Classification ML Project

Aaditi Agrawal
220006

Vasudharaje Srivastava
221176

Manvi Kumar
220630

Suneet Maharana
221105

Ritesh Baviskar
220286

Aditya Jagdale
220470

1 Task-1

Here we approach Task-1 for all the three datasets with different models specific to the dataset problem and then tuning it with proper rationale accordingly.

1.1 Dataset-1

Introduction

This task focuses on binary classification of emoticon sequences, with each input containing 13 emoticons mapped to binary labels (0 or 1). The task involves converting emoticons to integers, padding sequences, and testing various models—Logistic Regression, Random Forests, XGBoost, RNNs, CNNs, and CNN-based hybrid models—to achieve high accuracy while using minimal training data.

Preprocessing

Each emoticon has been mapped to a unique integer for for model compatibility. There were 214 unique emoticons resulting in a mapping of size 214. One-hot encoding was not used because it would have increased the model complexity because of its sparse nature.

Experimentation and results

Logistic Regression, **Random Forest**, and **XGBoost** were implemented as baseline models to establish initial benchmarks.

Model	Accuracy (%)
Logistic Regression	59.1
Random Forest	63.8
XGBoost	65.0
SVM	58.49

Table 1: Validation Set Accuracy of Different Models

To explore the possibility of presence of local patterns in the emoticon string, sequential models were tested.We used RNNs for this task because they are designed to process sequential data, making them ideal for capturing temporal dependencies in emoticon sequences. Emoticons, like words in a sentence, have context that evolves across the sequence. RNNs allow the model to learn relationships and patterns across the sequence, making them well-suited for understanding how different emoticons interact within a string and influence the final classification. Their capability to retain information from previous steps helps in understanding complex emoticon patterns more effectively.

Why CNNs and RNNs?

CNNs were explored after RNNs to leverage their unique advantages in handling sequential data:

1. **Local Pattern Recognition**: CNNs excel at capturing local patterns within fixed-length sequences using convolutional filters. In the case of emoticon sequences, groups of related emoticons may have important local dependencies, which CNNs can detect effectively.

2. **Parallel Processing and Speed**: Unlike RNNs, which process data sequentially, CNNs process data in parallel, making them faster and more computationally efficient. This parallelism also reduces training time, which is particularly beneficial when dealing with large datasets or short, fixed-length sequences like the 13-emoticon inputs here.
3. **Effective Handling of Short Sequences**: Given the short and uniform length of the sequences in this dataset, CNNs can extract relevant features without the vanishing or exploding gradient problems that RNNs may encounter. This makes CNNs well-suited for modeling dependencies in short sequences like emoticon strings.

Hybrid Models

Hybrid models were developed by combining CNN-extracted features with traditional classifiers:

- **Feature Extraction**: The output from the last convolutional layer of the CNN was used as input features for classifiers like Logistic Regression, Random Forest, and XGBoost. This approach allows the CNN to act as a feature extractor, leveraging its ability to capture complex spatial patterns in the sequences.
- **Classifier Integration**:
  - **Logistic Regression**: Applied to CNN-extracted features to maintain a simple, interpretable model.
  - **Random Forest and XGBoost**: Used to enhance non-linear decision boundaries with the powerful feature representations extracted by the CNN.

  This approach was inspired by prior work, which demonstrated that deep learning feature extraction combined with traditional models can improve performance in sequence classification tasks.

Model Accuracy Comparison

Model	Accuracy (%)
RNN + Logistic Regression	59.1
RNN + Random Forest	63.8
RNN + XGBoost	65.0
CNN	97.5
CNN + Random Forest	96.5
CNN + Logistic Regression	97.34
CNN + SVM	96.9

Table 2: Validation Set Accuracy of Different Models

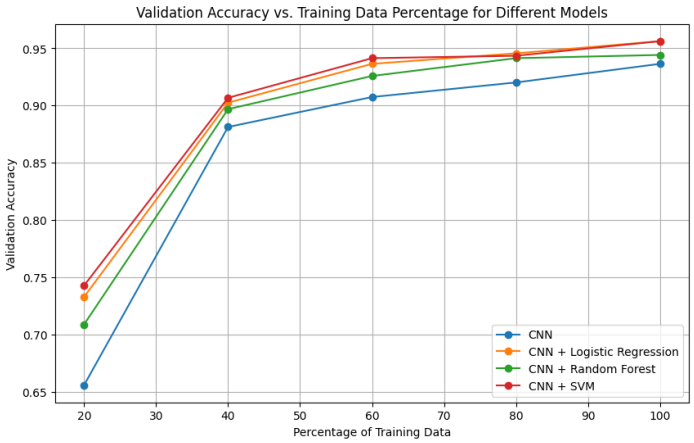

Figure 1: Validation accuracy vs Training data %

Training Data Efficiency

The validation accuracy improved significantly from 77.5% to 92.5% when increasing training data from 20% to 40%. However, the accuracy gain slowed and plateaued near 95% at 100% training data.

