**LG SUMMER INTERNSHIP REPORT**

***Multi-Class Classification of Entity Fields in Data Using Feature Driven Machine Learning***

***TABLE OF CONTENTS***

|  |  |  |
| --- | --- | --- |
| ***S. No.*** | ***Topic*** | ***Page No.*** |
| 1 | *List of Figures and Tables* | ii |
| 2 | Abstract | iv |
| 3 | Chapter 1: Introduction | 1 |
| 4 | Chapter 2: Review of Literature | 2 |
| 5 | Chapter 3: Material and Methods | 5 |
| 6 | Chapter 4: Results and Discussion | 22 |
| 7 | Chapter 5: Conclusion | 30 |
| 8 | References | 31 |

***LIST OF FIGURES AND TABLES***

|  |  |  |
| --- | --- | --- |
| ***Figure No.*** | ***Figure Legend*** | ***Page No.*** |
| Figure 1 | First Names Dataset (Kaggle) | *5* |
| Figure 2 | Full Names and Regions generated using Mostly AI | *5* |
| Figure 3 | Complete synthetic data | *6* |
| Figure 4 | Dataset with labels | *6* |
| Figure 5 | Python code for inserting noise | *9* |
| Figure 6 | Noisy Data | *10* |
| Figure 7 | Python code for Data Cleaning | *10* |
| Figure 8 | Python Code for Feature Engineering | *14* |
| Figure 9 | Python code for Label Encoding | *14* |
| Figure 10 | Python code for Feature EDA | *16* |
| Figure 11 | Python code for Model fitting | *17* |
| Figure 12 | Python code for fine-tuning using grid search | *18* |
| Figure 13 | Python code for feature selection | *20* |
| Figure 14 | Python code for model validation and ROC curve | *21* |
| Figure 15 | Data overview and result after cleaning | *22* |
| Figure 16 | Engineered features and their values stored in excel file | *22* |
| Figure 17 | Box plots for all continuous features | *25* |
| Figure 18 | Summary statistics for continuous features | *25* |
| Figure 19 | Class wise means for continuous features | *25* |
| Figure 20 | Correlation matrix and heatmap for continuous features | *26* |
| Figure 21 | Value count for categorical features | *26* |
| Figure 22 | Class wise cross tabs | *26* |
| Figure 23 | Initial Model Fitting Results | *27* |
| Figure 24 | Grid Search results | *28* |
| Figure 25 | Feature Selection results | *28* |
| Figure 26 | Validation set results | *29* |
| Figure 27 | Predictions on validation set | *29* |
| Figure 28 | ROC Curve | *29* |
| Table 1 | Engineered features | 11 |

***ABSTRACT***

Classification of texts is a key aspect in automated systems where user data processing, document verification, and data consistency are concerned. This internship project involves constructing an effective multi-class classification model which would correctly identify and then classify a text entity into the class it belongs.

For our project, we decided to use a dataset which would realistically mimic a real dataset of Indian names, emails and addresses which was successfully incorporated using tools like Kaggle and Mostly AI. The final dataset had three labelled classes: 0 – Name, 1 – Email, and 2 – Address.

20 hand-coded features were designed for every text record which would serve the purpose of differentiating the classes based on the text details. Exploratory data analysis was performed to find out about class distributions, detect inconsistencies, and check the discriminative ability of features through statistical tests.

Ultimately, we fitted our classification model which was a Random Forest Classifier. Here, using the concepts of Bootstrap Sampling and Out-of-bag testing, we calculated performance metrics and scored our model. Furthermore, we fine-tuned the hyperparameters of our model using grid search and performed dimensionality reduction by selecting the top performing features. The resulting model was tested and confirmed with another set of unseen data, showing robust multi-class functioning.

**Chapter – 1**

**INTRODUCTION**

In today’s digital world, huge amounts of user-provided data are produced every second. Such data may comprise of values like names, email addresses, and physical addresses, which are important for various purposes like identification, communication, and shipping.

Such inputs belong to **the semi-structured textual data** category — data that, though not entirely unstructured in nature like free-form text, does not follow a strict schema either. This makes such data hard for conventional systems to process, classify, or validate efficiently. Conventional rule-based algorithms tend to fall short here because of issues like –

* varied formatting,
* language and punctuation differences,
* inclusion of human error, etc.

With an understanding of the increasing demand for auto-classification of such noisy textual data, this project was conducted as a part of my summer internship. The project was intended to investigate a machine learning–based solution to this issue, particularly in how to extract structured patterns from unstructured input and how classification models can learn to distinguish between highly similar textual information.

**Chapter – 2**

**REVIEW OF LITERATURE**

The aim of this project is to create a robust and explainable **machine learning** solution for the **classification** of **semi-structured textual data** into pre-specified classes: names, email addresses, and physical addresses. The goal is to solve the real-world problem of dealing with user-sourced data that is not consistent in structure, formatting, and quality which is an endemic problem of digital systems that use user inputs for identification, communication, or delivery of services.

