Health Insurance Lead Prediction Using Machine Learning Algorithms

DISSERTATION

Submitted in partial fulfillment of the requirements of the
MTech Data Science and Engineering Degree Programme

By

Salman Khan
BITS ID No: 2020sc04719

Under the supervision of

Ganesh Balu Gholap
Vice President
Axis Bank Ltd.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA

March 10, 2023

Acknowledgements

I would like to express my deepest appreciation to ***Mr. Ganesh Balu Gholap***, my Supervisor (Axis Bank Ltd.) and Mentor, for his unwavering support and valuable suggestions during my thesis work. He was instrumental in reviewing my progress and providing valuable advice into the selected topic and the domain of Banking and Finance industry.

I am also grateful to ***Mr. A.R.K. Mishra***, my MTech Supervisor from BITS Pilani, for his valuable feedback and directions in completing my thesis work. He has been extremely understanding and insightful in supporting and directing my thesis work with his excellent academic expertise and guidance.

Furthermore, I would like to thank ***Mr. Avinash Mathew***, my ex-Supervisor (Tata Consultancy Services Ltd.) and mentor, for the understanding and support he provided during my MTech academic endeavor.

I also want to acknowledge the support and guidance of my ***teams*** and ***colleagues*** from Tata Consultancy Services Ltd. and Axis Bank Ltd. for their moral support during my thesis journey.

Lastly, I would like to express my gratitude to my ***family*** and ***friends*** for their unwavering support throughout my course of study.

Thank you all for making this unforgettable academic adventure possible.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

Certificate

This is to certify that the Dissertation entitled, *"**Health Insurance Lead Prediction Using Machine Learning Algorithms**"*, and submitted by ***Mr. Salman Khan***, ID No: ***2020sc04719*** in partial fulfillment of the requirements of DSECLZG628T Dissertation, embodies the work done by him/her under my supervision.

[illegible]

Signature of the Supervisor

Place: Bangalore

Date: 10-March-2023

Ganesh Balu Gholap
Vice President
Axis Bank Ltd.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANIFIRST SEMESTER 2022-23

DSECLZG628T DISSERTATION

Dissertation Title	:	Health Insurance Lead Prediction Using Machine Learning Algorithms
Name of Supervisor	:	Ganesh Balu Gholap
Name of Student	:	Salman Khan
ID No. of Student	:	2020sc04719

Abstract

Problem Statement

Banks are the backbone of the global financial system. The prosperity and all of banks have historically been proven to have a strong direct impact on the global economy. As such, banks are always on the lookout for additional revenue while minimizing campaign costs.

One of the most relevant opportunities arise from the concept of cross-sell. Banks can cross-sell their products to existing customers and increase the per-customer value.

Insurance products are becoming more popular in India. Based on this, banks are also trying to re-monetize their existing customers through products like Health and Term insurances.

Objective of Project

The project will be aimed at predicting the viability of leads for cross selling the bank’s insurance products to existing customers.

In this project, I will use the Kaggle dataset for Health Insurance Lead Prediction to create a solution to predict if the existing bank customer will opt in for Personal Loan as well.

Uniqueness

The project is unique in the way that this project will be based on the technology landscape of my organization so that the core parts of the solution implemented here is transferable to our environment. This technology combination is unique in the sense that there is no literature that solves this problem with this combination of tools/platforms.

Benefit to the Organization

- Re-monetization of existing customers.
- Increase in value-per-customer.
- Lower costs and efforts than acquiring new customers.

Scope of Work

- Formulate problem statement.
- Create a machine learning model to implement the proposed solution.
- Deploy the model in the selected technology landscape.

Resources Required

- Azure Paid Account
- Laptop with 4 GB RAM and at least 10 GB free space
- Jupyter Notebook
- Google Colab

Potential Challenges include data quality, adequacy, and technology integration.

Key Words:

Classification, Machine Learning, Banking, Financial Sector, Health Insurance, Lead Generation, Marketing Campaign, Optimization

List of Symbols & Abbreviations

CNN	–	Convolutional Neural Network
IQR	–	Inter-Quartile Range
PCA	–	Principal Component Analysis
ROC	–	Receiver Operating Characteristic
AUC	–	Area Under the Curve
RandomizedSearchCV	–	Randomized Search Cross-Validation
ML	–	Machine Learning
DWH	–	Data Warehouse
ETL	–	Extract-Transform-Load
IICS	–	Informatica Intelligent Cloud Services
UOL	–	Upper Outlier Limit
LOL	–	Lower Outlier Limit

List of Tables

List of Figures

CONTENTS

Table of Contents

Chapter-1: Introduction

1.1 Cross Sell Theory

Cross-selling in banking refers to the practice of offering additional products or services to existing customers. The idea behind cross-selling is that since banks have already established a relationship with their existing customers, it is more cost-effective to sell additional products to them than to acquire new customers.

Cross-selling in banking can involve offering a variety of products or services, such as credit cards, loans, insurance, and investment products, to customers who already have a checking or savings account. For example, if a customer has a checking account with a bank, the bank may offer them a credit card or a personal loan.

The cross-selling theory in banking suggests that by offering customers additional products and services, banks can increase revenue and profitability, while also improving customer loyalty and retention. This is because customers who have multiple products with a bank are less likely to switch to another bank.

However, it is important to note that cross-selling must be done ethically and in a way that benefits the customer. Banks must ensure that they are not pressuring customers into purchasing products they do not need or want. It is also important to provide clear information about the products being offered and their associated costs and risks.

1.2 Significance of Lead Prediction

Lead prediction in cross-selling is the process of identifying which existing customers are most likely to purchase additional products or services from the bank. Lead prediction is significant in cross-selling because it enables banks to focus their cross-selling efforts on customers who are most likely to be receptive to additional offers. This, in turn, increases the chances of success in cross-selling efforts and helps banks to achieve their revenue and profitability goals.

By using predictive analytics tools and techniques, banks can analyze customer data such as transaction history, demographics, and other behavioral patterns to identify customers who are most likely to be interested in specific products or services. For example, a bank may use predictive analytics to identify customers who have recently made large purchases or have shown an interest in investments, and then target those customers with offers for credit cards or investment products.

Lead prediction also helps banks to avoid wasting resources by targeting customers who are unlikely to be interested in additional products or services. By focusing on customers who are most likely to be receptive to cross-selling efforts, banks can avoid annoying or alienating customers with irrelevant or unwanted offers.

In summary, lead prediction is significant in cross-selling because it helps banks to:

- Focus their cross-selling efforts on customers who are most likely to be receptive to additional offers.
- Increase the chances of success in cross-selling efforts.
- Achieve revenue and profitability goals.
- Avoid wasting resources by targeting customers who are unlikely to be interested in additional products or services.

1.3 Classification Using Machine Learning

Classification is a type of machine learning algorithm that involves the categorization of input data into a pre-defined set of classes or categories. The goal of classification is to predict the class or category to which new data belongs, based on patterns identified in the training data.

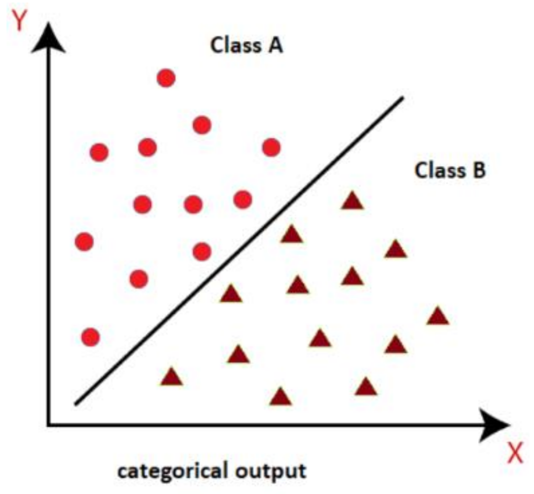


Diagram 1.3.a: Classification Process

Figure 1.3.a: Classification Process[6]

Classification algorithms are used in a wide range of applications, including spam detection, image recognition, sentiment analysis, and medical diagnosis. These algorithms learn from labeled data, which is data that has already been categorized into different classes or categories.

The process of classification involves several steps:

- **Data Preparation:** The input data is pre-processed and formatted into a suitable form for analysis.
- **Feature Extraction:** The relevant features or attributes of the data are identified and extracted.
- **Training:** The algorithm is trained on a labeled dataset, where each data point is associated with a pre-defined class or category.
- **Validation:** The algorithm is tested on a separate validation dataset to measure its performance and identify any issues.
- **Prediction:** Once the algorithm has been trained and validated, it can be used to predict the class or category of new data.

There are many different classification algorithms, including:

- **Decision trees:** These algorithms create a tree-like model of decisions and their possible consequences.
- **Naive Bayes:** These algorithms are based on Bayesian probability theory and assume that all features are independent of each other.
- **Support vector machines:** These algorithms find the optimal hyperplane that separates the classes in the data.
- **Logistic regression:** These algorithms model the relationship between the input features and the probability of belonging to a particular class.

Overall, classification is a powerful tool for automatically categorizing data and making predictions based on patterns in the data.