Conclusion

In conclusion, the CNN model delivered the highest validation accuracy (97.5%) due to its strong capability to extract local features from emoticon sequences. Convolutional layers effectively capture spatial relationships, while max-pooling preserves critical features, enhancing generalization. This end-to-end approach allows the CNN to better learn the inherent structure of sequences compared to feature extraction-based models. Consequently, CNNs are particularly well-suited for this task, demonstrating their effectiveness in processing emoticon-based sequential data

1.2 Dataset-2

Introduction

High-dimensional data poses significant challenges in machine learning, including increased computational complexity and the risk of overfitting. When dealing with such data, dimensionality reduction techniques become essential to simplify the model while retaining the most informative features. This report investigates the process of developing a convolutional neural network (CNN) model for binary classification on a dataset with high-dimensional sequential data. The dataset comprises over 7,000 samples, each with features of dimension 13×768. Due to a constraint of maintaining the number of trainable parameters below 10,000, Principal Component Analysis (PCA) was employed to reduce the feature dimensionality.

Preprocessing

Principal Component Analysis (PCA)

To address the high dimensionality and meet the parameter constraint, PCA was applied to reduce the number of features from 768 to 50. PCA is a statistical technique that transforms the original variables into a new set of uncorrelated variables called principal components, ordered by the amount of variance they capture from the data.

PCA is computationally efficient and scales well with large datasets, making it suitable for our dataset of over 7,000 samples. It is a linear, unsupervised method that does not require label information or extensive hyperparameter tuning, simplifying implementation and integration into our preprocessing pipeline. Compared to non-linear methods like t-SNE or UMAP—which are computationally intensive and better suited for data visualization rather than feature extraction for modeling—PCA offers a practical balance between efficiency and effectiveness. Additionally, PCA does not increase model complexity or introduce additional parameters, helping us meet the constraint of keeping the total trainable parameters under 10,000 in our CNN model. Also the number 50 was also selected after comparing various dimensions in that range while also satisfying the trainable parameters constraints on the CNN model.

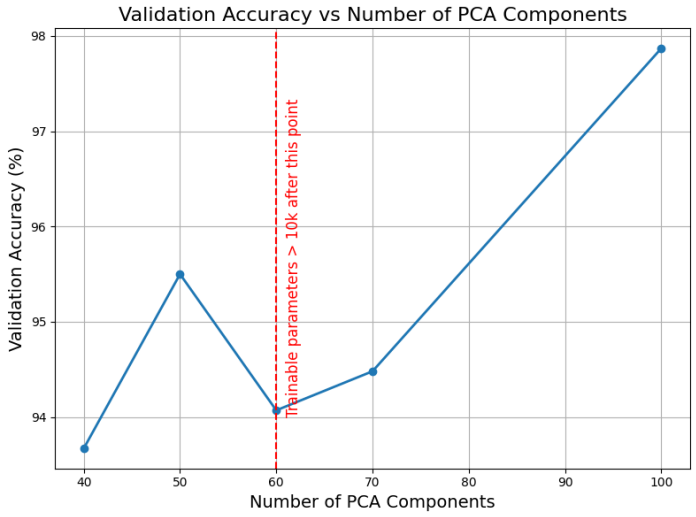

Figure 2: Validation Accuracy vs Number of PCA components

Implementation Steps

1. **Reshaping Data**: The original data has a shape of $(n_{\text{samples}}, 13, 768)$. For PCA, we reshape it into a 2D array:

$$\mathbf{X}_{\text{flattened}} \in \mathbb{R}^{(n_{\text{samples}} \times 13) \times 768}$$

2. **Applying PCA**: We fit the PCA model to $\mathbf{X}_{\text{flattened}}$ to reduce the feature dimension from 768 to 50:

$$\mathbf{X}_{\text{reduced\_flat}} = \text{PCA}_{50}(\mathbf{X}_{\text{flattened}})$$

3. **Reshaping Reduced Data**: The reduced data is reshaped back to the 3D form:

$$\mathbf{X}_{\text{reduced}} \in \mathbb{R}^{n_{\text{samples}} \times 13 \times 50}$$

4. **Train-Test Split**: The reduced data is split into training and validation sets using an 80-20 split.