The project centres on developing a **classification pipeline** that is able to classify text entries with a similar appearance by learning structural patterns and linguistic characteristics. This includes –

* creating an indicative dataset that resembles imperfections in the real world,
* parsing significant features from raw text input,
* training a supervised learning model to classify the data with a high degree of accuracy. There is a specific focus on using ensemble learning methods, like Random Forests, due to their interpretability and strength.

Apart from model creation, the project also seeks to mimic real-world data environments through the injection of controlled noise and typographical errors into the data set. The ultimate purpose is to test the efficacy of the method on unseen data and show the potential of the approach in systems that need automated data validation, smart form management, or preprocessing of information provided by users.

**1. RANDOM FOREST**

Random Forest is a **supervised ensemble learning** method that is widely applied for classification and regression. It creates many decision trees and combines their predictions to get a final result. Through the aggregation of different decisions made by multiple weak learners (decision trees), the algorithm prevents overfitting and increases predictive power, particularly on noisy or difficult datasets.

The Random Forest algorithm works by training a **set of decision trees** individually. Each tree is trained on a different subset of the training set, created by **bootstrap sampling**, i.e., each tree is presented with a slightly different version of the original data. When constructing a tree:

* At every node, a random subset of features is chosen.
* The tree computes all the selected features and selects the feature that gives the best split, commonly employing Gini impurity or entropy as the split criterion.
* This is done recursively until a termination condition is reached, e.g., reaching maximum depth, minimum number of samples at a node, or all samples at a node coming from the same class.

By incorporating **randomness** in both data and feature selection, Random Forest makes each tree uncorrelated from others. For prediction, all the trees vote for the class label, and the output is decided by **majority voting** (in case of classification problems). This ensemble method is effective in identifying diverse patterns in data and is very robust against outliers and noise.

**Important Parameters of Random Forest –**

* **n\_estimators:** Number of trees in the forest. Increasing numbers of trees generally enhance performance but add computation time.
* **max\_depth:** Restricts the depth of individual trees to limit overfitting.
* **min\_samples\_split** and **min\_samples\_leaf:** Specify the minimum number of samples needed to split a node or to be at a leaf node.
* **max\_features:** Best number of features for building the model.
* **bootstrap:** If we used bootstrap sampling in the model.
* **oob\_score:** If out of bag samples are used in the model.

These parameters must be carefully tuned to balance complexity and generalization.

**2. BOOTSTRAP SAMPLING**

Bootstrap sampling is an important statistical technique for creating different versions of a dataset. In Random Forest, it ensures that each decision tree receives a different training set. On average, about **63%** of the original data is included in each bootstrap sample, while the remaining **37%** is excluded (these become out-of-bag samples). This technique introduces variability between trees, which is crucial for ensemble learning to work effectively.

**Out-of-Bag (OOB) Technique**

The OOB samples are utilized to estimate the performance of the tree on new data. OOB is a method that offers an unbiased internal model accuracy estimate without needing an additional validation set. By combining predictions for all OOB samples over all trees, the model is able to calculate an **OOB score**, which serves as a good measure of generalization performance.

**3. GRID SEARCH**

Grid Search is the most widely used method for performing **hyperparameter tuning**. It tries all the combinations of given parameter values and finds the best combination with respect to a **scoring criterion**, e.g., accuracy or OOB score. In this project, Scikit-learn's **GridSearchCV** was utilized to tune Random Forest parameters. **Cross-validation** was used while doing this to make sure that the parameters chosen are performing well on unseen data.

**4. ROC CURVE**

The ROC (Receiver Operating Characteristic) Curve is a graphical plot of the **True Positive Rate (Sensitivity)** against the **False Positive Rate (1 - Specificity)** at different threshold levels. It is an effective diagnostic tool to check how well a model can discriminate among classes. For multiclass situations, the **One-vs-Rest (OvR) approach** is employed, where an individual ROC curve is graphed for each class. **AUC** (Area Under the Curve) is utilized as a measure to measure performance — the higher the AUC, the better the separability of classes.

. **Chapter – 3**

**MATERIAL AND METHODS**

**STEP 1 : DATA COLLECTION**

The first step of the project was collecting data that best reflects actual user-inputs in three categories: names, email addresses, and physical addresses. To generate realistic yet controlled data, a hybrid method was adopted using both open-source data and AI-based synthetic data.

Starting with names, Indian **first names** were obtained from a public **Kaggle** dataset comprising varied Indian name entries.

