Mini-Project 1

Hardik Jindal
Roll No. 220420

Naman Gupta
Roll No. 220686

Gupil
Roll No. 220416

Trijal Srivastava
Roll No. 221144

Harshit Srivastava
Roll No. 220444

Introduction

This project aims to address a binary classification problem using three datasets, each offering distinct feature representations derived from the same raw data. These datasets provide different views of the problem, which include:

- Emoticon-based features
- Deep learning-based features obtained via neural network embeddings
- Sequences of digits

Each dataset captures unique aspects of the underlying raw data, resulting in feature spaces that may offer complementary insights for classification. The goal was to develop and evaluate robust binary classification models for each dataset, optimising predictive accuracy and computational efficiency. Furthermore, leveraging the diversity of feature representations, we explored whether ensembling different feature sets can improve overall classification performance.

1 Task 1

Overview

The first task involved training binary classification models on three distinct datasets, each representing the same underlying classification problem but with unique feature representations. Each dataset was processed and analyzed independently to identify the best-performing model based on two primary criteria: (i) high classification accuracy and (ii) low training data requirements.

1.1 Emoticon Dataset

1.1.1 Dataset Overview

The dataset consists of input sequences composed of 13 emojis and a corresponding binary label as the output. Across the dataset, **214 unique** emojis are present in the input strings. Upon analyzing the frequency distribution of these emojis, several noteworthy patterns emerge:

- There are **seven** particularly frequent emojis in every input string.
- These seven frequent emojis constitute exactly **10 out of the 13** positions in each string across the training and validation datasets.
- A closer examination of the distribution of these seven emojis reveals no discernible pattern in their relation to the binary labels. The label distribution appears randomized, offering no clear separation between the two classes based solely on the presence or absence of these emojis.

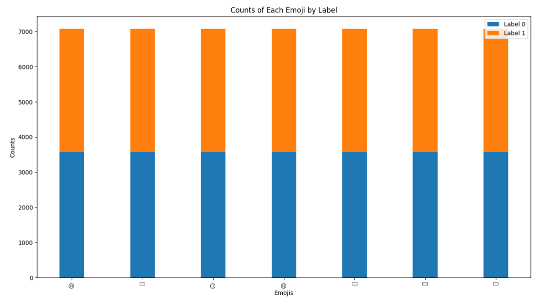


Figure 1: In order to plot this histogram, we eliminate all emojis that appear less frequently than 1000 in the Training Dataset.

These observations suggest that most of the input string composition is dominated by a subset of frequent emojis, potentially complicating the model's ability to learn meaningful patterns purely from these features.

1.1.2 Data Preprocessing

The preprocessing of the dataset involved several steps to prepare the emoji sequences for model training. The primary steps included filtering out the most frequent emojis, tokenizing the remaining sequences, and padding them to ensure uniform length. The detailed steps are as follows:

- **Filtering Frequent Emojis**: We first removed the most frequent emojis from each sequence to focus on the less common ones, as they might carry more distinguishing information for the classification task.
- **Tokenization**: Each emoji in the sequence was treated as a character and tokenized accordingly. We used a character-level tokenizer to convert the emoji sequences into integer representations.
- **Padding**: Since the emoji sequences varied in length, we padded them to a uniform length based on the longest sequence in the training data. This ensured consistency across the input data during model training.

The following code snippet illustrates the preprocessing steps implemented:

1.1.3 Experimentation

We experimented with a few prevalent binary classifier models to establish a direction to work with. (Note: Some models required one-hot encoding as it was a better choice than designating arbitrary numbers to the emojis, which may confuse the model into thinking the dataset values have a hierarchy order.) We tested with:

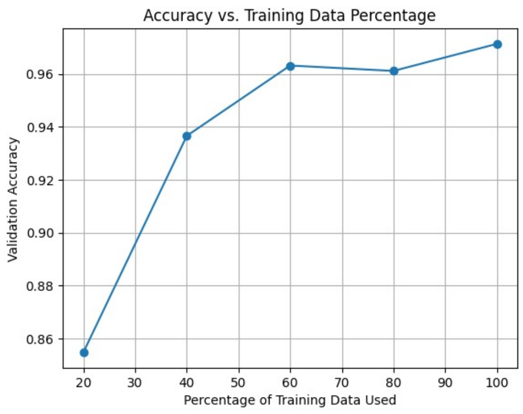
Model	**Accuracy (%)**
Logistic Regression	87.93
Decision Tree	75.87
Random Forest	86.09
XGBoost	75.66
MLP (Neural Network)	88.34*
LSTM	98.36

*: We observed that the best results are seen when a neural network (besides LSTM) is used on the data. Adding onto the fact that we are working with a sequence of characters that is generating a binary response, it led us to use **LSTM**, which has the use case as needed above.