1.4 Structure of Thesis

This section briefly summarizes the rest of the chapters and outlines the dissertation.

Chapter-2: Literature Review
This chapter will review the previous work done in this area.

Chapter-3: The Data

This chapter will go through the data collection, exploration, outlier analysis, missing values and correlation analysis.

Chapter-4: Methodology

This chapter will explain all the methodologies and processes used in the thesis work.

Chapter-5: Model Experimentation

This chapter will explain all the model experimentation that was performed to find the optimal approach for the problem.

Chapter-2: Literature Review

Health insurance lead prediction using machine learning models is a rapidly growing field that has the potential to revolutionize the way health insurance companies identify and target potential customers. In this literature survey, we will explore the various machine learning models that have been used for health insurance lead prediction, the results achieved, and the challenges faced by researchers.

The project "Health Insurance Lead Prediction" by Miguel Santana[22] is a machine learning project aimed at predicting the likelihood of individuals to purchase health insurance. The project uses a dataset of customer information, including demographics, income, and health-related variables, to train and evaluate machine learning models for lead prediction.

The project involves data cleaning and preprocessing, exploratory data analysis, feature engineering, and model selection and evaluation. Several machine learning algorithms, including logistic regression, decision trees, random forests, and gradient boosting, are used to build predictive models. The performance of the models is evaluated using various metrics, including accuracy, precision, recall, and F1-score.

The project is implemented in Python using popular machine learning libraries such as scikit-learn, pandas, and numpy. The code and documentation for the project are available on the GitHub repository, along with the dataset used for the analysis.

The author had experimented with imbalanced and SMOTE balanced target variable distribution with multiple algorithms like Gradient Boosting, Random Forest, Support Vector Machine, Logistic Regression, etc., using Pycaret moeling. He was able to obtain an accuracy of 76% and ROC Score of 0.66 for XGBClassifier (Extreme Gradient Boosting) post hyperparameter tuning using Grid Search CV.

However, the precision, F1-score and recall for the positive response cases using the model were 0.56, 0.02 and 0.05 respectively. This could mean that the model, while having very good ability to identify leads who would respond negatively, had limited ability in identifying the leads who responded positively.

The project "Health Insurance Lead Prediction" by Basil Jose[23] is another similar machine learning project in the same direction with the same dataset.

In the project, the author used two models on the dataset. The first model that used XGBClassifier (Extreme Gradient Boosting) obtained an accuracy and ROC-AUC Score of 78.4% and 0.81 respectively. The second model that used CatBoost Classifier obtained an accuracy and ROC-AUC Score of 71.4% and 0.80 respectively.

The author seems to have performed extensive prior testing and effective feature engineering so as to have arrived at these good results. A standalone analysis of positive response cases was not available so as to verify the performance of the models on the smaller target class.

The project "Health Insurance Customer Response Prediction" by Gabriel Atkin[24] is another project on the same dataset.

In the project, the author used Logistic Regression to classify the target class. The results of the analysis show that the Logistic Regression Classifier performed with an accuracy of 67%.

At the same time, a standalone analysis of the positive respondents had a precision, recall and F1-score of 0.34, 0.4 and 0.37 respectively. This could indicate a sub-optimal performance in identifying the smaller target class of positive respondents.

The project "Health Insurance Lead Prediction - JOB-A-THON - AV" by Sparsh Gupta[1] is also another project with the same dataset.

In the project, the author made a comparative analysis of multiple models like kNN, Logistic Regression, Elastic Net, Ridge Classifier, CNN, XGBClassifier, etc. The results of the analysis show that the XGBClassifier Classifier performs the best among the models evaluated, with an accuracy of ~74%.

A standalone analysis of positive response cases was not available so as to verify the performance of the models on the smaller target class.

Overall, the projects provide a useful example of applying machine learning techniques to predict customer interest in buying health insurance, and the findings could be valuable as a starting point for enthusiasts to work on such projects.

Chapter-3: The Data

This chapter will discuss on the various aspects / observations related to the data.

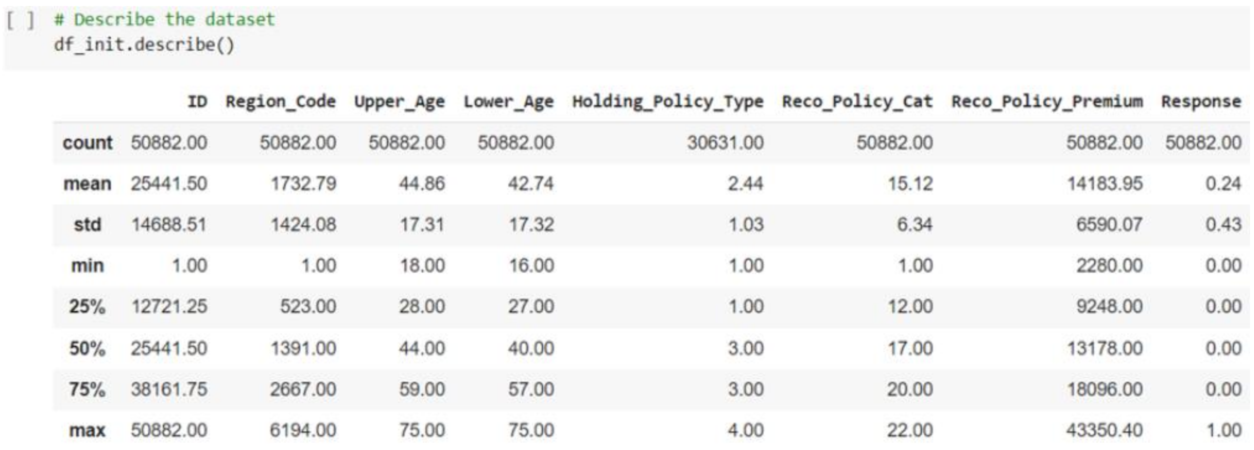
3.1 Data Collection

The data used for the project was downloaded from Kaggle[1].

There were two files – train.csv and test.csv. The former (with 50882 records) will be used for all the training, experimentation and testing. The latter will be used for prediction using the final model.

3.2 Data Exploration

The train dataset had 50882 records. Description is as below:



```
[ ] # Describe the dataset
    df_init.describe()
```

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
count	50882.00	50882.00	50882.00	50882.00	30631.00	50882.00	50882.00	50882.00
mean	25441.50	1732.79	44.86	42.74	2.44	15.12	14183.95	0.24
std	14688.51	1424.08	17.31	17.32	1.03	6.34	6590.07	0.43
min	1.00	1.00	18.00	16.00	1.00	1.00	2280.00	0.00
25%	12721.25	523.00	28.00	27.00	1.00	12.00	9248.00	0.00
50%	25441.50	1391.00	44.00	40.00	3.00	17.00	13178.00	0.00
75%	38161.75	2667.00	59.00	57.00	3.00	20.00	18096.00	0.00
max	50882.00	6194.00	75.00	75.00	4.00	22.00	43350.40	1.00

Diagram 3.2.a: Describing Training Data

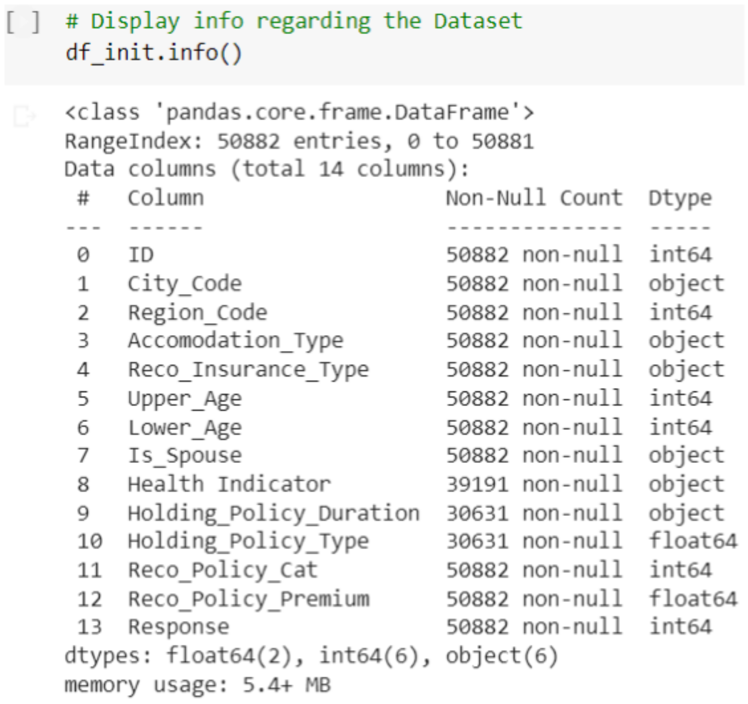
Figure 3.2.a: Describing Training Data

The raw dataset that we used have 14 columns. The column-level breakdown is as below:

- **ID** - System generated numerical value with no investigative significance. This can be removed as part of dimensionality reduction.
- **City_Code** - Values like C1, C2, etc. that indicates the city. These values are categorical in

nature and can be one-hot encoded.