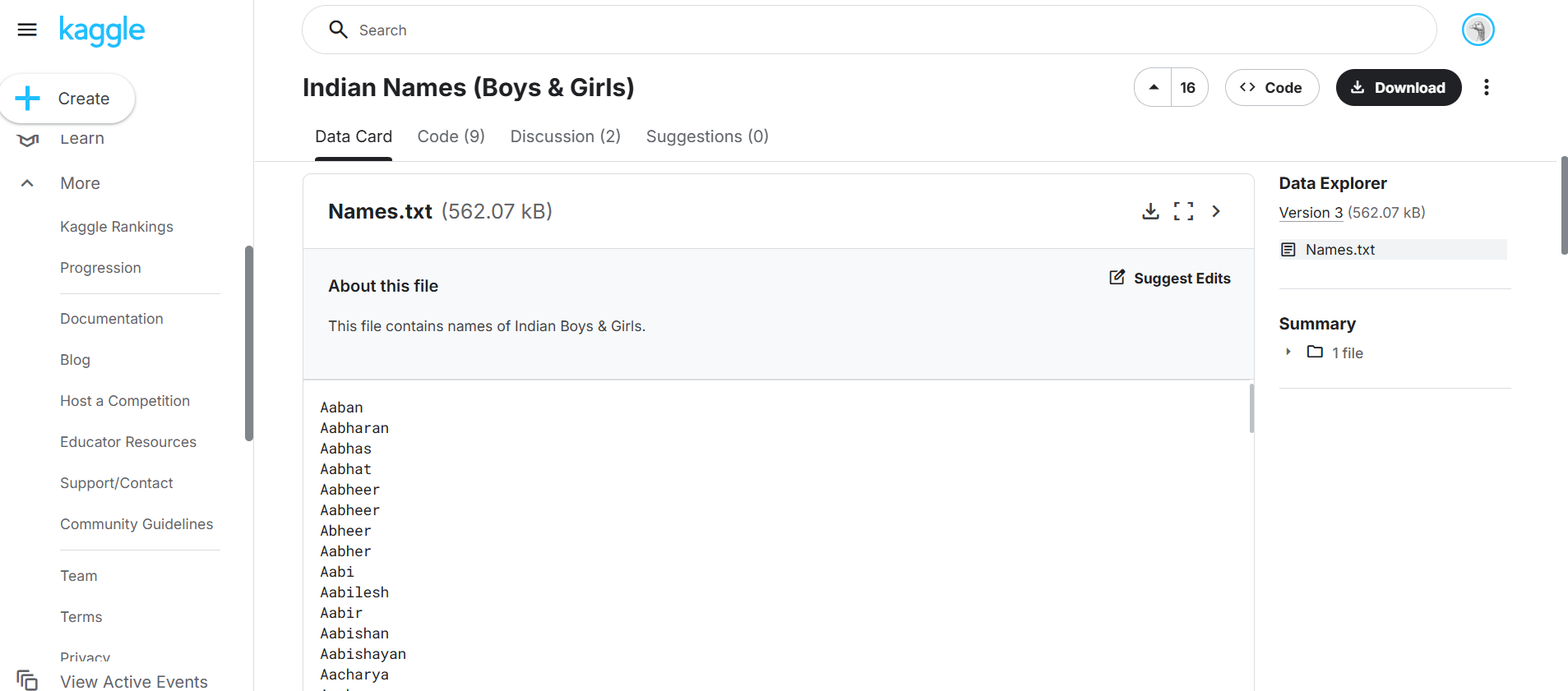


Figure : First Names Dataset (Kaggle)

To supplement this, **Mostly AI,** a synthetic data generating platform was employed to create artificial but very realistic **surnames**. Based on the so obtained full names, we again used the AI platform to obtain the most appropriate **state/region** for the person.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Full Names and Regions generated using Mostly AI

Finally, **synthetic addresses** based on the state names and **synthetic emails** based on the full names were generated. This allowed for natural variations present in real submissions to be replicated in the dataset, such as regional differences and format deviations.

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Complete synthetic data

**STEP 2 : DATA PREPARATION**

Once the raw data had been gathered, the next step was to transform it into a common form ready for further handling. All produced entries were merged into one dataset with **two columns**: one of the **text value**, and another of its respective **label** –

* 0 for Name,
* 1 for Email,
* 2 for Address

The dataset was then stored in Excel format.

A screenshot of a computer

AI-generated content may be incorrect.

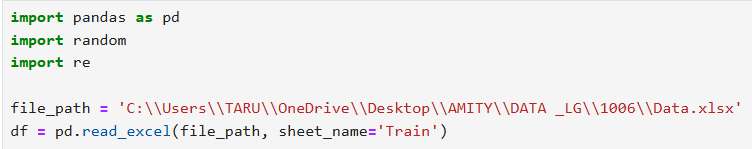
Figure : Dataset with labels

Each row was made independent of consistent encoding, and free of formatting anomalies that were introduced during merging.

**STEP 3 : DATA MANIPULATION**

In order to mimic the inconsistencies commonly experienced in authentic user inputs, **15%** of the data — distributed evenly across all three classes — was manually manipulated in order to introduce noise. Two different techniques were employed for introducing these errors :

In the first method, a personalized **Python script** was employed to inject errors programmatically.





A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

A close-up of a computer code

AI-generated content may be incorrect.

Figure : Python code for inserting noise

In the second method, data entries were **manually shuffled** within the classes. For example, some emails were falsely labelled as 0 or 2 instead of 1, which may be part of a human error. Similar false labelling was introduced for names and addresses.