1.1.4 Results

Percentage of data	Accuaracy
100	97.14
80	96.11
60	96.32
40	93.66
20	85.48

Table 1: Table to test captions and labels.



1.1.5 Challenges

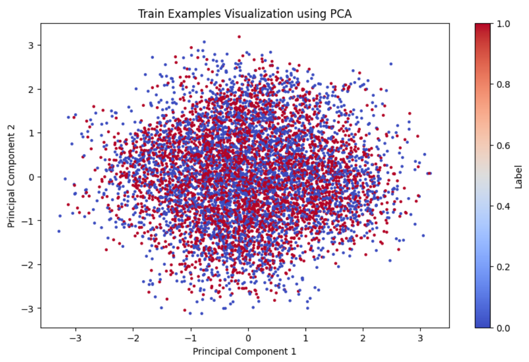


Figure 2: Visualization of the training examples using Principal Component Analysis (PCA), reduced to two dimensions shows that there is no clear separation between labels

The challenging aspect of this task was identifying a prevalent pattern in the dataset that would help the model work better. Also, there was no visual boundary or separation between labels that could be observed with Principal Component Analysis or similar. So, we moved to use neural networks. Additionally, tuning the neural network to work under the 10k parameters mark also came with its challenges.

1.2 Deep Features Dataset

1.2.1 Dataset Overview

The dataset used for this project consists of sequences of 13 emojis, where each sequence is associated with a binary label. Each emoji in the sequence is represented as a 768-dimensional embedding, resulting in a 3D tensor for the dataset with a shape of $(n\_samples, 13, 768)$. For the training set, there are 7,080 examples, and for the validation set, there are 489 examples.

Each example in the dataset contains the following:

- **Input**: A sequence of 13 emojis, where each emoji is encoded as a 768-dimensional vector.
- **Labels**: A binary label associated with each sequence, indicating the target class.

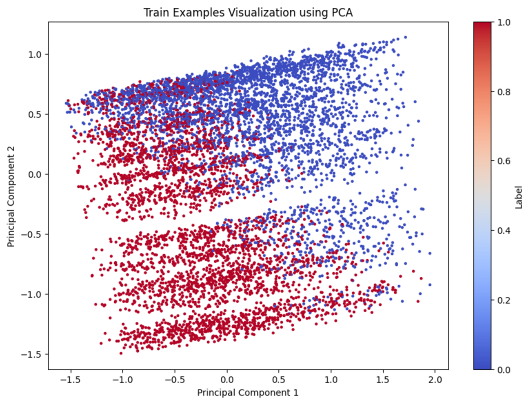


Figure 3: Visualization of the training examples using Principal Component Analysis (PCA), reduced to two dimensions. Each point represents a flattened input sequence, coloured according to its binary label. This visualization helps to explore the separability of the data in lower-dimensional space, highlighting potential patterns or clustering of examples based on their labels.

1.2.2 Preprocessing of Data

The preprocessing of the dataset involved converting the input data into a suitable format for machine learning models since only deep learning models like CNN and LSTM can handle 3D inputs; the rest all use 2D data as inputs, such as RandomForest. The key steps are outlined below:

- **Shape Inspection**: The dataset consists of 3D arrays with the shape $(n\_samples, sequence\_length, embedding\_size)$. The training data has a shape of $(7080, 13, 768)$, while the validation data has a shape of $(489, 13, 768)$. This means there are 7,080 and 489 examples, respectively, each represented as a sequence of 13 tokens, with each token described by a 768-dimensional embedding vector.
- **Flattening the Input**: Since models such as RandomForest expect 2D input, the 3D arrays (sequences of token embeddings) are flattened. The last two dimensions (sequence length and embedding size) are combined into a single dimension, resulting in input shapes of $(n\_samples, sequence\_length \times embedding\_size)$. For the training data, this results in a shape of $(7080, 9984)$, where each example is now represented by a single vector of 9984 features (i.e., 13×768). The validation and test sets are similarly flattened.
  The inspiration for this approach comes from the MNIST dataset NN-based approach for Multi-label Classification.