- **Region_Code** - Numerical codes that indicate the region. These are numbers but they are categorical in nature. These too, can be one-hot encoded.
- **Accomodation_Type** - Type of accommodation (Rented/Owned/etc.). Categorical in nature and can be one-hot encoded.
- **Reco_Insurance_Type** - Recommended insurance type. Categorical and can be one-hot encoded.
- **Upper_Age** - Upper age among the applicants in the policy. Numerical value and can be scaled.
- **Lower_Age** - Lower age among the applicants in the policy. Numerical value and can be scaled.
- **Is_Spouse** - Is the applicant married? Categorical and can be one-hot encoded.
- **Health Indicator** - Health indicator code. Categorical and can be one-hot encoded.
- **Holding_Policy_Duration** - The duration for which the policy was held. This is an ordinal attribute and can be label encoded.
- **Holding_Policy_Type** - Type of policy held. Categorical and can be one-hot encoded.
- **Reco_Policy_Cat** - Recommended policy category. Categorical and can be one-hot encoded.
- **Reco_Policy_Premium** - Recommended premium amount. Numerical value and can be scaled.
- **Response** - Target variable that says Yes/No. A binary value.



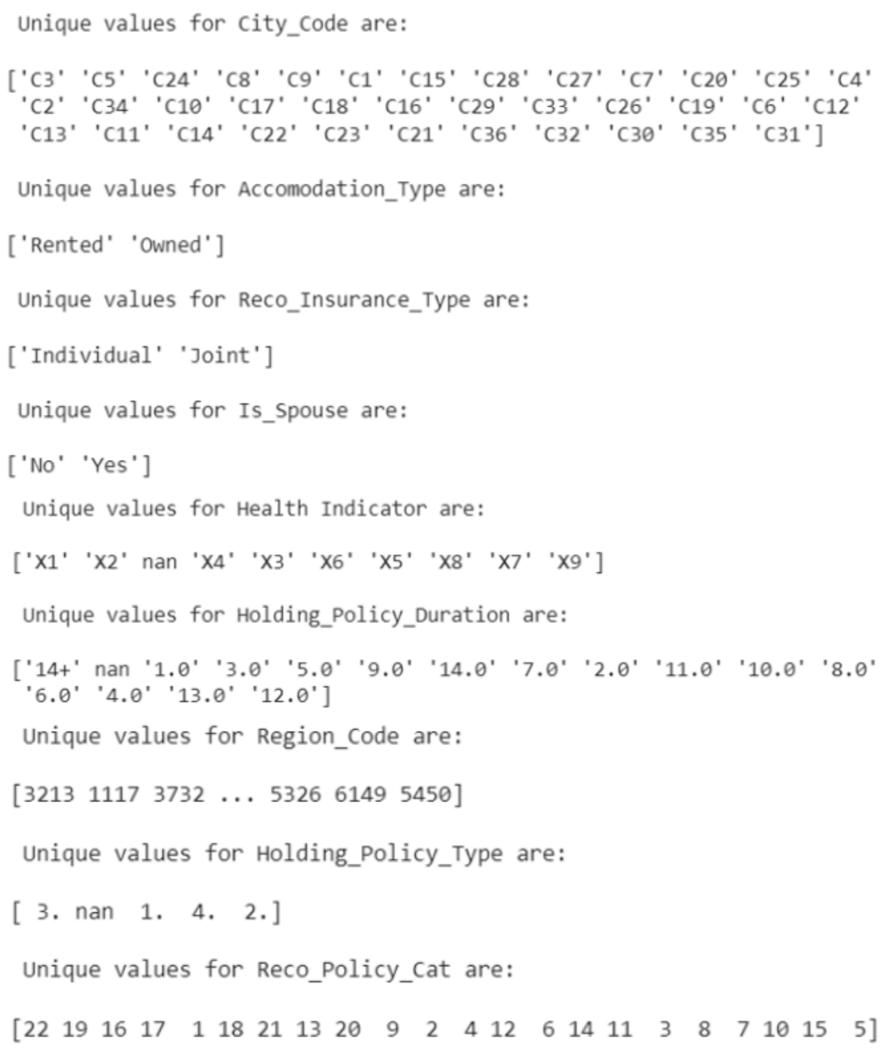
```
[ ] # Display info regarding the Dataset
    df_init.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50882 entries, 0 to 50881
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       50882 non-null  int64  
 1   City_Code                50882 non-null  object 
 2   Region_Code              50882 non-null  int64  
 3   Accomodation_Type        50882 non-null  object 
 4   Reco_Insurance_Type      50882 non-null  object 
 5   Upper_Age                50882 non-null  int64  
 6   Lower_Age                50882 non-null  int64  
 7   Is_Spouse                50882 non-null  object 
 8   Health Indicator         39191 non-null  object 
 9   Holding_Policy_Duration  30631 non-null  object 
 10  Holding_Policy_Type      30631 non-null  float64
 11  Reco_Policy_Cat          50882 non-null  int64  
 12  Reco_Policy_Premium      50882 non-null  float64
 13  Response                 50882 non-null  int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 5.4+ MB
```

Diagram 3.2.b: Information on Dataset

Figure 3.2.b: Information on Dataset

Unique Values in the relevant categorical-type columns are as below:



```
 Unique values for City_Code are:

['C3' 'C5' 'C24' 'C8' 'C9' 'C1' 'C15' 'C28' 'C27' 'C7' 'C20' 'C25' 'C4'
 'C2' 'C34' 'C10' 'C17' 'C18' 'C16' 'C29' 'C33' 'C26' 'C19' 'C6' 'C12'
 'C13' 'C11' 'C14' 'C22' 'C23' 'C21' 'C36' 'C32' 'C30' 'C35' 'C31']

 Unique values for Accomodation_Type are:

['Rented' 'Owned']

 Unique values for Reco_Insurance_Type are:

['Individual' 'Joint']

 Unique values for Is_Spouse are:

['No' 'Yes']
 Unique values for Health Indicator are:

['X1' 'X2' nan 'X4' 'X3' 'X6' 'X5' 'X8' 'X7' 'X9']

 Unique values for Holding_Policy_Duration are:

['14+' nan '1.0' '3.0' '5.0' '9.0' '14.0' '7.0' '2.0' '11.0' '10.0' '8.0'
 '6.0' '4.0' '13.0' '12.0']
 Unique values for Region_Code are:

[3213 1117 3732 ... 5326 6149 5450]

 Unique values for Holding_Policy_Type are:

[ 3. nan  1.  4.  2.]

 Unique values for Reco_Policy_Cat are:

[22 19 16 17  1 18 21 13 20  9  2  4 12  6 14 11  3  8  7 10 15  5]
```

Diagram 3.2.c: Unique Values of Categorical Variables

Figure 3.2.c: Unique Values of Categorical Variables

3.3 Data Visualization

Some of the visualizations for column-value distributions are as below:

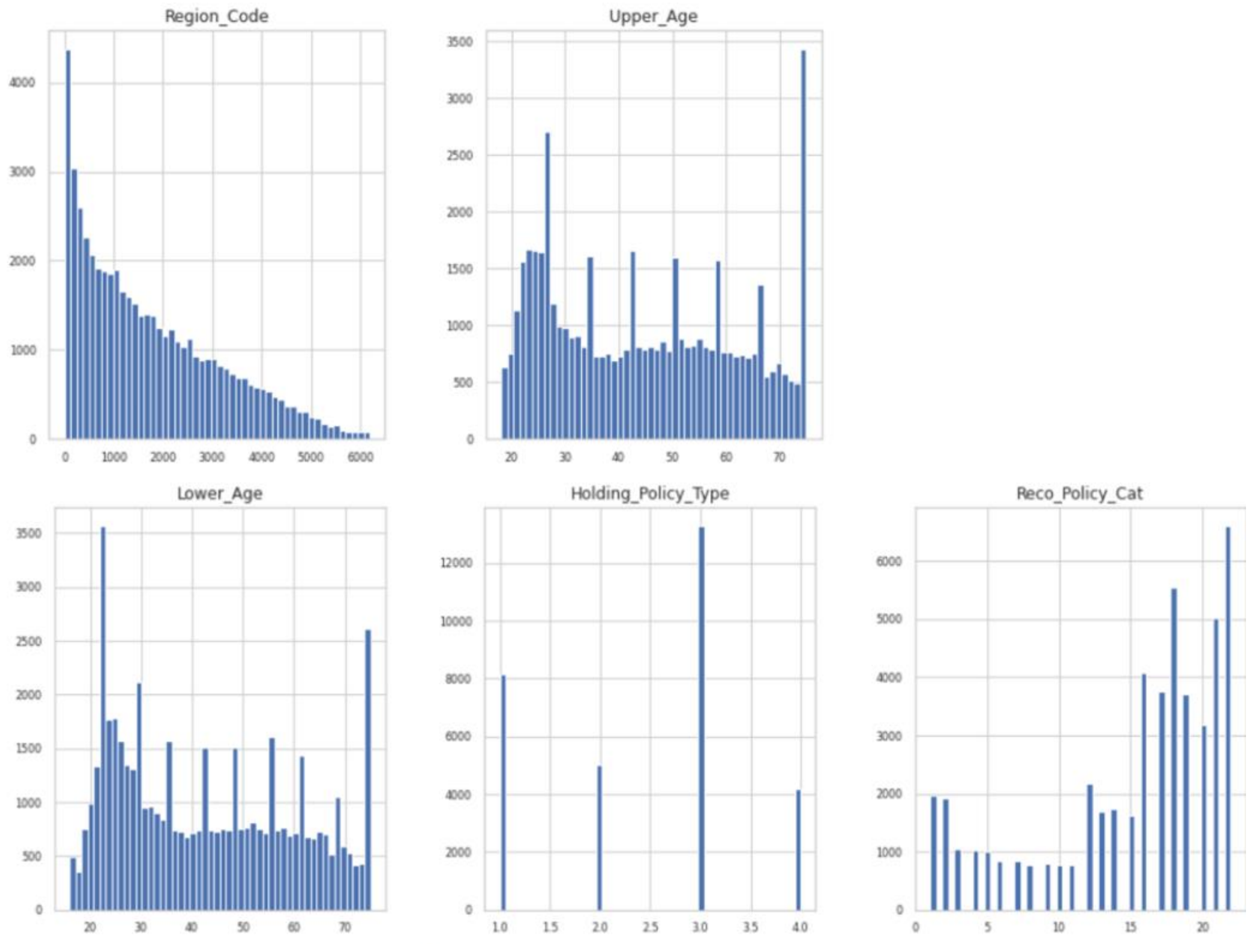


Diagram 3.3.a: Data Visualization Part 1

Figure 3.3.a: Data Visualization Part 1

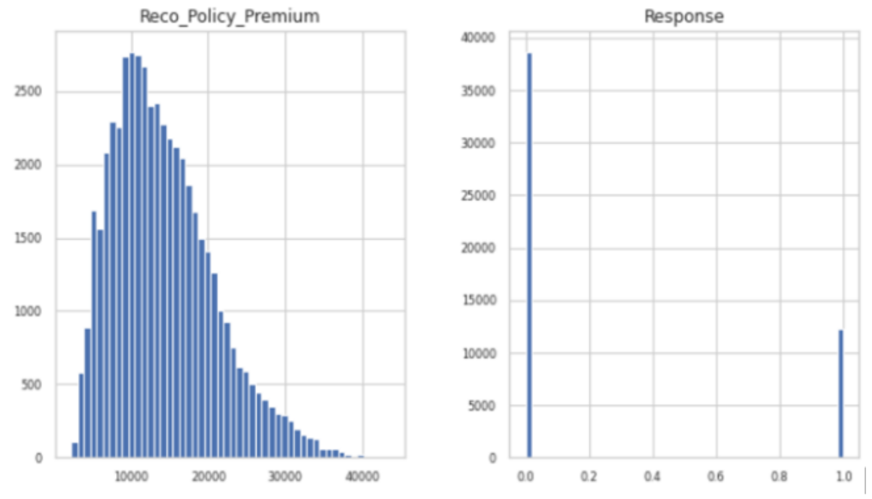


Diagram 3.3.b: Data Visualization Part 2

Figure 3.3.b: Data Visualization Part 2

Inferences are as below:

- Target variable (Response) distribution is imbalanced. This is expected because when cold-calling, the no. of people positively responding to a campaign will be much less than the no. of people who responds negatively to it.
- There are more records belonging to certain regions than others. The records across regions are not uniform.
- Age distributions are similar to each other.
- Recommended Policy Premium has a near-normal distribution with a left-skew.

3.4 Outlier Analysis

The outliers for the relevant continuous variables (Upper_Age, Lower_Age and Reco_Policy_Premium) were explored and investigated.

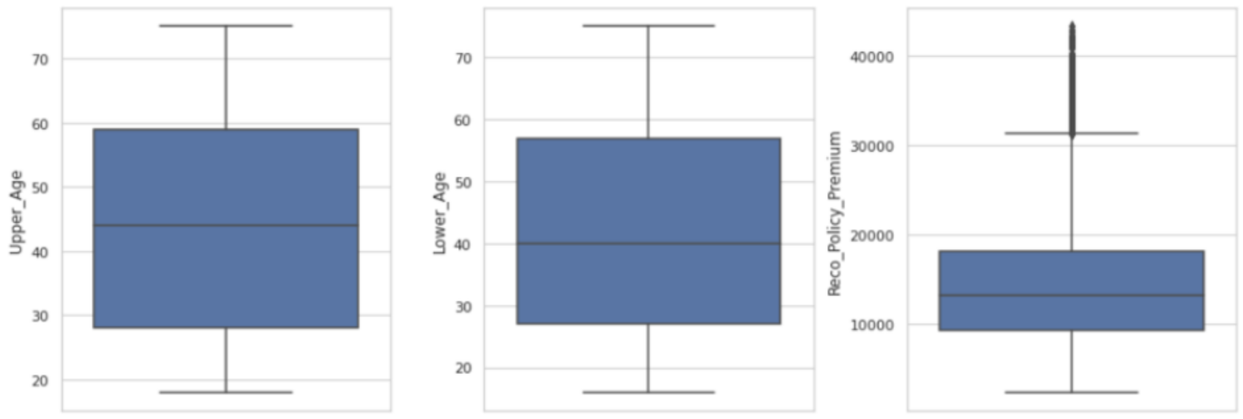


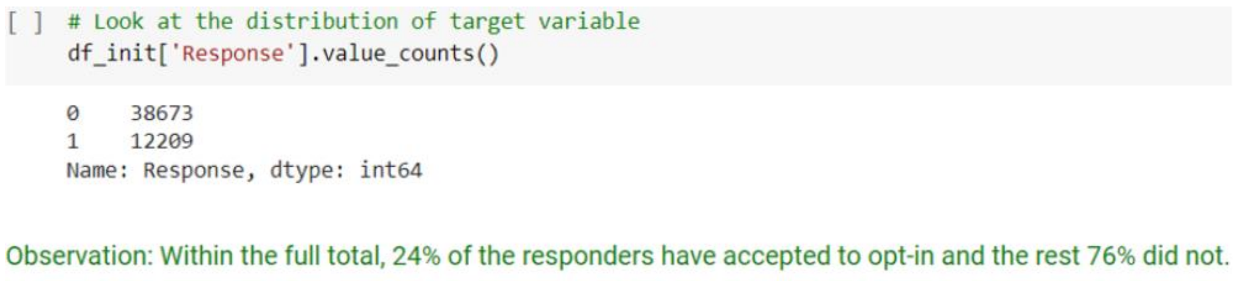
Diagram 3.4.a: Outlier Analysis

Figure 3.4.a: Outlier Analysis

The above is a boxplot visualization of the outlier analysis.

The results and inferences were as follows:

- Outliers were found only for Reco_Policy_Premium variable.
- The distribution of the target variable of the outliers were the same as the distribution of the target variable normal values. Strictly thinking from the perspective of this dataset, this indicates that the outliers do not have any additional prediction power.


```
[ ] # Look at the distribution of target variable
    df_init['Response'].value_counts()

    0    38673
    1    12209
    Name: Response, dtype: int64
```

Observation: Within the full total, 24% of the responders have accepted to opt-in and the rest 76% did not.

Diagram 3.4.b: Total Distribution of Target Variable

Table 3.4.b: Total Distribution of Target Variable

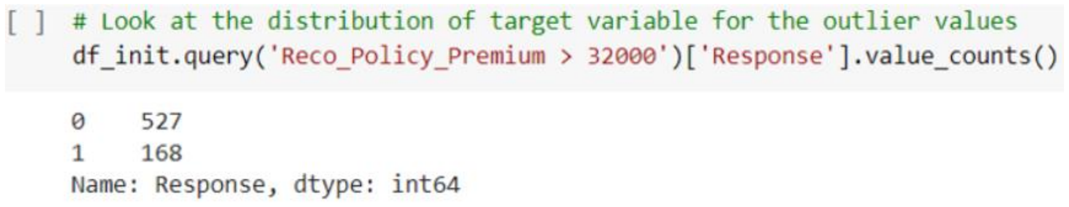

```
[ ] # Look at the distribution of target variable for the outlier values
    df_init.query('Reco_Policy_Premium > 32000')['Response'].value_counts()

    0    527
    1    168
    Name: Response, dtype: int64
```