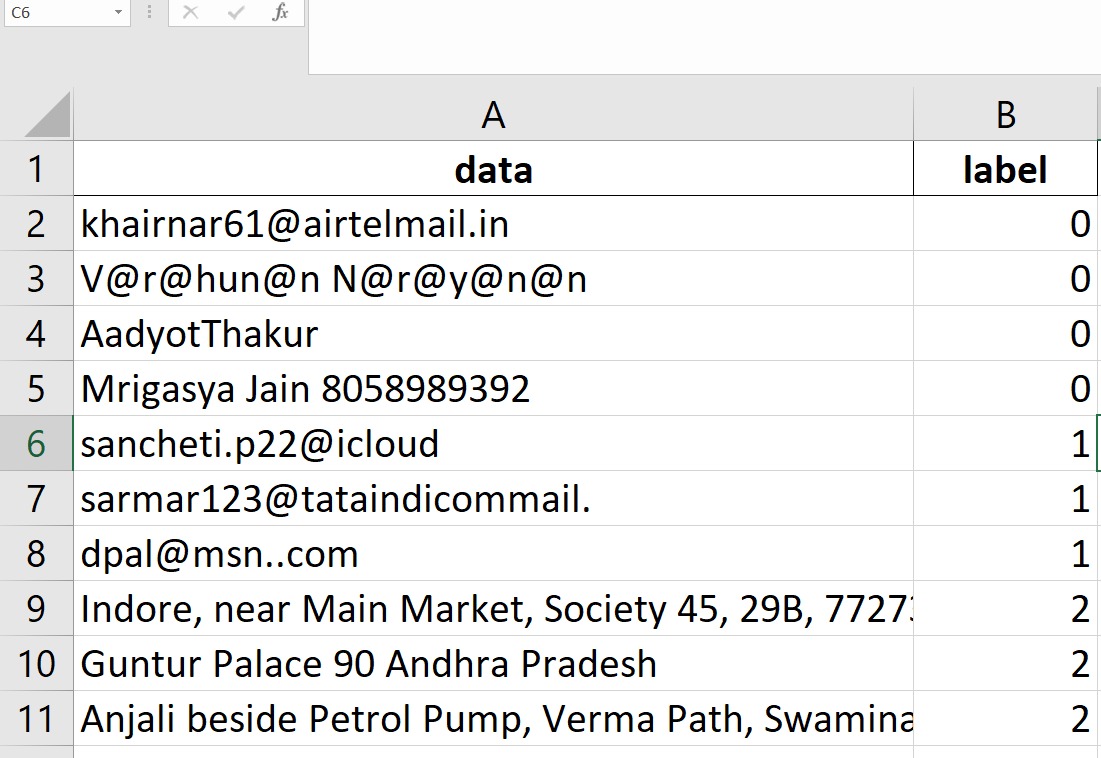


Figure : Noisy Data

**STEP 4 : DATA CLEANING**

After the dataset was ready, data cleaning was carried out to maintain uniformity without compromising the intentionally introduced flaws. There were 2 major operations carried out:

* null value deletion, and
* duplicate record elimination.

A screenshot of a computer code

AI-generated content may be incorrect.

Figure : Python code for Data Cleaning

These were done to ensure that the dataset was free from **empty and redundant records** that may skew the learning process.

**STEP 5 : FEATURE ENGINEERING**

To facilitate efficient classification of data, a collection of 20 hand-designed features was created for every data entry. These features were intended to extract structural and pattern-based features that are typically present in our data. The features were calculated and stored in a new excel sheet.

|  |  |  |
| --- | --- | --- |
| **Feature Name** | **Feature Type** | **Description** |
| string\_length | Continuous | Total characters that are present in the string |
| word\_count | Continuous | Number of words separated by spaces |
| space\_count | Continuous | Count of spaces in the text |
| digit\_count | Continuous | Number of numerical digits |
| alphabet\_count | Continuous | Total number of alphabetic characters |
| special\_char\_count | Continuous | Count of special characters (e.g., @, #, .) |
| vowel\_count | Continuous | Number of vowels in the string |
| consonant\_count | Continuous | Number of consonants in the string |
| at\_count | Continuous | Number of @ symbols in the text |
| dot\_count | Continuous | Number of periods (.) in the text |
| has\_consecutive\_specialchar | Boolean | Indicates if special characters appear consecutively (e.g., .., @@) |
| has\_email\_regex\_pattern | Boolean | If matches standard email patterns |
| has\_city\_keyword | Boolean | True if city-related keywords (e.g., "Nagar", "Vihar") are detected |
| has\_state\_keyword | Boolean | True if text contains known Indian state names |
| has\_pincode\_pattern | Boolean | True if text contains a valid 6-digit PIN code |
| has\_valid\_TLD | Boolean | True if email ends with a valid top-level domain (e.g., .com, .in) |
| name\_length\_cluster | Boolean | Cluster category based on name length |
| name\_length\_category | Categorical | Textual label for name length category (e.g., “compact”, “extended”) |
| starts\_with\_char\_type | Categorical | Character type of the first character (e.g., digit, letter, symbol) |
| ends\_with\_char\_type | Categorical | Character type of the last character (e.g., digit, letter, symbol) |

*Table 1 : Engineered features*

 A screenshot of a computer code

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.  