1.2.3 Experimentation

The experimentation began with a deep learning-based approach, given the 3D nature of the dataset. The input sequences, consisting of 13 emojis represented by 768-dimensional embeddings, naturally approached using sequence modelling. Therefore, an initial LSTM-based model was designed to capture temporal dependencies in the sequences.

CNN-Based Approach The model architecture involved a combination of convolutional and dense layers to extract features and perform binary classification:

Layer	Output Shape	Param #	Connected to
Conv1D	(None, 11, 128)	295,040	Conv1D Layer
GlobalMaxPooling1D	(None, 128)	0	Max pooling after Conv1D
Dense	(None, 64)	8,256	Dense Layer with ReLU activation
Dropout	(None, 64)	0	Dropout Layer with rate 0.5
Dense	(None, 1)	65	Dense Layer with Sigmoid activation

Table 2: CNN Model Architecture

Despite experimenting with various hyperparameters (including filter size, learning rate, and number of LSTM units), the model did not achieve the required 95% accuracy, as set by the professor. Even after multiple rounds of hyperparameter tuning, the model's performance plateaued, and further improvements were marginal.

Linear Classification with SVM Given the suboptimal results of the deep learning approach, the next step was to flatten the 3D dataset into a 2D form. After visualizing the data using PCA, we noticed potential separability between the classes, prompting the use of a Support Vector Classifier (SVC).

- **Results**: Although the linear model captured some of the structure, none of the configurations achieved a satisfactory accuracy level. The SVC consistently underperformed, even with various hyperparameters and kernel functions.

Tree-Based Approach: RandomForest Classifier The final approach was to employ a tree-based model, specifically the RandomForest classifier, given its robustness and ability to handle high-dimensional data.

- **Results**: This approach yielded the best performance, successfully achieving the target accuracy of over 95%. RandomForest's ensemble nature and ability to handle complex decision boundaries made it well-suited for the classification task.

In conclusion, while the deep learning and linear approaches were unsuccessful, the RandomForest model provided the desired accuracy, making it the final choice for the classification problem.

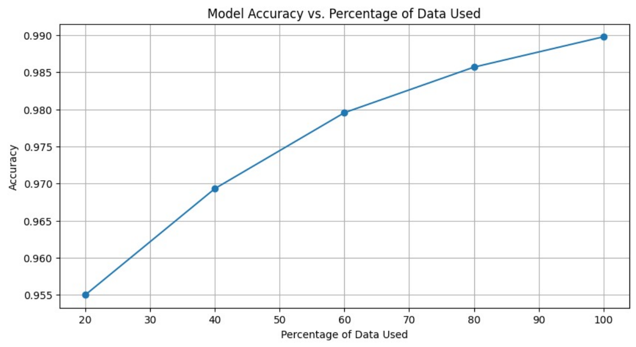
Table 3: Comparison of Models and Hyperparameters with Accuracy

Model	Hyperparameters	Accuracy (%)	Remarks
CNN	Filters: 128, Kernel Size: 3, Dense Units: 64, Dropout: 0.5	96.1	Reach target, param exceed
CNN	Filters: 4, Kernel Size: 3, Dense Units: 64, Dropout: 0.5	92	Target not reached
SVC (Linear)	Kernel: Linear	95.1	Not so good performance
SVC (Polynomial)	Kernel: Polynomial, Degree:5	98.36	Good performance
SVC (RBF)	Kernel: RBF	97.4	Good performance
RandomForest	n_estimators: 100	98.1	Good performance
RandomForest	n_estimators: 200	98.97	Best performance

1.2.4 Results

Percentage of data	Accuracy (%)
100	98.97
80	98.56
60	97.95
40	96.93
20	95.50

Table 4: Table comparing the best-fit model on Subsets of Data



1.2.5 Challenges

The biggest hurdle was handling the high dimensional data, which rendered any thought of visual pattern observation useless. Flattening the data to 2-D in the pre-processing was a unique insight that was reached after many initial hypotheses failed to reach the desired accuracy.

