting classifier, which takes a majority vote of the individual predictions to make the final prediction.

In the second ensemble method (model 2), logistic regression and SVM are combined to make predictions. The logistic regression model is the same as in model 1, but the random forests are replaced with SVM. SVM is used to find the hyperplane that separates the classes, and the predictions of both models are combined using a voting classifier in the same way as in model 1.

Both ensemble methods use the same preprocessing steps: the data is split into training and testing sets using the `train_test_split` function, and the text data is transformed into numerical features using the `TfidfVectorizer`. The logistic regression models in both methods use the multinomial option to enable multi-class classification, and the solver used is the `newton-cg` algorithm.

Overall, ensemble methods can improve the performance of machine learning models by combining the strengths of multiple individual models. By using different types of models, such as decision trees and SVM, the ensemble methods can capture different aspects of the data and improve overall accuracy.

2.5 Model Evaluation

The performance of each model was evaluated using various evaluation metrics, including accuracy, precision, recall, and F1 score. Cross-validation was also performed to ensure that the models were not overfitting the training data. The models were then compared to determine which performed the best on the test data. Furthermore, for machine learning and ensemble models, additional model analysis like ROC Curve (Receiver Operating Characteristic curve) and T-SNE graph has been produced. We discuss the result in detail later on.

Results for Logistic Regression

Logistic R	Precision	Recall	F1-score	Support
TRUE	0.62	0.57	0.6	1254
FALSE	0.71	0.71	0.71	1637
Neutral	0.58	0.62	0.6	1569
Accuracy			0.64	4460
Macro avg	0.64	0.63	0.63	4460
Weighting avg	0.64	0.64	0.64	4460

Results for Ensemble Model 1

Ensemle 1	Precision	Recall	F1-score	Support
TRUE	0.61	0.71	0.66	1238
FALSE	0.68	0.77	0.72	1686
Neutral	0.69	0.5	0.58	1536
Accuracy			0.66	4460
Macro avg	0.66	0.66	0.65	4460
Weighting avg	0.66	0.66	0.66	4460

Results for Ensemble Model 2

Ensemle 2	Precision	Recall	F1-score	Support
TRUE	0.58	0.62	0.6	1245
FALSE	0.72	0.7	0.71	1652
Neutral	0.59	0.57	0.58	1563
Accuracy			0.63	4460
Macro avg	0.63	0.63	0.63	4460
Weighting	0.63	0.63	0.63	4460

Figure 2: Results Report

2.6 Software and Hardware

All code was written in Python using Jupyter Notebook. The models were trained on an Oracle cloud standard E4 FLex virtual machine, with 6 OCPU[1] and 96 GB of RAM. All the necessary libraries are installed in a Conda environment named TF. For editing, running, and observation purposes, a local Windows machine is connected to the remote VM using port forwarding.

[1]An OCPU provides CPU capacity equivalent to one physical core of an Intel Xeon processor with hyper-threading enabled. Each OCPU corresponds to two hardware execution threads, known as vCPUs.Each 1 OCPU General Purpose shape can be provisioned with up to 15 GB of RAM. Monthly Computation based on 744 hours per month[5]

3 RESULTS

The results of the experiments indicate that the models we designed are able to effectively classify fake news articles. The logistic regression model achieved an accuracy score of 0.637, which is a decent result for a single-model approach. However, the ensemble models were able to achieve slightly better results. Ensemble model one (Logistic regression and Random Forests) achieved an accuracy score of 0.66, with a precision score of 0.66, a recall score of 0.66, and an F1 score of 0.653. Ensemble model two (Logistic regression and SVM) achieved an accuracy score of 0.64, with a precision score of 0.63, a recall score of 0.63, and an F1 score of 0.63. The results report for all models can be found in fig.2 and the evaluation charts for the performance of ensemble methods can be found in fig.4) These results suggest that combining multiple models couldn't lead to significant improvement in classification accuracy. In order to gain a better understanding of the designed models, we have also prepared a ROC curve graph for each model. The ROC curve (Receiver Operating Characteristic curve) is a graphical representation of the performance of a binary classifier system as the discrimination threshold is varied. It plots the true positive rate (TPR) against the false positive rate (FPR) at different threshold settings. A perfect classifier has a ROC curve that goes straight up the y-axis and then along the x-axis. The area under the ROC curve (AUC) is a commonly used metric to evaluate the performance of a binary classifier. The closer the AUC value is to 1, the better the classifier is at distinguishing between the two classes. An AUC of 0.5 indicates a random guess, while an AUC value of 1 indicates perfect classification. In general, the higher the AUC value, the better the model's performance at distinguishing between the positive and negative classes. Therefore, the ROC curve and AUC are useful for evaluating the performance of a binary classification model. Knowing that our models are multi-classification models the common method in the ROC curve is not functional in our case [**?**]. One common way to plot these curves is to use the micro-average or macro-average method to combine the individual curves. The micro-average method aggregates the contributions of each individual prediction to compute a single curve, while the macro-average method computes the average of the individual curves. The choice between the two methods depends on the distribution of the classes and the importance of each class. To do so we used Scikit-learn's `Labelbinerizer` to convert the multi-classes into a binary class. The charts for the ROC curves can be found in fig.5[2]