By reducing the feature dimension to 50, we significantly decrease the computational complexity and the number of parameters in the model.

Rationale for Using Neural Networks

The primary reason for choosing CNN as the leading model for this task lies in its ability to maintain the spatial structure of input data. CNNs are particularly efficient for processing grid-like data and can effectively capture patterns in input matrices without flattening, which ensures that the inherent relationships in the data are preserved. Given the 13x768 dimensionality, the CNN's convolutional layers can efficiently detect local dependencies along both dimensions.

Flattening the input data, as required by traditional models like Random Forest (RF) or regression, would have resulted in the loss of this two-dimensional structure, leading to potential information degradation. Hence, RF and regression models were excluded from consideration. This is why we chose to use **only Neural Networks** as baseline models as well.

Benchmark Models

The dataset was also evaluated using three other neural network models:

GRU: The Gated Recurrent Unit was used to analyze temporal relationships in data, focusing on its efficiency in managing short-term dependencies. RNN + LSTM: This hybrid model was designed to capture both short-term and long-term dependencies, utilizing the recurrent capabilities of RNNs and the gated memory capabilities of LSTMs. Bi-LSTM: A bidirectional variant of the LSTM that processes the input in both forward and backward directions, making it suitable for capturing context in sequences.

Model	Accuracy (%)
RNN + LSTM	94.07
Bi-LSTM	94.48
GRU	95.50
CNN	96.52

Table 3: Validation Set Accuracy of Different Models

Model Training and parameter constraints

All models, including the CNN, were constrained to use fewer than 10,000 parameters. This was essential to maintain computational efficiency and ensure faster training times, given the large input dimensions and significant number of inputs.

Results and Performance

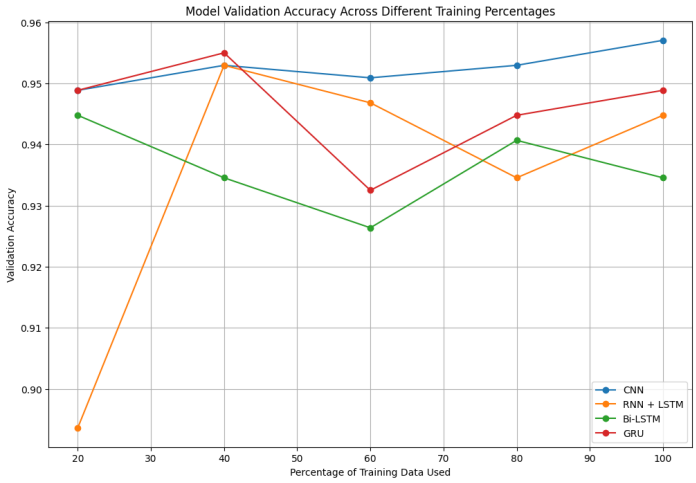

Figure 3: Accuracy vs % of training data for all the models

We can see that the CNN model has a very high training efficiency from 20% all the way up to 100% training data which makes it a robust model. The CNN model had a bit better performance over the GRU, RNN + LSTM, and Bi-LSTM models in terms of accuracy at 100% training data. The results suggest that CNN's ability to retain spatial information and efficiently extract hierarchical features makes it better suited for this dataset structure. Additionally, the lower parameter count enabled faster convergence and reduced computational load during training.

Conclusion

CNN emerged as the most suitable model for this dataset due to its consistent accuracy over varied training data and effectiveness in retaining input structure and its computational efficiency. Despite the benchmarks provided by GRU, RNN + LSTM, and Bi-LSTM models, CNN's design allowed it to excel in capturing relevant patterns within the data.

Dataset-3

Introduction

The dataset consists of a string of length of 50 and binary label associated with each sample. Moreover, the string only consists of only character from 0-9. Basic statistics of the dataset are as follows:
Number of samples with label = 1: 3504
Number of samples with label = 0: 3576
and the total number of samples is 7080. The dataset is not polarised which makes the further analysis simpler. There could have been two broad ways in which the label depends on the string.

1. The label just depends on the presence or absence of certain character. In this case the order of occurrence of characters in the string does not matter. In this case a simple model that can identify and learn non-sequential relations would work well.
2. The label depends on the order of occurrence of characters matters. In this case a model which could capture sequential or contexual relations need to be used.