Figure : Python Code for Feature Engineering

**STEP 6 : LABEL ENCODING**

Following feature engineering, a few of the newly created features were **categorical** in nature and had to be transformed into **numerical format** prior to being utilized for model training. This was accomplished by **label encoding** on the last three categorical features.

Each distinctive category in these features was translated into an integer through the utilization of **scikit-learn's LabelEncoder**. It was carried out to keep all features, whether initially numerical or transformed, in a format that was compatible for feeding into the classification model.

A screenshot of a computer code

AI-generated content may be incorrect.

Figure : Python code for Label Encoding

**STEP 7 : FEATURES EDA**

Feature-wise EDA was done to identify the behaviour and importance of every engineered feature among the three target classes. For the **continuous features**, **class-wise means** and **summary statistics** were calculated, and a **correlation matrix** was generated to identify multicollinearity. Distribution patterns were visualized using **boxplots** across labels. For **Boolean** and **categorical** features, **value counts** and **class-wise crosstabs** were produced in order to analyse the distribution and frequency of feature values per class.

A screenshot of a computer code

AI-generated content may be incorrect.

A computer code with many text

AI-generated content may be incorrect.

Figure : Python code for Feature EDA

While some insights have been acquired, *no feature selection was carried out here*. The project instead used a model-driven feature selection approach later, via Random Forest's importance ratings and out-of-bag (OOB) performance.

**STEP 8 : MODEL FITTING AND EVALUATION**

For the task of classification, a **Random Forest Classifier** was chosen because it is strong, interpretable, and can effectively deal with both categorical and continuous features. The model was trained on the full feature-engineered dataset both encoded categorical ones, and the target was the class label (0, 1, or 2) with **bootstrap aggregation (bagging)**.





Figure : Python code for Model fitting

Following training, the model's performance was evaluated with the **OOB error rate**, **confusion matrix**, and **classification report**. These gave an indication of how effectively the model separated the three classes. A **feature importance bar chart** was also created to help interpret which features were most important in making the classification decision, informing future feature selection and model reduction.

**STEP 9 : HYPERPARAMETER TUNING**

To improve the performance and generalization capability of the Random Forest model, **hyperparameter tuning** was performed using **GridSearchCV** from scikit-learn. In total, **72 combinations** were tested using **5-fold cross-validation**, with performance being measured in terms of **classification accuracy** and **out-of-bag (OOB)** score as evaluation measures.

A screenshot of a computer code

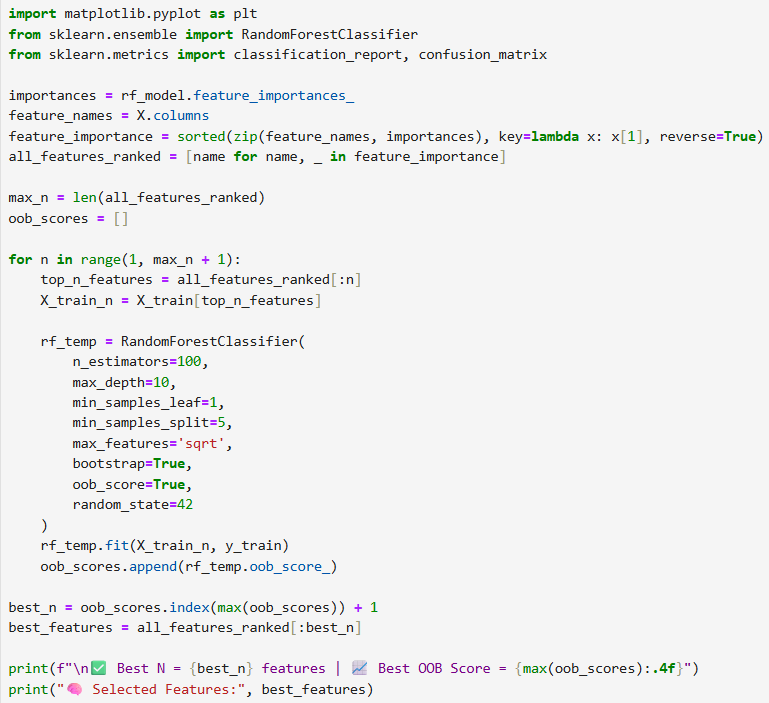
AI-generated content may be incorrect.

Figure : Python code for fine-tuning using grid search

The optimized model exhibited enhanced consistency and improved noisy input handling. These hyperparameters were then kept constant for subsequent operations like feature selection and final validation so that the process could remain stable with repeated runs.

**STEP 10 : FEATURE SELECTION**

To select the most effective features for classification, a systematic **feature selection process** was adopted using the **feature importance** **values** derived from the Random Forest model. The features were initially ranked in the order of declining importance. Subsequently, a **loop-based method** was used where the model was repeatedly trained iteratively with only the **top n** features (ranging from 1 to the number of features).