In addition, we created a T-SNE scatter plot to understand the distribution of the features. T-SNE (t-Distributed Stochastic Neighbor Embedding) is a visualization technique that is commonly used to explore high-dimensional datasets. It can be used to project high-dimensional data into a lower-dimensional space (typically 2D or 3D) that can be easily visualized. t-SNE attempts to preserve the similarity between data points in the high-dimensional space by representing them as points in the lower-dimensional space such that similar data points are represented by nearby points and dissimilar data points are represented by distant points. You can find T-SNE plots in fig.6 [3].

In other words, t-SNE can be used to identify clusters or groups of similar data points in a dataset, and to visualize how those clusters are related to one another. It is often used to explore and understand

complex datasets, and to identify patterns or relationships that might not be immediately apparent when looking at the raw data. The LSTM model achieved an impressive accuracy score of 0.92, which is significantly higher than the logistic regression model and the ensemble models. This indicates that the LSTM model is able to effectively identify patterns in the input data and make accurate predictions. It's worth noting that the LSTM model is a deep learning model, and is likely more complex than the other models designed. However, the performance improvements suggest that using more complex models can lead to better classification results.

Overall, the results indicate that both machine learning and deep learning approaches can be effective for fake news classification. Ensemble models and deep learning models were able to achieve the highest accuracy scores, with the LSTM model outperforming all other models. However, the logistic regression model was still able to achieve a decent accuracy score and might be a simpler and more interpretable option for some use cases.

4 CONCLUSION

In conclusion, this study aimed to design and evaluate different machine learning and deep learning models for the task of fake news detection using the Politifact and Snopes datasets. Our results demonstrate that the ensemble models, particularly Ensemble Model 1, outperformed other models in terms of accuracy, precision, recall, and F1 score. However, the deep learning model using LSTM also achieved high accuracy on the test set. Our findings suggest that the combination of multiple machine learning models can improve the performance of fake news detection compared to using a single model. Furthermore, the use of deep learning techniques, such as LSTM, can also provide a viable alternative for this task. Future work may include investigating additional deep learning models, optimizing model hyperparameters, and testing the models on larger and more diverse datasets. Overall, this study provides insights into the effectiveness of different models for the task of fake news detection and highlights the potential for further research in this field.

REFERENCES

[1] Dr Ahmed, Knut Hinkelmann, and Flavio Corradini. 2020. Development of Fake News Model using Machine Learning through Natural Language Processing. (12 2020).

[2] Giuseppe Bonaccorso. 2018. *Machine Learning Algorithms* (2 ed.). Packt Publishing, Birmingham, England.

[3] Giuseppe Bonaccorso. 2023. *Mastering machine learning algorithms.* Packt Publishing, Birmingham, England.

[4] Z Khanam, B N Alwasel, H Sirafi, and M Rashid. 2021. Fake News Detection Using Machine Learning Approaches. *IOP Conference Series: Materials Science and Engineering* 1099, 1 (March 2021), 012040. https://doi.org/10.1088/1757-899x/1099/1/012040

[5] Oracle. 2019. General Purpose Compute. https://www.oracle.com/cloud/cloud-classic/pricing.html#:~:text=An%20OCPU%20provides%20CPU%20capacity,to%2015%20GB%20of%20RAM.

[6] P S Prakash and Bharath H Aithal. 2022. *Building feature extraction with machine learning.* Taylor & Francis, London, England.