We start by using a simple sequential model on the dataset. Then we shuffle the characters in the string and evaluate the accuracies in both cases to see if changing the sequential order makes a difference.
We used a LSTM(least short term memory) for this task which is a basic sequential model. The model showed a difference of around 15% when the characters in the string were shuffled. This shows that there are contexual relations that when taken into consideration would give better prediction results.

Preprocessing

Two methods of tokenization were tried out : One hot encoding and using Tokenizer from Keras. One hot encoding increases the number of features by a factor of 10 and thus has been used only for basic models in the experimentation phase.
The Tokenizer from Keras is initialized with the parameter charlevel=True. This means that the tokenizer will treat characters (not words) as individual tokens.The size of the vocabulary is the number of distinct characters for which the mapping was created. Padding to length 50 has been done(although not required for the dataset) to ensure constant length of the sequence.

Experimentations and Results

The training dataset was split into training and validation dataset for by making a split of 20% on te training dataset. The validation dataset provided was used as test dataset which was never shown to the model and was used for finding the accuracy of the models.

KNN, Logistic regression, XGBoost(Boosted Trees) and Random Forest :
Basic models, namely KNN, Logistic regression, XGBoost(Boosted Trees) and Random Forest were used to test the make preidctions. Preprocssing was done using one-hot encoding method. The best parameters for these model were found using GridSearchCV library in Scikit-Learn. As expected, these models work well with non-sequential data and the accuracy obtained was not very appreciable, but motivated us to look into models that could capture contexual relations.

Model	Accuracy (%)
Logistic Regression	64.83%
KNNs	56.24%
XGBoost	68.71%
Random Forest	64.21%

Table 4: Validation Set Accuracy of Different Models

Implementation pipeline :
The flow of the models tested below the following pipeline :

1. Preprocessing the string of length 50 using Tokenizer
2. Using Embedding library with appropriate embedding dimensions to capture semantic relations.
3. Using a suitable model to extract features.
4. A fully connected linear model at last that gives a single valued output on which sigmoid is used. Threshold of 0.5 is used for making a binary prediction.

For capturing contexual relations and semantics, an embedding layer has to be used that could capture the semantics of the occurrences of characters.
RNN (Recurrent Neural Networks)
A simple RNN architecture was used with 64 neurons was used. Manual hyperparameter tuning was done and an embedding of dimension 50 was used(one character in the string was mapped to a vector of length 50). A batch size of 16 was used.

1. **Input Layer:** The input consists of sequences of word indices with shape (batch size, 50).

2. **Embedding Layer:** Output Shape: $(\text{batch\_size}, 50, 50)$
3. **SimpleRNN Layer:** Output Shape: $(\text{batch\_size}, 64)$
4. **Dense Output Layer:** Output Shape: $(\text{batch\_size}, 1)$

The accuracy of the model was found to be around 70%.

CNN (Convolutional Neural Networks)
While RNNs are powerful for modeling sequential data, CNNs can outperform them in various scenarios due to their feature extraction efficiency, and better handling of local patterns and long-range dependencies.

Architecture :

Layer	Type	Parameters	Activation Function
Embedding	Embedding Layer	Input Dim: vocab size	N/A
		Output Dim: 50	
		Input Length: 50	
Conv1D (Layer 1)	Convolutional Layer	Filters: 32	ReLU
		Kernel Size: 3	
Conv1D (Layer 2)	Convolutional Layer	Filters: 16	ReLU
		Kernel Size: 3	
Conv1D (Layer 3)	Convolutional Layer	Filters: 8	ReLU
		Kernel Size: 3	
Conv1D (Layer 4)	Convolutional Layer	Filters: 8	ReLU
		Kernel Size: 3	
Dropout	Dropout Layer	Dropout Rate: 0.5	N/A
Flatten	Flatten Layer	Converts multidimensional to 1D	N/A
Dense (Output)	Fully Connected Layer	Units: 1	Sigmoid

Table 5: CNN Model Architecture and Parameters

The above hyperparameters were found experimentally by manual finetuning. The convolution layers captures the contexual relations in the given string. The number of filters, kernel size and dropout parameters are completely experimental keeping the constraint of 10,000 trainable parameters into consideration. The best possible embedding size was found via experimentaion by plotting graph of embedding size vs validation accuracy.