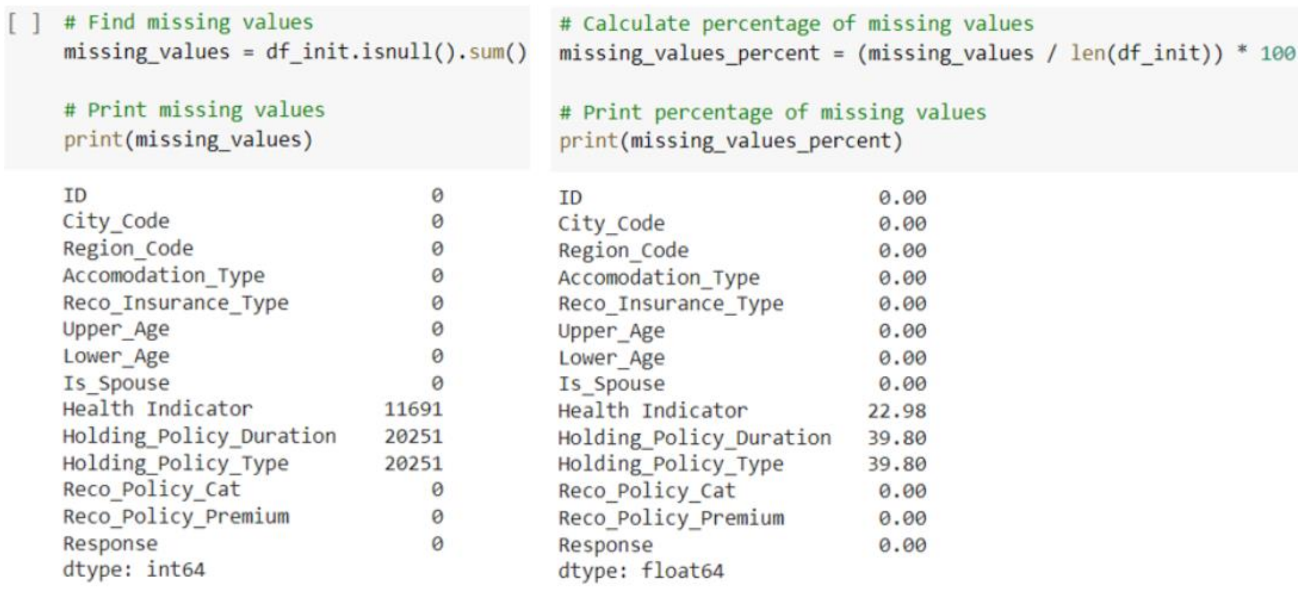
Diagram 3.4.b: Outlier Distribution of Target Variable

Table 3.4.c: Outlier Distribution of Target Variable

- Within the outliers, ~24% of the responders have accepted to opt-in and the rest ~76% did not. This is consistent with the total distribution. It is reasonable to assume that the outliers do not have a high correlation with the target variable.
- As seen in the previous section, the distribution for Reco_Policy_Premium variable mostly adheres to a (slightly left-skewed) Gaussian distribution.
- Thus, we can safely remove them using five-number summary.
- The rest of the values can be scaled for this column.

3.5 Missing Values

The state of missing values was investigated.



```
[ ] # Find missing values
    missing_values = df_init.isnull().sum()

    # Print missing values
    print(missing_values)

    ID                          0
    City_Code                   0
    Region_Code                 0
    Accomodation_Type           0
    Reco_Insurance_Type         0
    Upper_Age                   0
    Lower_Age                   0
    Is_Spouse                   0
    Health Indicator        11691
    Holding_Policy_Duration 20251
    Holding_Policy_Type     20251
    Reco_Policy_Cat             0
    Reco_Policy_Premium         0
    Response                    0
    dtype: int64
```

```
# Calculate percentage of missing values
missing_values_percent = (missing_values / len(df_init)) * 100

# Print percentage of missing values
print(missing_values_percent)

ID                          0.00
City_Code                   0.00
Region_Code                 0.00
Accomodation_Type           0.00
Reco_Insurance_Type         0.00
Upper_Age                   0.00
Lower_Age                   0.00
Is_Spouse                   0.00
Health Indicator           22.98
Holding_Policy_Duration    39.80
Holding_Policy_Type        39.80
Reco_Policy_Cat             0.00
Reco_Policy_Premium         0.00
Response                    0.00
dtype: float64
```

Diagram 3.5.a: Missing Values in Dataset

Table 3.5.a: Missing Values in Dataset

Below are the inferences:

- Three columns - Health Indicator, Holding_Policy_Duration and Holdin_Policy_Type have missing values. Their % of misses are 22.98%, 39.8% and 39.8% respectively.
- All of these are categorical values. It is not known what the absence of these values indicate. Thus, it might be worth checking if the absence of these values themselves is an indicator for target prediction.
- Thus, the decision is to perform one-hot encoding on the categorical variables and assume a bucket (non-entity) to indicate the absence.

3.6 Correlation Analysis

Correlation analysis was performed for the original dataset using Spearman's Correlation Coefficient.

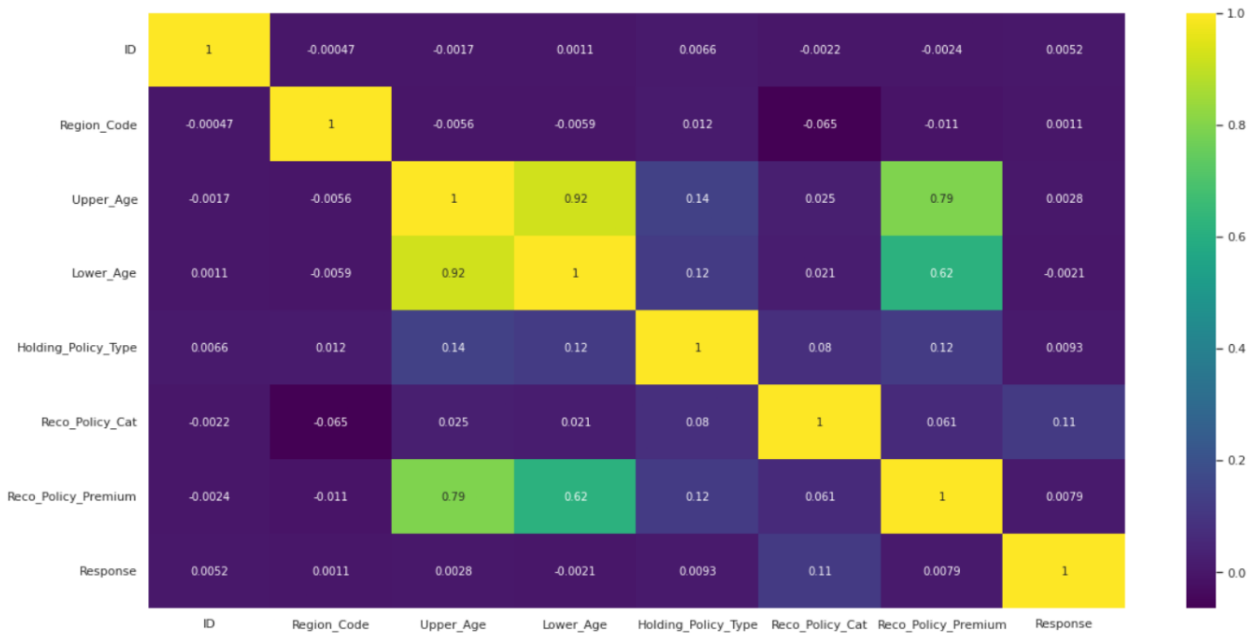


Diagram 3.6.a: Correlation Matrix of Dataset

Table 3.6.a: Correlation Matrix of Dataset

We do not see a good correlation for any of the predictor variables (numerical) with the target variable.

3.7 Data Preprocessing

Data preprocessing contained the below steps:

- Drop unwanted columns from the training set (ID).
- Remove Outliers using five-number summary.
- Continuous variable normalization using Min-Max Scaling.
- Feature Encoding of categorical variables using One-Hot Encoding and Label Encoding.

Chapter-4: Methodology

This chapter discusses the various processes and methodologies adopted in the thesis work.

4.1 Solution Architecture

The solution architecture (shown below) closely corresponds to the infrastructure in which the experiment is likely to be deployed.