A screenshot of a computer program

AI-generated content may be incorrect.

Figure : Python code for feature selection

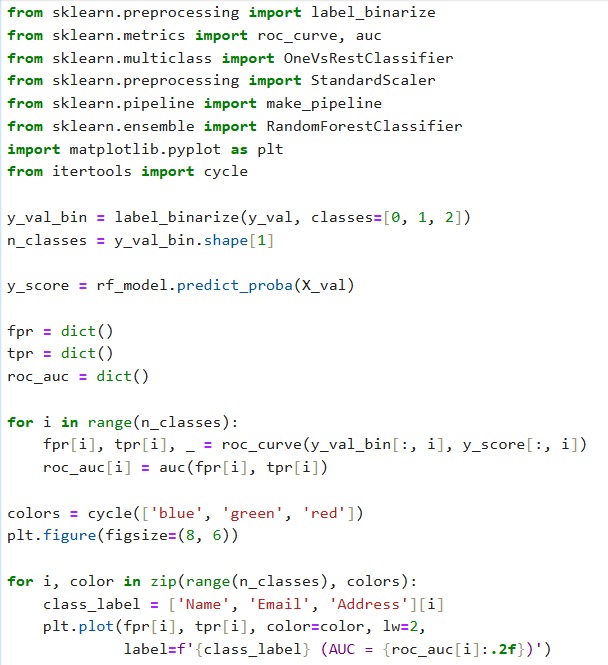
The **optimal feature number (N)** was determined from the peak OOB. A **graphical plot** of OOB Score against Number of Features was employed to find the point of performance saturation.

**STEP 11 : MODEL VALIDATION**

Finally, a distinct validation dataset was utilized that contained **75 unseen instances** (25 each from the Name, Email, and Address categories). The same preprocessing processes were performed on this set so as to maintain the uniformity. The model trained with the optimal feature set was utilized to predict.

Performance was measured based on a confusion matrix and classification report, which provided precision, recall, and F1-scores for every class.



A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure : Python code for model validation and ROC curve

**Chapter – 4**

**RESULTS** **AND DISCUSSION**

Random Forest model, trained on the engineered feature set, was able to generate a range of outputs showing its classification on the three target classes. The section contains some of the most important evaluation factors, which indicate how well the model classified each class.

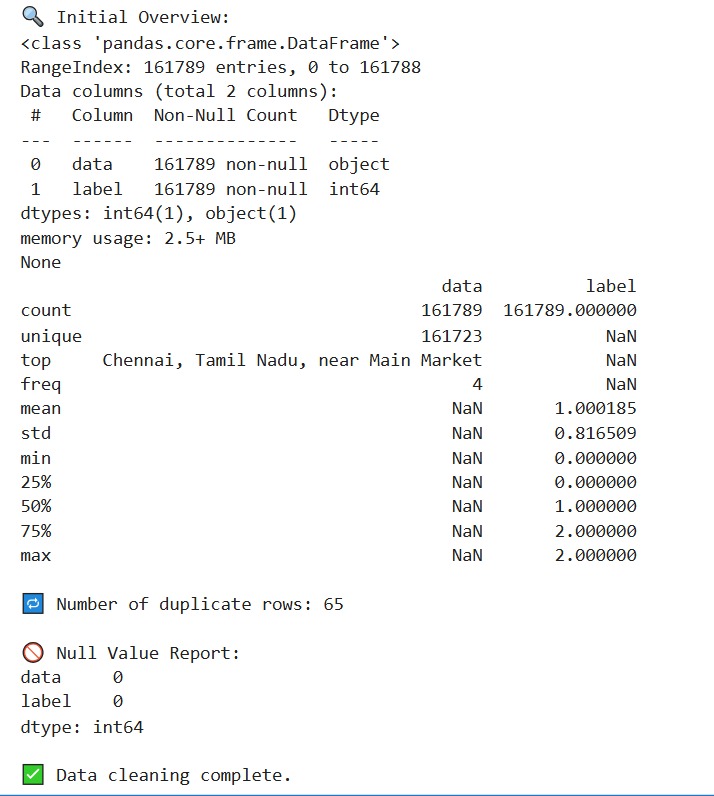
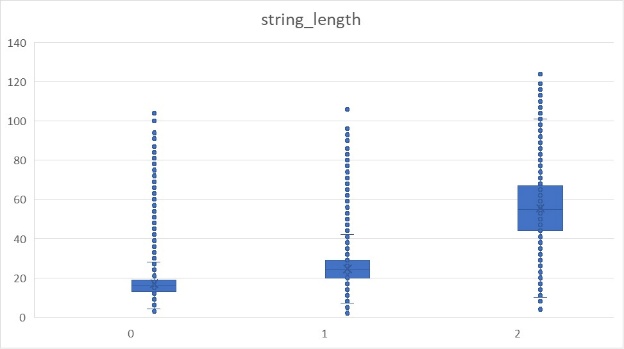


Figure : Data overview and result after cleaning

A white sheet with black numbers

AI-generated content may be incorrect.