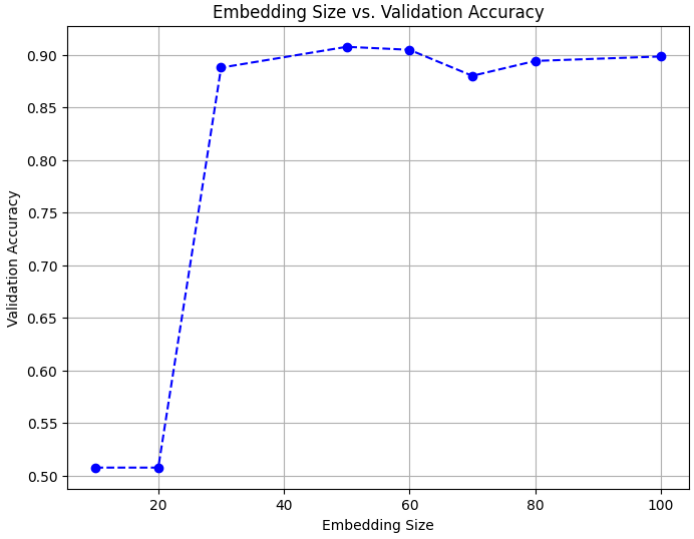

Figure 4: Embedding size vs validation accuracy

The dropout layer improves generalization and prevents overfitting by randomly disabling a proportion of neurons during training, encouraging the network to learn more robust features. A dropout of 0.5 means that 50% of the weights were dropped randomly.

The dense layer at last is similar to using a Logistic Regression classifier on the features obtained after the transformations from the convolution. Experimentation were done and different classifiers were used on the features obtained,

namely Randomforest classifier, XGBoost classifier, SVM and Logistic Regressor. Upon using various classifiers on the features at the last layer before the dense layer, the following results were obtained :

Model	Accuracy (%)
CNN	**93.66%**
CNN + Logistic Regression	92.22%
CNN + SVM	91.61%
CNN + Random Forest	87.11%
CNN + XGBoost	88.54%

Table 6: Validation Set Accuracy of Different Models

To test the robustness of the above models so obtained, they were trained on 0.2, 0.4, 0.6, 0.8 and 1.0 fraction of the training dataset. The base CNN model performed the best even on smaller fractions of the training dataset. The below graph plots the validation accuracies when other classifiers were used on the features extracted using CNNs(after the dropout layer).

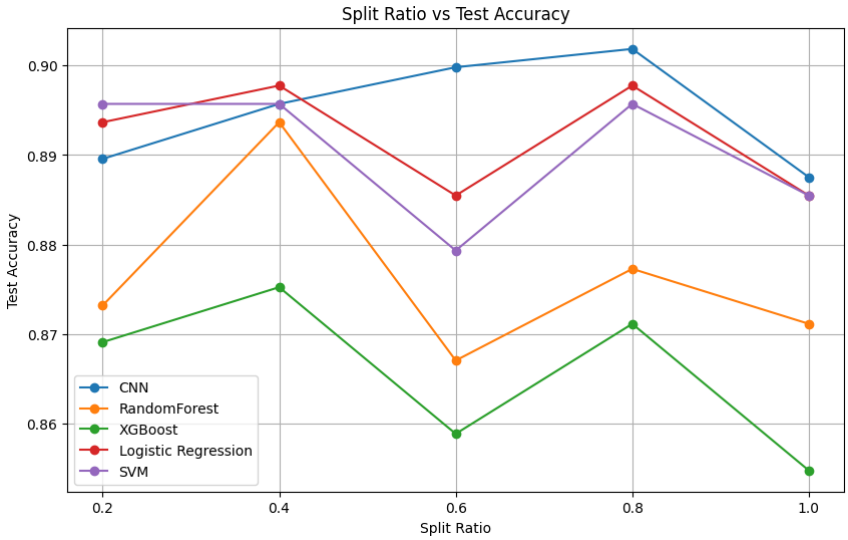

Figure 5: Split ratio vs test accuracy

The above experiments proved that the base CNN model is the best model in terms of amount of data required for training and accuracy obtained.