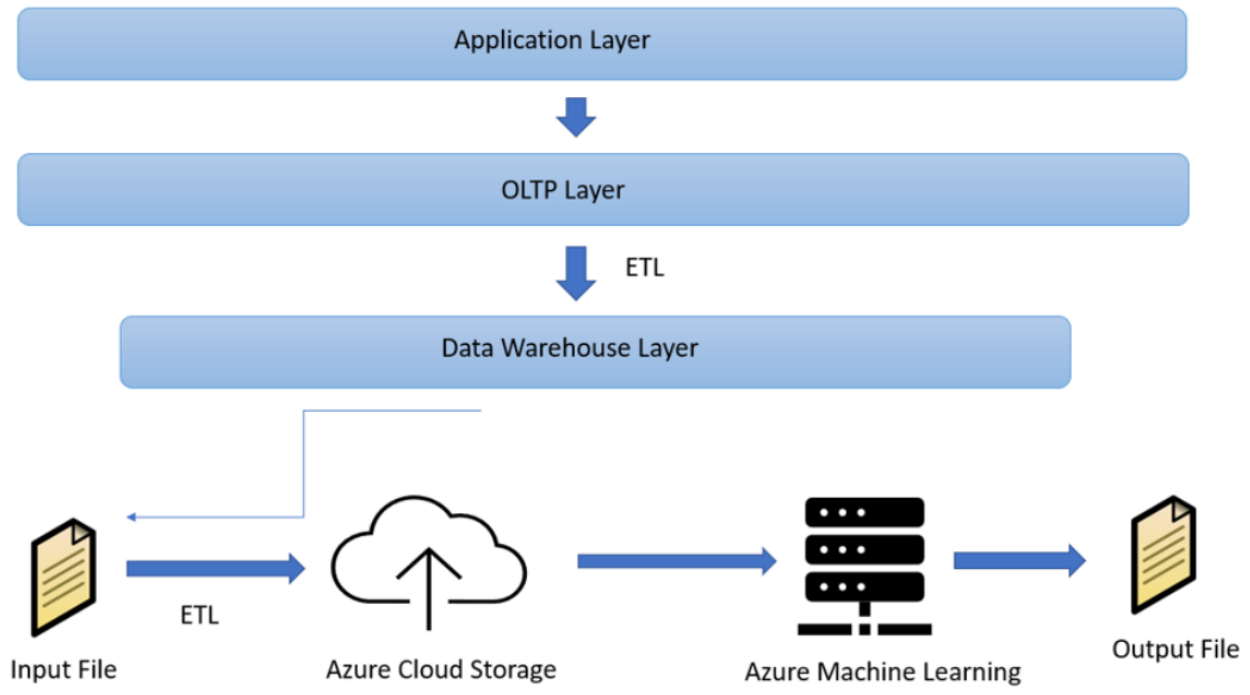


Diagram 4.1.a: Solution Architecture

Figure 4.1.a: Solution Architecture

The proposed solution is as follows:

- The input file for the ML process comes from the DWH layer.
- The input file will be uploaded to Azure Cloud Storage using ETL tools like Informatica / IICS.
- The ML pipeline will exist in the Azure Machine Learning service.
- The ML pipeline will consume the input file and generate the output file.

- Input file format will be the same as the file from Kaggle.
- Output format will, however, have only the Customer ID and Predicted Response.
- The file will be sent to the sales representatives in different areas / regions / cities so that they can streamline their efforts for cold-calling and further solicitation using the most viable leads first.

4.2 Process Diagram

The various steps followed in the thesis work is illustrated in the below diagram.

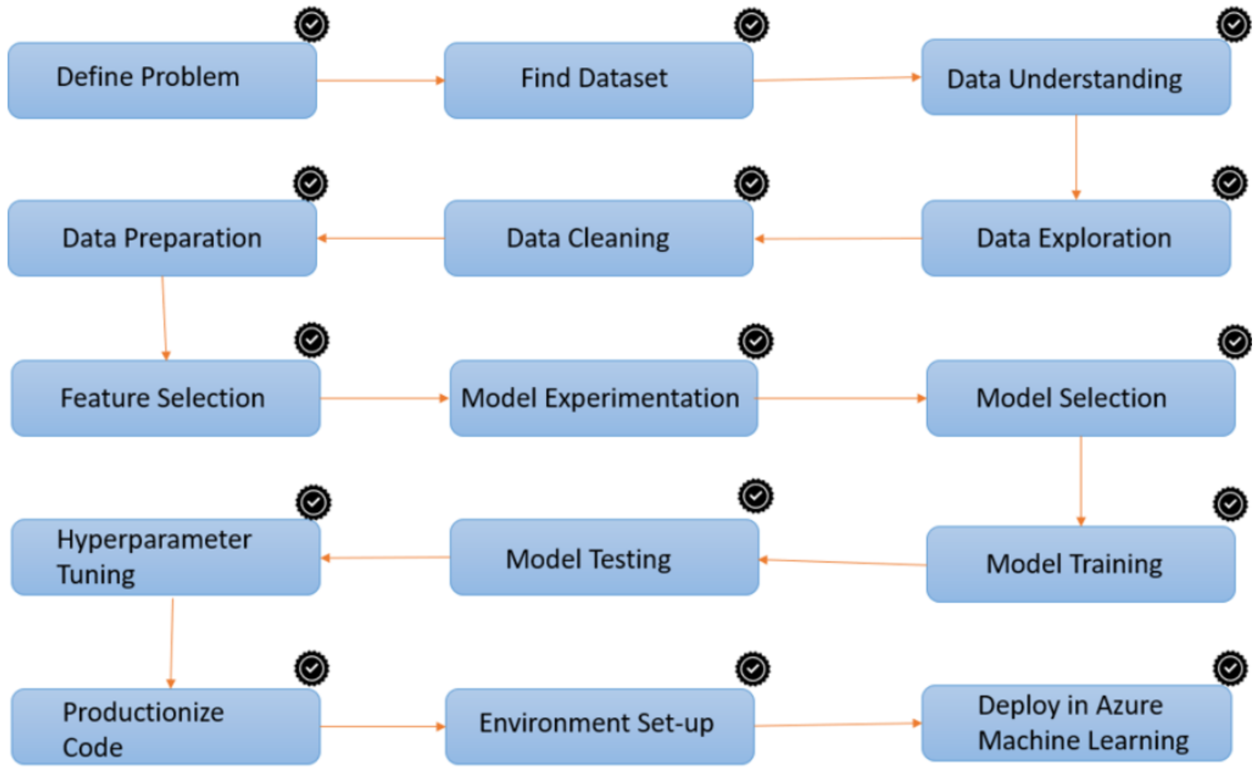


Diagram 4.2.a: Process Diagram

Figure 4.2.a: Process Diagram

The process descriptions are as follows:

- <u>**Define Problem:**</u> Define the scope and nature of the business problem that we are trying to solve.

- **Find Dataset:** Find a dataset that has characteristics similar to that of the business problem. The objective is to create an approach to solving the problem. We should be able to replicate the approach even with a dataset that has different columns. It is the thought-process and technical implementation that we are trying to focus on here.
- **Data Understanding:** Understand the data to find out what the different columns and their values mean.
- **Data Exploration:** Explore the statistical properties of the dataset.
- **Data Cleaning:** Remove or restructure the data to be used in production.
- **Data Preparation:** Apply Data preprocessing steps including Outlier Treatment, Normalization, Feature Encoding, etc.
- **Feature Selection:** Find out the features that have the highest prediction power.
- **Model Experimentation:** Perform experimentation with various models and conditions to find the model suitable for this particular dataset.
- **Model Selection:** Evaluate the experiments results and identify the optimal model for solving the problem with this particular dataset.
- **Model Training:** Train the identified model using the dataset.
- **Model Testing:** Test the efficacy of the selected model.
- **Hyperparameter Tuning:** Find the model parameters that gives the best results.
- **Productionize Code:** Modularize the code used and create a production deployable code.
- **Environment Set-up:** Simulate the production environment using Azure services.
- **Deploy in Azure Machine Learning:** Run the production code in Azure services and confirm that it is working.

4.3 Feature Selection

Feature encoding resulted in 5000+ columns. Feature selection was applied as mentioned in Appendix A (A.11).

Dimensionality reduction was one method performed to reduce the number of columns and still preserve as much information as possible. Principal Component Analysis (PCA) was used for this.

Filter Method was another method used for feature selection. These methods use a statistical measure to evaluate the relevance of each feature. The most commonly used measures are correlation coefficient and ANOVA F-value.

The SelectKBest class from sklearn.feature_selection can be used to select the top k features based on the chosen statistical measure. The statistical measure chosen was f_classif. f_classif is the ANOVA F-value between label/feature for classification tasks[3].

Wrapper methods like RFE were not used because of the cost of computation incurred upon running them on a dataset with 5000+ columns.

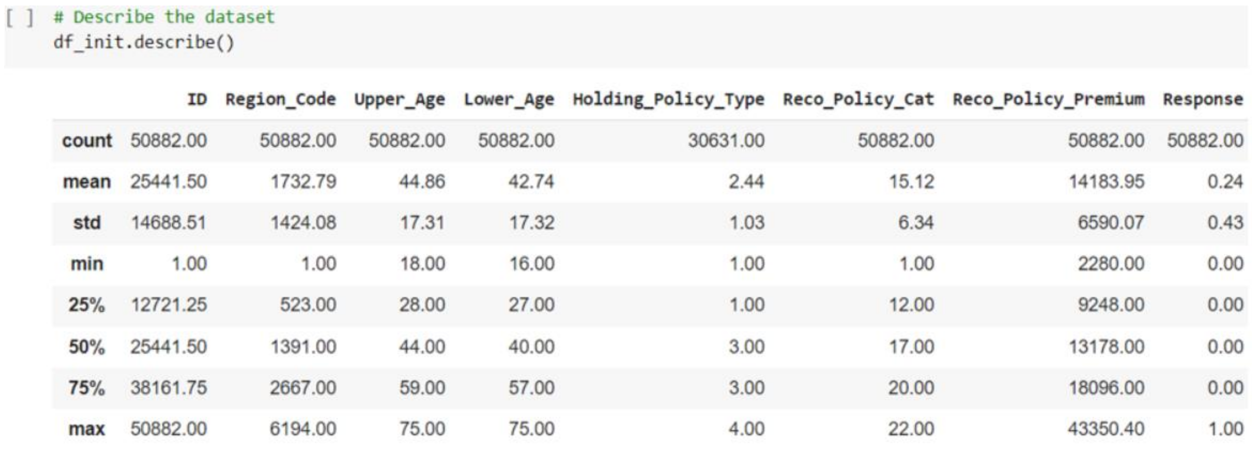
This particular section was experimented with in the below 3 ways:

- Used Filter method on the fully encoded and scaled dataset to find 100 / 50 best features out of 5000+ attributes.
- Used PCA to dwindle down the dimensionality from 5000+ to 10.
- Used PCA to dwindle down the dimensionality from 5000+ to 3.