1.3 Text Sequence Dataset

1.3.1 Description of Data

The dataset contained 50 characters input strings of digits, giving a binary label as the output. The digits could be any from ’0123456789’. For the training set, we have 7080 data points and for the validation set, we have 489 data points.

1.3.2 Preprocessing of Data

Initially, our approach was to form embeddings of the whole input string and feed them into our Deep learning-based model. A clear observation from the emoticon dataset was that each emoji was represented as a sequence of 3,4 or 5 digits. However, it was difficult to form an algorithm that could correctly find these sequences of digits. So, we resorted to the below approach:

1. **Digit Encoding**: A Label Encoder is used to map each digit from ’0’ to ’9’ to a unique integer $e_i \in \{0, 1, \ldots, 9\}$.
2. **Sequence Transformation**: The input string S, composed of digits, is transformed into a sequence of encoded integers by replacing each digit s_i in S with its corresponding encoded value e_i.

A small check showed that ’000’ was common between all strings in the start. We removed these zeroes, and the accuracy increased since the input dimension was reduced and the complexity of the data decreased.

1.3.3 Experimentation

We experimented with many models, from basic regression models to deep learning architectures. Our experimentation varied from changing the number of trainable parameters to hyperparameter tuning. We obtained far better results with deep learning-based models. We observed that using models with more than 10K parameters improved our accuracy greatly, but since the restriction was imposed on the number of trainable parameters, we adhered to that.
Our main experimentation includes:

- **Logistic Regression**: We first applied logistic regression as the baseline model, which gave us an accuracy of a mere 50 % on the validation dataset, indicating that a linear model is struggling to differentiate between the two classes
- **Random Forest** Next, we applied the random forest classifier, which has some non-linearity, by averaging over multiple trees. This improved the accuracy to 57% hinting that an ensemble model would be a better direction to proceed in
- **XGBoost** Finally, we implemented XGBoost, a gradient-boosting decision tree algorithm optimized for high performance. XGBoost achieved the highest accuracy among the models tested, reaching 60%. However, the accuracy remained modest, highlighting the challenges in effectively classifying sequential data with these methods.
- **LSTM** Next, to capture the sequential information in the strings, we applied a bilayer LSTM architecture. We added an embedding layer before and a similar dense layer after the LSTM units with a dropout layer in between to prevent overfitting. This gave a remarkable boost in accuracy to a little over 80%.
- **CNN** Using a CNN for sequence classification helps detect local patterns in the digit sequences, which might correlate with the binary labels. CNNs are parameter-efficient, leveraging shared weights across the input. They also train faster than LSTMs due to parallel computation. We obtained an accuracy of around 84% with this approach.
- **LSTM+CNN (Hybrid Model)** To leverage the benefits of both LSTM and CNN, we thought of creating a hybrid model, comprising of a convolution layer then adding an LSTM layer on top. Combining CNNs with LSTM captures both local features (via CNN) and long-term dependencies (via LSTM). This hybrid approach can improve accuracy while maintaining a manageable parameter count. We got an accuracy of around 89 % on this model.

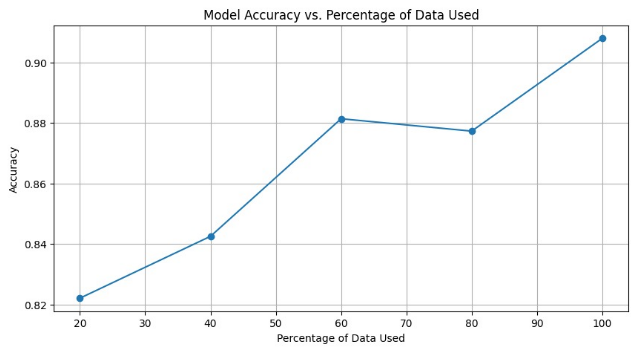
Layer (type)	**Output Shape**	**Param #**
embedding (Embedding)	(None, 47, 12)	120
conv1d (Conv1D)	(None, 45, 32)	1,184
max pooling1d (MaxPooling1D)	(None, 22, 32)	0
Dropout (Dropout)	(None, 22, 32)	0
lstm (LSTM)	(None, 24)	5,472
dense (Dense)	(None, 32)	800
dense (Dense)	(None, 2)	66

Table 5: LSTM+CNN Model Architecture

1.3.4 Results

Percentage of data	Accuracy(%)
100	90.79
80	87.73
60	88.14
40	84.25
20	82.21

Table 6: Table to test captions and labels.