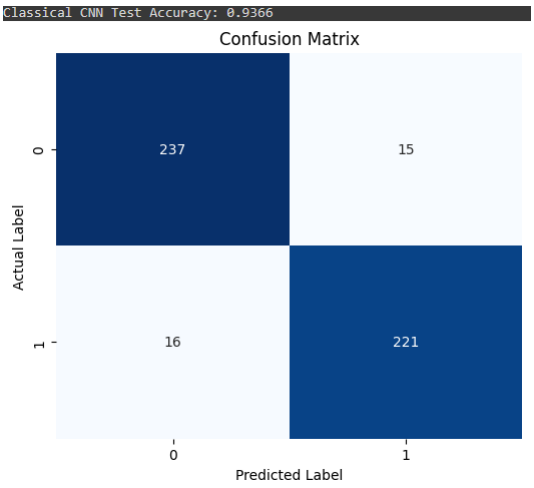

Figure 6: Confusion matrix for CNN model

2 Task-2

2.1 Preprocessing

The preprocessing pipeline includes three datasets: **Emoticons**, **Deep Features**, and **Text Sequences**, each processed differently to prepare for modeling.

Emoticons Dataset uses `LabelEncoder` to convert emoticons into numerical labels. This transformation helps the model treat the emoticons as categorical data, making them easier to process.

Deep Features Dataset involves flattening the 3D feature array into a 2D array to make it compatible with machine learning algorithms while preserving essential information.

Text Sequences Dataset is processed using `CountVectorizer`, which converts text sequences into bigram token counts, with a maximum feature size of 1000. This tokenization helps the model capture character-level patterns and context from the text.

Other Feature Engineering Methods

1. **StandardScaler** is used to standardize the features by removing the mean and scaling to unit variance. This ensures that all features are on a comparable scale, which is crucial for models like logistic regression and SVM that rely on distance metrics. Standardization also improves convergence rates, performance, and regularization across models. It is applied across all models to ensure balanced feature importance and prevent any feature from dominating due to its scale.

2. **PCA (Principal Component Analysis)** is employed for dimensionality reduction, transforming the data into uncorrelated variables (principal components) while retaining maximum variance. This method reduces noise and enhances performance by focusing on the most informative features. By reducing dimensions, PCA simplifies data visualization and decreases computational load, leading to faster training times. As the number of PCA components increases, accuracy generally improves because the model captures more variance and information. However, it's essential to monitor for overfitting as too many components might lead to excessive model complexity.

PCA Components	Average Accuracy	Accuracy (100% Dataset)
20	96.82%	96.93%
50	97.36%	97.96%
200	98.02%	98.36%
500	98.17%	98.98%
1000	98.17%	98.98%

Table 7: Summary of Average Accuracy and Accuracy (100% Dataset) for Logistic Regression with Various PCA Components

3. **SMOTE (Synthetic Minority Over-sampling Technique)** addresses class imbalance by generating synthetic samples for the minority class. It does so by interpolating between real minority class samples and their nearest neighbors. This technique is particularly useful when the dataset has a significant imbalance, as it improves the model's ability to learn patterns in the minority class. SMOTE helps improve the recall and f1-score for the minority class, which is often essential for real-world tasks.

However, in many cases, the model trained without SMOTE performs better in terms of accuracy. This is because the non-SMOTE model learns from the real distribution of the data, which can lead to better generalization on unseen data. Furthermore, the introduction of synthetic data in SMOTE can add noise, which might degrade accuracy, even though it helps balance the class distribution. Non-SMOTE models also benefit from effective regularization, reducing overfitting and improving test performance.

We have used SMOTE for over-sampling and then reduced the number of features using PCA(number of components=1000) and applied logistic regression. Below table represents the comparison between accuracy of SMOTE vs NON-SMOTE for 1000 PCA components.

Training Data Size	SMOTE Accuracy	Non-SMOTE Accuracy
20%	94.07%	96.73%
40%	96.73%	98.16%
60%	97.96%	97.55%
80%	98.16%	98.57%
100%	98.98%	98.98%

Table 8: Accuracy Comparison Between SMOTE and Non-SMOTE Models for 1000 PCA Components

2.2 Experimentaion and Results

2.2.1 Logistic Regression Model

This section discusses the application of the Logistic Regression model in our machine learning experiments, highlighting the importance of regularization and its impact on performance. We first applied StandardScaler(for normalising) and then PCA(achieved best accuracy with number of PCA components=1000). After that applied Logistic Regression model.

Regularization Basics:

Regularization helps prevent overfitting by adding a penalty to the loss function based on the size of the coefficients. In the case of C:

A smaller C (e.g., 0.1) means stronger regularization, which can be beneficial when the dataset is small, as it encourages the model to be simpler and prevents it from fitting noise. Conversely, a larger C (e.g., 1) means weaker regularization, allowing the model to fit the training data more closely.

Small Datasets:

When the dataset is small, there's a higher risk of overfitting. A smaller C (0.1) can help by constraining the model and forcing it to focus on the most important features, leading to better generalization on unseen data.