After applying the above 3 operations on the fully encoded dataset and scaled dataset, the lazypredict method was used to find how the various classification algorithms would fare with it.

4.4 Outlier Treatment

Outlier treatment was performed in-line with the process mentioned in Appendix A (A.1) and for the variable Reco_Policy_Premium.

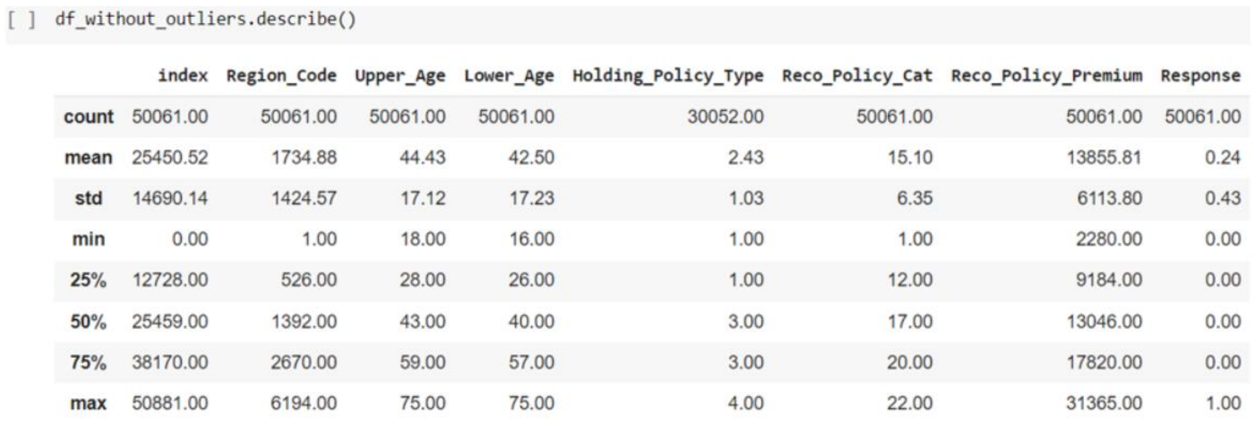


```
[ ] # Describe the dataset
    df_init.describe()
```

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
count	50882.00	50882.00	50882.00	50882.00	30631.00	50882.00	50882.00	50882.00
mean	25441.50	1732.79	44.86	42.74	2.44	15.12	14183.95	0.24
std	14688.51	1424.08	17.31	17.32	1.03	6.34	6590.07	0.43
min	1.00	1.00	18.00	16.00	1.00	1.00	2280.00	0.00
25%	12721.25	523.00	28.00	27.00	1.00	12.00	9248.00	0.00
50%	25441.50	1391.00	44.00	40.00	3.00	17.00	13178.00	0.00
75%	38161.75	2667.00	59.00	57.00	3.00	20.00	18096.00	0.00
max	50882.00	6194.00	75.00	75.00	4.00	22.00	43350.40	1.00

Diagram 4.4.a: Dataset Description WITH Outliers

Table 4.4.a: Dataset Description WITH Outliers


```
[ ] df_without_outliers.describe()
```

	index	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
count	50061.00	50061.00	50061.00	50061.00	30052.00	50061.00	50061.00	50061.00
mean	25450.52	1734.88	44.43	42.50	2.43	15.10	13855.81	0.24
std	14690.14	1424.57	17.12	17.23	1.03	6.35	6113.80	0.43
min	0.00	1.00	18.00	16.00	1.00	1.00	2280.00	0.00
25%	12728.00	526.00	28.00	26.00	1.00	12.00	9184.00	0.00
50%	25459.00	1392.00	43.00	40.00	3.00	17.00	13046.00	0.00
75%	38170.00	2670.00	59.00	57.00	3.00	20.00	17820.00	0.00
max	50881.00	6194.00	75.00	75.00	4.00	22.00	31365.00	1.00

Diagram 4.4.b: Dataset Description WITHOUT Outliers

Table 4.4.b: Dataset Description WITHOUT Outliers

4.5 Standardization

Min-Max Scaling was applied as mentioned in Appendix A (A.2).

The package MinMaxScaler from sklearn library was used for the same[4]. Min and Max values were set as 0 and 1 respectively (default).

Chapter-5: Model Experimentation

5.1 Model Experimentation Plan

To find how different models will perform under different variations of the input, a plan for model experimentation was formulated.

5.1.1 Using Lazy Predict

Lazy Predict[5] helps to build a lot of basic models without much code and helps understand which models works better without any parameter tuning.

The model experimentation plan using Lazy Predict is as given below.

Case 1: Model Experimentation with Original Dataset

Run the Lazy Predict package on the preprocessed dataset. There are no column additions or encoding applied here.

Case 2: Model Experimentation After Feature Selection

Run the Lazy Predict package on the preprocessed and fully encoded dataset after applying feature selection.

Case 3a: Model Experimentation After Dimensionality Reduction (PCA=10)

Run the Lazy Predict package on the preprocessed and fully encoded dataset after applying dimensionality reduction using Principal Component axes value as 10.

Case 3b: Model Experimentation After Dimensionality Reduction (PCA=3)

Run the Lazy Predict package on the preprocessed and fully encoded dataset after applying dimensionality reduction using Principal Component axes value as 3.

5.1.2 Using Deep Learning

Convolutional Neural Network (CNN) model was used on the similar kind of dataset variations on which we ran the Lazy Predict operation.

The model experimentation plan using CNN is as below.

Case 4a: Model Experimentation with Deep Learning with PCA=10

Run the CNN model on the preprocessed and fully encoded dataset after applying dimensionality reduction using Principal Component axes value as 10.

Case 4b: Model Experimentation with Deep Learning with PCA=3

Run the CNN model on the preprocessed and fully encoded dataset after applying dimensionality reduction using Principal Component axes value as 3.

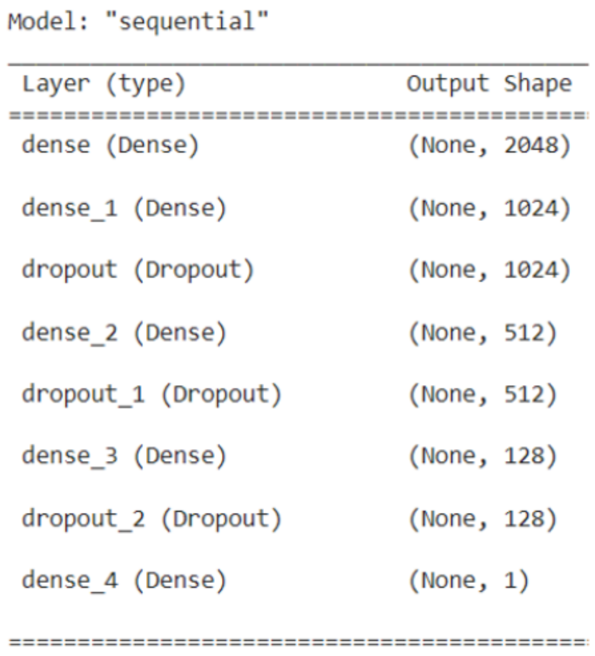
Case 4c: Model Experimentation with Deep Learning with Selected Features

Run the CNN model on the preprocessed and fully encoded dataset after applying feature selection.

Case 4d: Model Experimentation with Deep Learning with Encoded Dataset

Run the CNN model on the preprocessed and fully encoded dataset without applying any feature selection.