1.3.5 Challenges

The dataset was undoubtedly the most difficult to work with. It was difficult to draw any definite pattern or relationship from observation. Consecutively, simple regression and classification models like logistic regression, random forests, and gradient descent models failed to capture any strong pattern among the data points, resulting in their low accuracy.

Even complex deep learning algorithms did not give us a very high accuracy percentage due to the above facts. Additionally, solving the task with the restriction imposed on the number of trainable tokens was another challenge, as accuracy was being compromised.

2 Task 2

Overview

The second task builds upon the insights gained from Task 1 by investigating whether combining the three datasets can lead to a more accurate and robust classification model. Although the datasets originate from the same raw data, the different feature representations may capture complementary information, providing an opportunity to enhance predictive performance through dataset integration.

2.1 Preprocessing of Data

2.1.1 Preprocessing Steps

The preprocessing of the dataset involved several key steps to transform the input emoticon strings and numerical strings into a format suitable for machine learning models.

Text Tokenization and Padding The input emoticon strings were tokenized at the character level using the Keras `Tokenizer`, treating each emoji as a separate character. The tokenization process was applied to both the training and validation sets.

After tokenization, the sequences were padded to ensure all input sequences had the same length, which is required by machine learning models. The maximum sequence length was determined based on the longest sequence in the training set, and all shorter sequences were padded with zeros.

Chunking the numerical strings Each input string was then split into smaller chunks for further analysis. A helper function `split_into_chunks` was used to split the input sequences into chunks of size 2.

Flattening the Dataset 2 Again, since the rest of our dataset was suitable for 2D models, we simply flattened the entire dataset. This results in a shape of $(7080, 9984)$, where each example is now represented by a single vector of 9984 features (i.e., 13×768). The validation and test sets are similarly flattened.

Combining Dataframes After the steps above, the newly created columns were concatenated with the labels of the original dataset. To avoid redundancy, the `input_emoticon` column was removed from the data frame.

Correlation-Based Feature Selection We performed a correlation analysis between each feature and the target label to reduce dimensionality and focus on the most important features. Only features with an absolute correlation greater than 0.00001 (determined by hyperparameter tuning) were retained. This threshold was selected to filter out irrelevant or weakly correlated features while preserving the most predictive ones.

The original number of features in the training set was **10,022**. After correlation filtering, **2341** features were retained. This process significantly reduced the dimensionality of the dataset, making the model more efficient while preserving predictive power.

This preprocessing pipeline ensured that the data was transformed and structured to be efficiently processed by the machine learning algorithms while preserving the structure and information of the original input strings.

2.2 Experimentation

2.2.1 Model Comparison and Evaluation

Validation Set Accuracy After training on the full training set using 5-fold cross validation, various models were evaluated on the unseen validation set, with the following results:

Classifier	**Validation Set Accuracy (%)**
Random Forest	98.57
Support Vector Machine (SVM)	56.65
Naive Bayes	94.89
K-Nearest Neighbors (KNN)	51.74
Logistic Regression	95.91

Table 7: Validation set accuracy of classifiers

Conclusion The validation set results show that the Random Forest classifier performed the best, achieving the highest validation accuracy of 98.57 %. Based on these results, Random Forest was chosen as the final model for this task.

Feature Importance Analysis The feature importances were extracted from the trained Random-Forest model using the `feature_importances_` attribute. These importances reflect the relative importance of each feature in making predictions. The top 20 most important features were identified and are visualized in Figure 4. The results show that *Dataset 2* contributes the most to the model's decision-making process.