Large Datasets:
With a larger dataset, the model has more data to learn from, which can help it to identify patterns more effectively. In this case, a larger C (1) allows the model to fit the data more closely without risking overfitting as much since there's more information available to guide the learning process.

Bias-Variance Tradeoff:
A smaller C can lead to higher bias but lower variance, making it suitable for smaller datasets. Conversely, a larger C can reduce bias and allow the model to capture more complexity, which is beneficial when there's sufficient data to support it.

Results:
Table 9 summarizes the dataset sizes and their corresponding best hyperparameters when PCA $= 1000$.

Table 9: Best Parameters for PCA = 1000

Dataset Size	**Best Hyperparameter** C
20%	0.1
40%	0.1
60%	0.1
80%	1.0
100%	1.0

Logistic Regression acheives highest accuracy of **98.98%** for number of PCA components=1000 and total training data of size 100%. It can be observed that, as the datasize for training increases, model accuracy increases.

2.2.2 CNN Model

After preprocessing through normalization and PCA, we applied the same CNN architecture used in the first dataset. The table below presents the accuracy corresponding to various training data sizes and the number of PCA components=1000(highest Average Accuracy for different dataset sizes): 96.56

Table 10: Model Accuracy with Different Training Sizes and PCA Components

Training Data Size in %	**Number of PCA Components**	**Accuracy in %**
20	1000	94.07
40	1000	96.73
60	1000	96.93
80	1000	98.36
100	1000	96.73

- It can be observed that the model accuracy is increasing as training data size increases but drops when it becomes 100% which is not the case for Logistic Regression model. The contrasting accuracy trends between CNNs and logistic regression as dataset size increases stem from model complexity and overfitting. CNNs, being more complex, can capture intricate patterns, but may not improve performance significantly with larger datasets due to overfitting or capacity saturation. Conversely, logistic regression, being simpler, can continually benefit from more data, refining its estimates without as much risk of overfitting. As a result, CNNs may struggle to generalize, leading to stagnating or declining accuracy, while logistic regression often maintains robust performance with increased data sizes.

2.2.3 Random Forest

After conducting preprocessing through normalization and PCA, we implemented the Random Forest algorithm on the dataset. The model achieved an average accuracy of approximately 93%, with the highest accuracy reaching around 95% when utilizing 100% of the training data. Also, it takes higher training time compared to the other two models

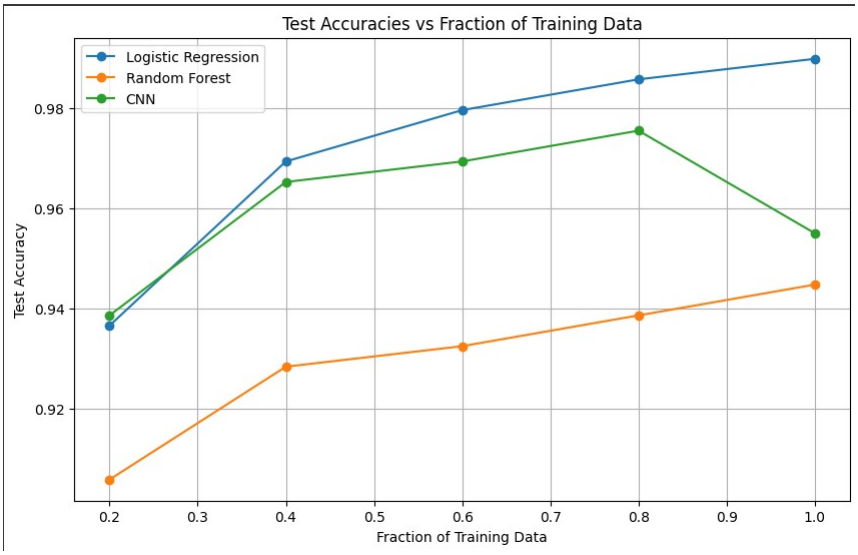

Figure 7: Comparison of Random forest,CNN and Logistic regression through a line plot