Figure : Engineered features and their values stored in excel file

A graph with a blue square and a blue square

AI-generated content may be incorrect.A graph with blue dots

AI-generated content may be incorrect.

A graph with blue squares and numbers

AI-generated content may be incorrect.

A graph with blue squares and dots

AI-generated content may be incorrect.

A graph with blue and white lines

AI-generated content may be incorrect.A graph with blue and white lines

AI-generated content may be incorrect.A graph with blue squares and lines

AI-generated content may be incorrect.A screenshot of a graph

AI-generated content may be incorrect.A graph with blue squares and dots

AI-generated content may be incorrect.

Figure : Box plots for all continous features

A screenshot of a table

AI-generated content may be incorrect.

Figure : Summary statistics for continuous features

A table with numbers and letters

AI-generated content may be incorrect.

Figure : Class wise means for continuous features

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Correlation matrix and heatmap for continuous features

A screenshot of a data

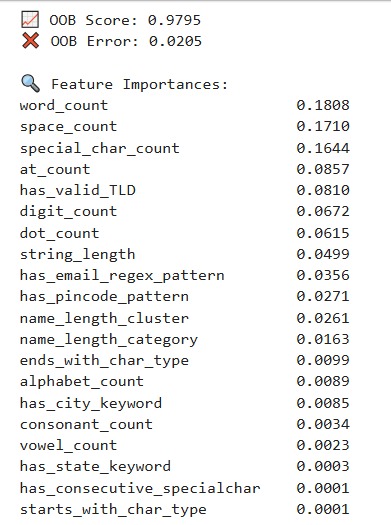
AI-generated content may be incorrect.

Figure : Value count for categorical features

A table of data with numbers and letters

AI-generated content may be incorrect.

Figure : Class wise cross tabs



A screenshot of a computer

AI-generated content may be incorrect.A graph with a bar graph

AI-generated content may be incorrect.

Figure : Initial Model Fitting Results



Figure : Grid Search results

A screenshot of a computer

AI-generated content may be incorrect.

A graph with green line

AI-generated content may be incorrect.

Figure : Feature Selection results

A screenshot of a computer

AI-generated content may be incorrect.

Figure : Validation set results

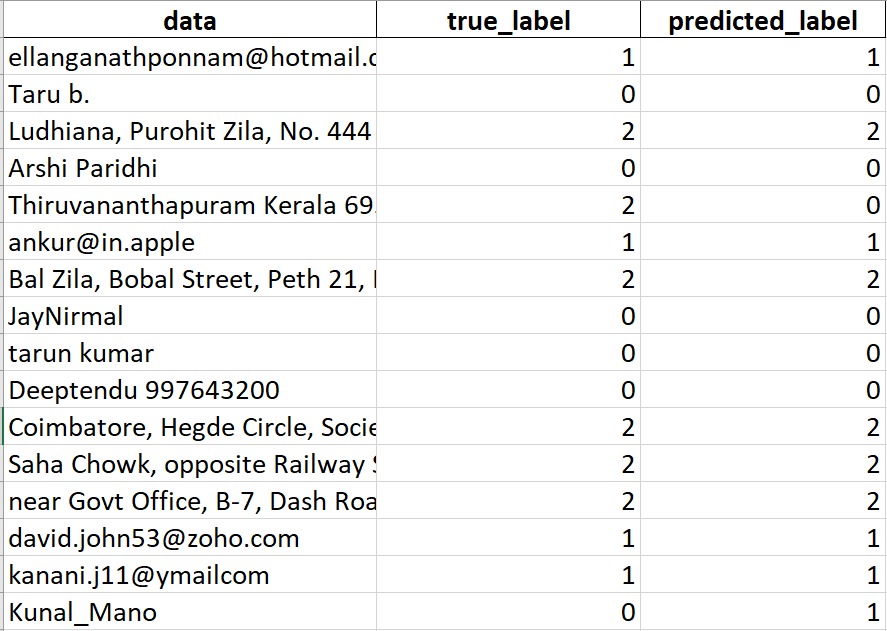


Figure : Predictions on validation set

A graph showing a positive rate

AI-generated content may be incorrect.