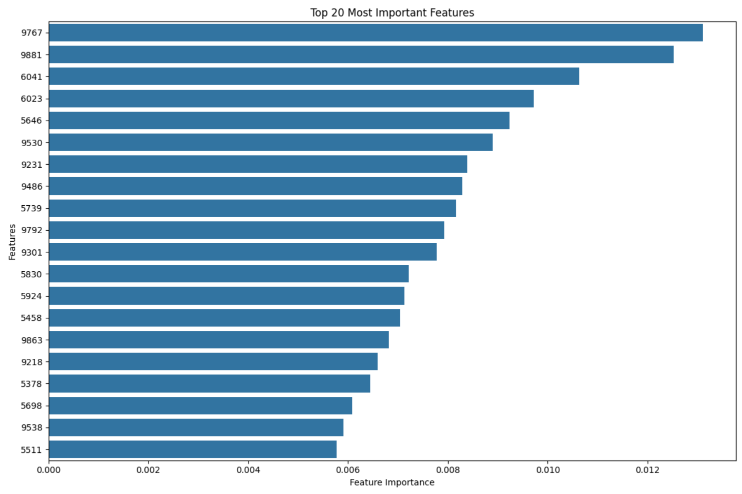
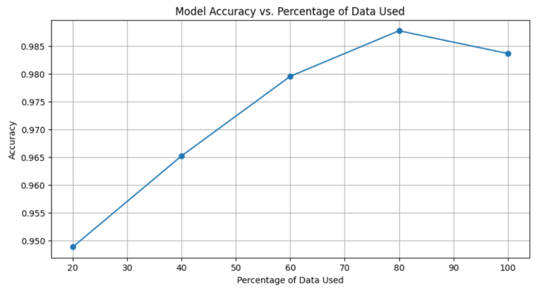


Figure 4: Top 20 Most Important Features as determined by the RandomForest Classifier. The x-axis represents feature importance, while the y-axis lists the features ranked by importance.

2.3 Results

Percentage of data	Accuracy
100	98.36
80	98.77
60	97.95
40	96.52
20	94.88

Table 8: Table to test captions and labels.



2.4 Conclusion based on the Analysis

During the experiments, we observed that although the dataset contained different representations of the input data (emoticon strings, numerical representations, and deep feature embeddings), the deep feature embeddings significantly outperformed the other two. This can be attributed to the ability of deep learning models to effectively capture high-level abstractions and latent patterns within the data, mapping it into a more meaningful feature space. In contrast, the raw emoticon strings and their corresponding numerical representations failed to encapsulate the underlying structure of the data as effectively, leading to inferior model performance.

2.5 Challenges

The main challenge was to ensure that the individual models combined correctly, i.e., handling any data incompatibility and removing redundancy before using the data. Additionally, it was challenging to find how dataset 2 contributed the most and would majorly determine the model.

3 Conclusion

3.1 Comparative Analysis

Conclusively, the following models were found to be best suitable:

Dataset	Model	Accuracy (%)
1.1	LSTM	97.14
1.2	Random Forest	98.97
1.3	LSTM+CNN	90.79
2	Random Forest	98.36

As seen here the model on the combined dataset performs at par with the best of the individual models, and much better than our approach for the text sequence dataset.

3.2 Patterns and insights

- The Random Forest models were faster and had better results on lower training data than the neural networks.
- The combined model was beneficial as it worked on a simpler model (Random Forest) and did not compromise accuracy and scalability.
- The pre-processing stage was crucial as even complex algorithms failed to perform if data was not processed accordingly.
- The datasets had some correlations that helped us to work with them. Once observed that the text sequence data was somehow 'encoded' to the emojis one, the next steps became more natural to work upon.
- Although all the datasets worked on sequential data, the best model did not always implicitly retain the sequential data (as in dataset 1.2), a direct consequence of feature importance.

References

[1] Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory*. Neural Computation, 9(8), 1735–1780.

[2] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11), 2278-2324.

[3] Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[4] Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). *Recurrent Convolutional Neural Networks for Text Classification*. Proceedings of the 29th AAAI Conference on Artificial Intelligence.

[5] Cox, D. R. (1958). *The regression analysis of binary sequences (with discussion)*. Journal of the Royal Statistical Society: Series B (Methodological), 20(2), 215-242.

[6] Breiman, L. (2001). *Random forests*. Machine Learning, 45(1), 5-32.

[7] Quinlan, J. R. (1986). *Induction of decision trees*. Machine Learning, 1(1), 81-106.

[8] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.