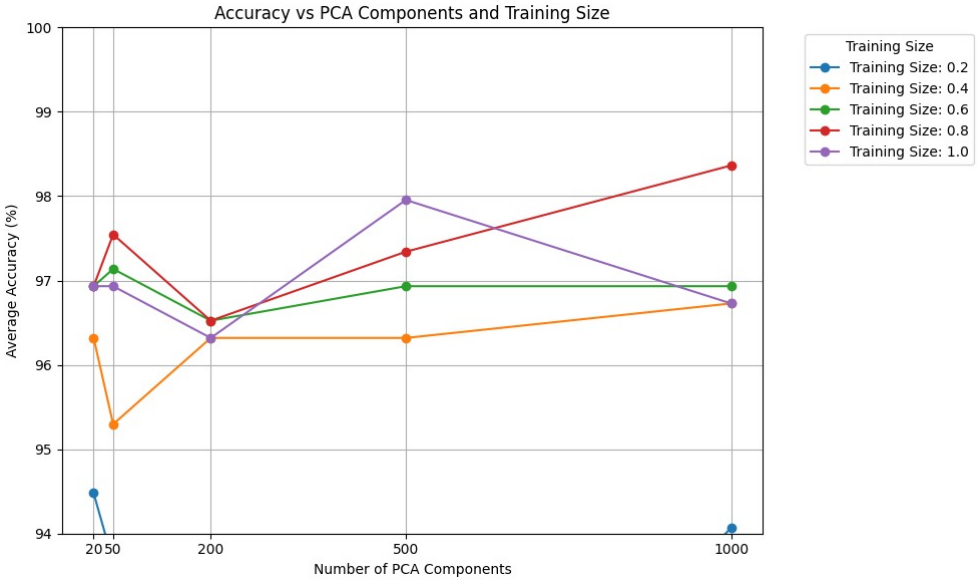

Figure 8: CNN Accuray w.r.t. PCA and training data size

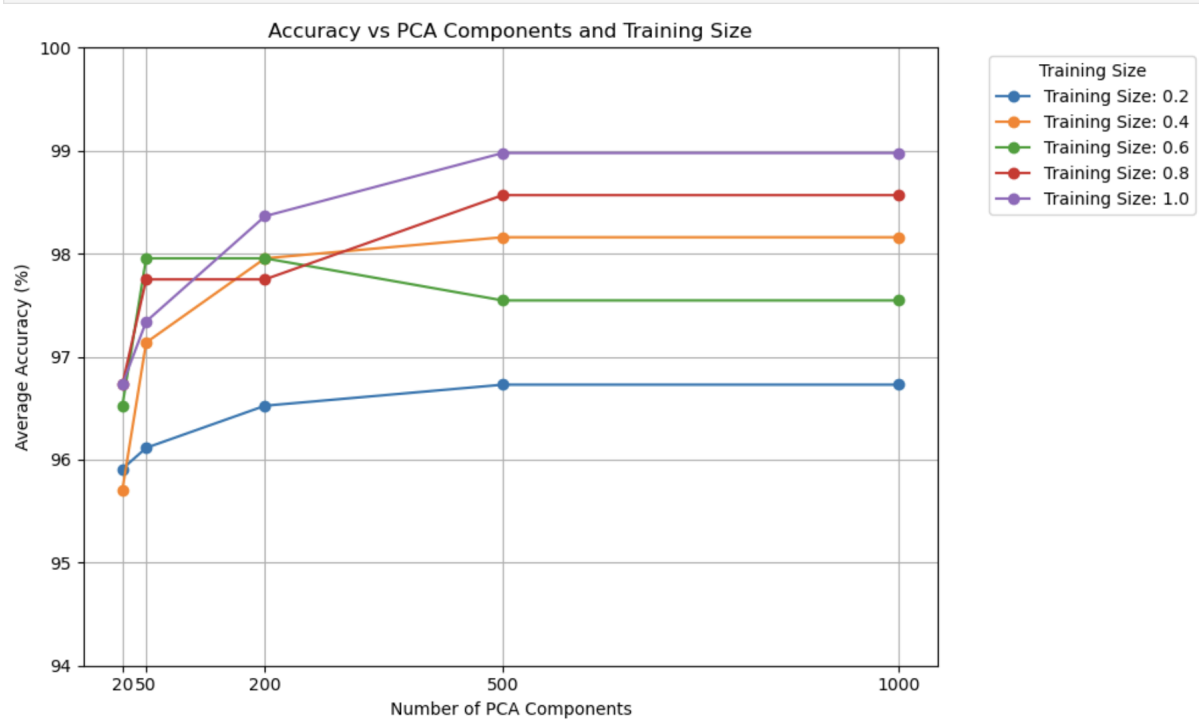

Figure 9: Logistic Regression Accuray w.r.t. PCA and training data size

References

[1] Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer Series in Statistics.
[2] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
[3] Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:1412.6980.

[4] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[5]Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[6]Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[7]Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94, 2012.

[8]Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.