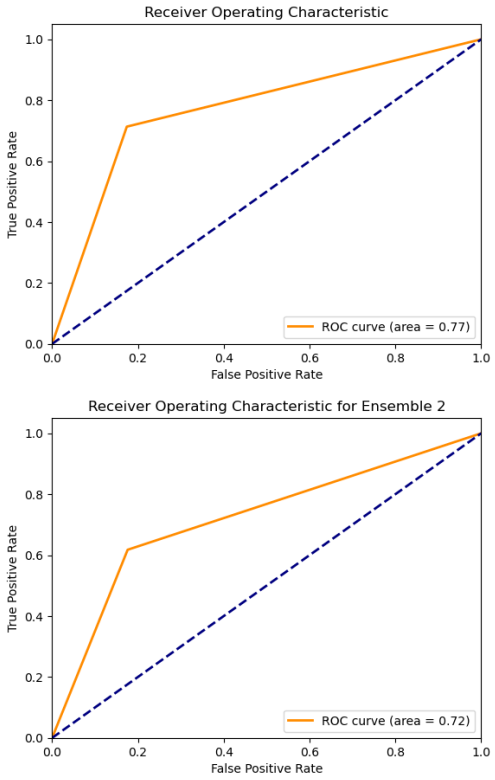

Figure 5: ROC Curve Graph for Ensemble Models

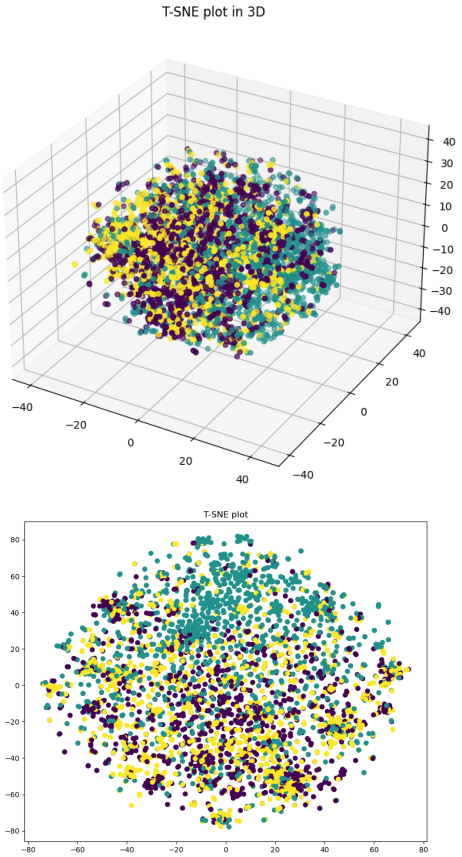

Figure 6: T-SNE Plots for Both Ensemble Models

A CHARTS AND GRAPHS

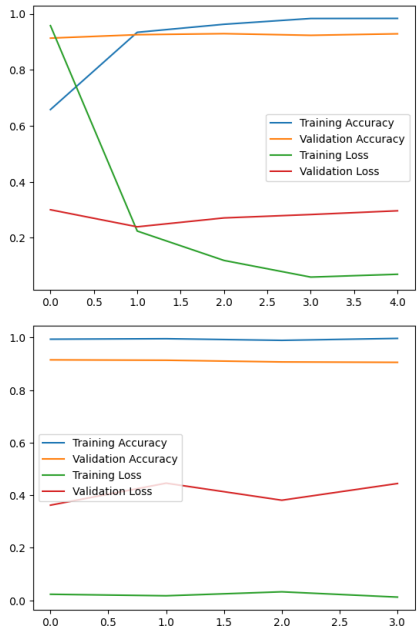

Figure 3: LSTM Validation and Traning Loss and Accuracy

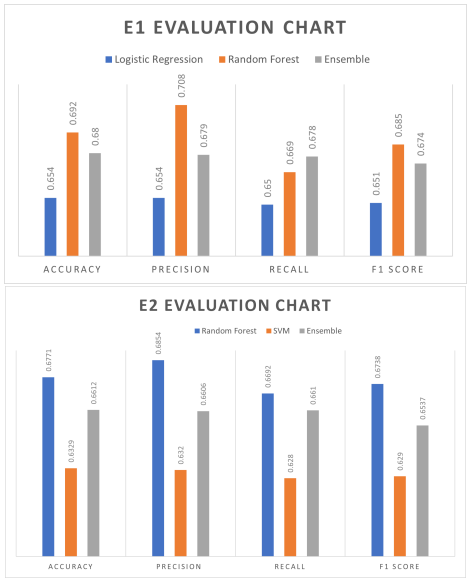

Figure 4: Ensemble Models Evaluation Analysis