Figure : ROC Curve

**Chapter – 5**

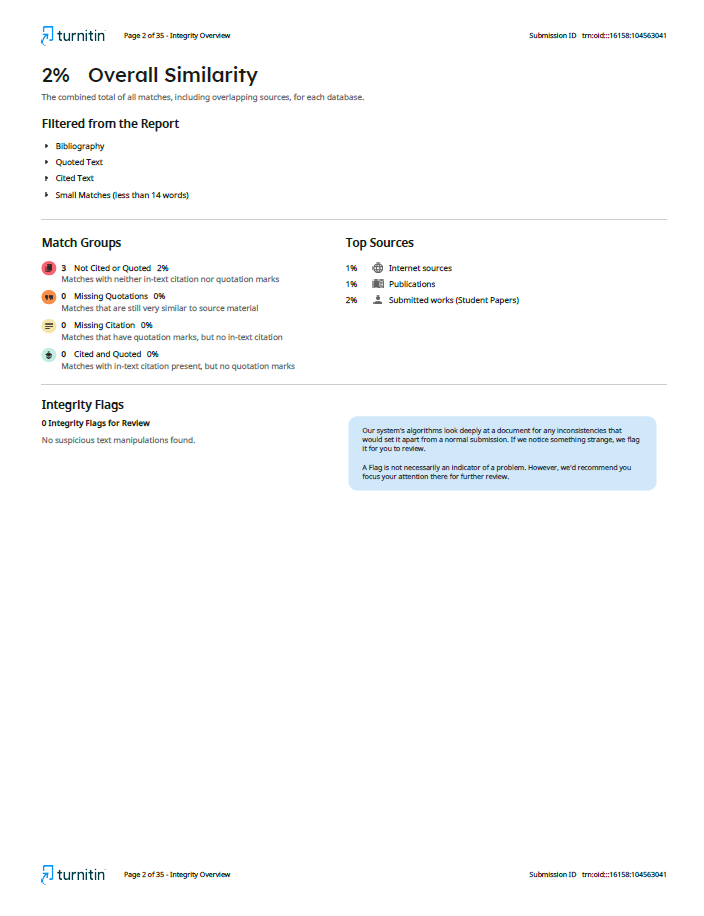
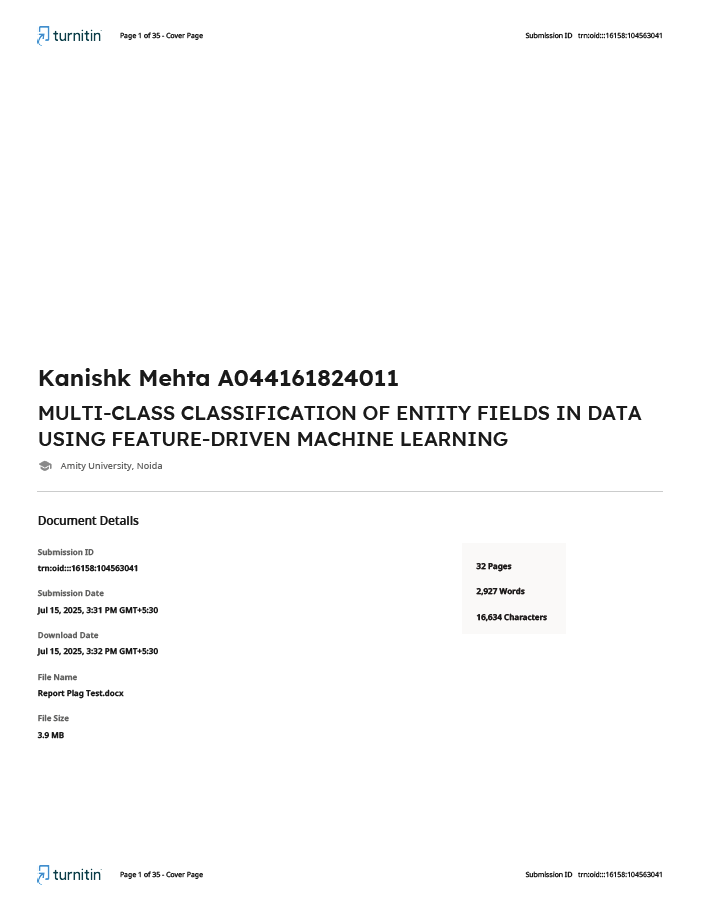
**CONCLUSION**

This project effectively met the goal of classifying semi-structured textual data—in particular names, email addresses, and physical addresses—using a Random Forest-based machine learning pipeline.

On the front of future potential, the existing pipeline can be expanded in a number of ways. One way is to use deep learning-based models such as LSTMs or transformer-based structures to learn more contextual meanings in text data. Another way is to extend the dataset to cover foreign entries or multilingual forms to make the model more adaptive to worldwide applications. Lastly, incorporating this classification model into a real-time form validation system can give instant feedback on malformed or misclassified entries while the user is inputting, enhancing data quality at the moment of collection.

***REFERENCES***

1. **Mostly AI.** Synthetic data platform. from <https://mostly.ai/>
2. **ChatGPT by OpenAI.** Conversational AI tool used for documentation - from <https://chatgpt.com/>
3. **Kaggle.** Source of public datasets for machine learning - from <https://www.kaggle.com/>
4. **Datacamp Blog.** Understanding Classification in Machine Learning - from <https://www.datacamp.com/blog/classification-machine-learning>
5. Ben Postance, **Email Header Classification Project -** from <https://bpostance.github.io/posts/header-classifier/>
6. Goodchild, M. F. & Li, L. (2012). **Assuring the quality of volunteered geographic information. *Transactions in GIS*, 16(4), 491–513 -** from <https://onlinelibrary.wiley.com/doi/10.1111/tgis.12522>

NT