5.1.3 CNN Model Architecture



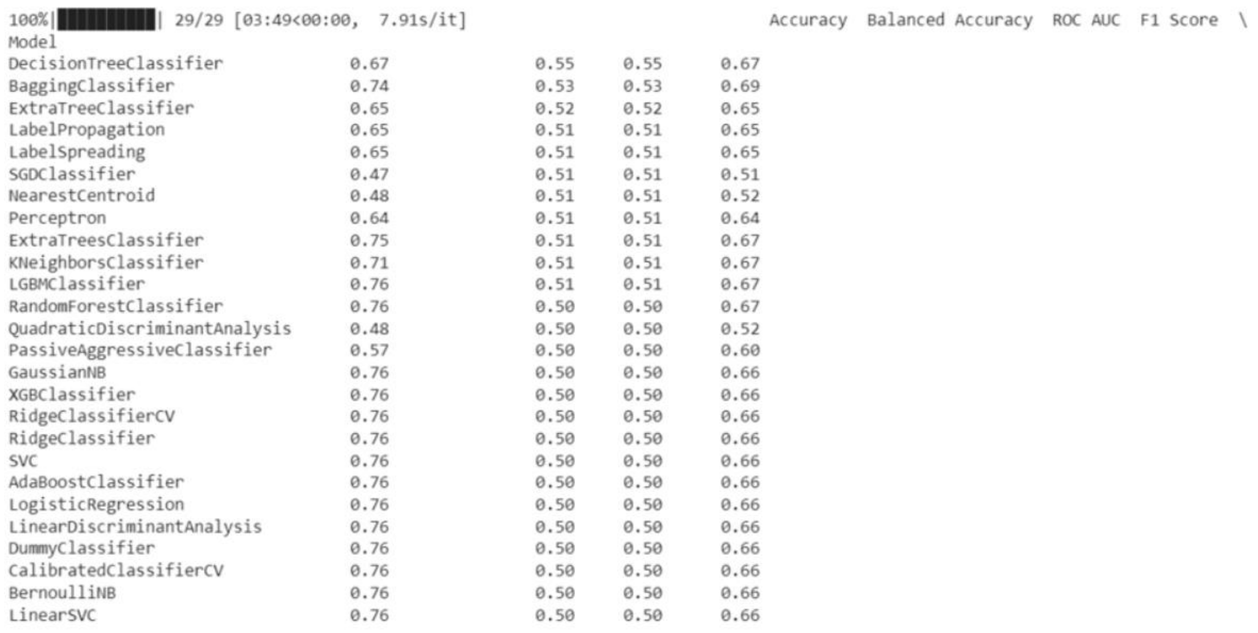
```
Model: "sequential"
_________________________________________________
 Layer (type)                 Output Shape
=================================================
 dense (Dense)                (None, 2048)

 dense_1 (Dense)              (None, 1024)

 dropout (Dropout)            (None, 1024)

 dense_2 (Dense)              (None, 512)

 dropout_1 (Dropout)          (None, 512)

 dense_3 (Dense)              (None, 128)

 dropout_2 (Dropout)          (None, 128)

 dense_4 (Dense)              (None, 1)

=================================================
```

Diagram 5.3.1.a: CNN Model Architecture

Table 5.1.3.a: CNN Model Architecture

5.2 Using Original Dataset

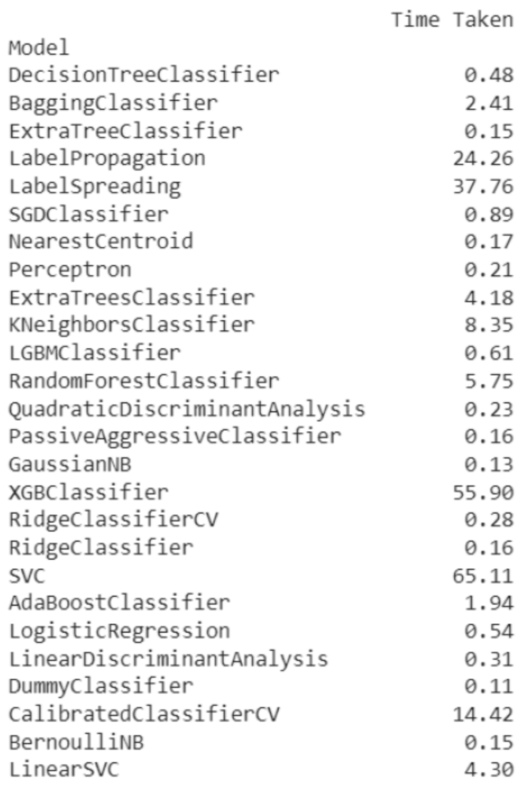
Below is the result of Case 1 experiment.



```
100%|██████████| 29/29 [03:49<00:00,  7.91s/it]                          Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
DecisionTreeClassifier              0.67              0.55     0.55      0.67
BaggingClassifier                   0.74              0.53     0.53      0.69
ExtraTreeClassifier                 0.65              0.52     0.52      0.65
LabelPropagation                    0.65              0.51     0.51      0.65
LabelSpreading                      0.65              0.51     0.51      0.65
SGDClassifier                       0.47              0.51     0.51      0.51
NearestCentroid                     0.48              0.51     0.51      0.52
Perceptron                          0.64              0.51     0.51      0.64
ExtraTreesClassifier                0.75              0.51     0.51      0.67
KNeighborsClassifier                0.71              0.51     0.51      0.67
LGBMClassifier                      0.76              0.51     0.51      0.67
RandomForestClassifier              0.76              0.50     0.50      0.67
QuadraticDiscriminantAnalysis       0.48              0.50     0.50      0.52
PassiveAggressiveClassifier         0.57              0.50     0.50      0.60
GaussianNB                          0.76              0.50     0.50      0.66
XGBClassifier                       0.76              0.50     0.50      0.66
RidgeClassifierCV                   0.76              0.50     0.50      0.66
RidgeClassifier                     0.76              0.50     0.50      0.66
SVC                                 0.76              0.50     0.50      0.66
AdaBoostClassifier                  0.76              0.50     0.50      0.66
LogisticRegression                  0.76              0.50     0.50      0.66
LinearDiscriminantAnalysis          0.76              0.50     0.50      0.66
DummyClassifier                     0.76              0.50     0.50      0.66
CalibratedClassifierCV              0.76              0.50     0.50      0.66
BernoulliNB                         0.76              0.50     0.50      0.66
LinearSVC                           0.76              0.50     0.50      0.66
```

Diagram 5.2.a: LazyPredict Evaluation Results for Case 1

Table 5.2.a: Lazy Predict Evaluation Results for Case 1

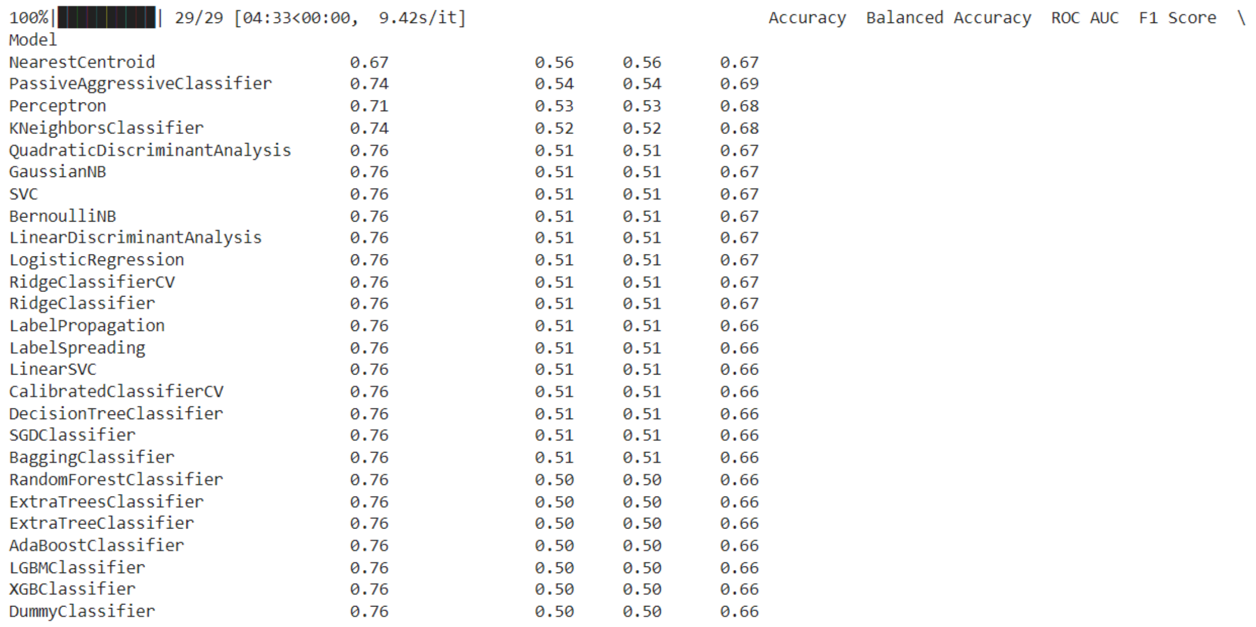


```
                               Time Taken
Model
DecisionTreeClassifier               0.48
BaggingClassifier                    2.41
ExtraTreeClassifier                  0.15
LabelPropagation                    24.26
LabelSpreading                      37.76
SGDClassifier                        0.89
NearestCentroid                      0.17
Perceptron                           0.21
ExtraTreesClassifier                 4.18
KNeighborsClassifier                 8.35
LGBMClassifier                       0.61
RandomForestClassifier               5.75
QuadraticDiscriminantAnalysis        0.23
PassiveAggressiveClassifier          0.16
GaussianNB                           0.13
XGBClassifier                       55.90
RidgeClassifierCV                    0.28
RidgeClassifier                      0.16
SVC                                 65.11
AdaBoostClassifier                   1.94
LogisticRegression                   0.54
LinearDiscriminantAnalysis           0.31
DummyClassifier                      0.11
CalibratedClassifierCV              14.42
BernoulliNB                          0.15
LinearSVC                            4.30
```

Table 5.2.b: Lazy Predict Runtimes for Case 1

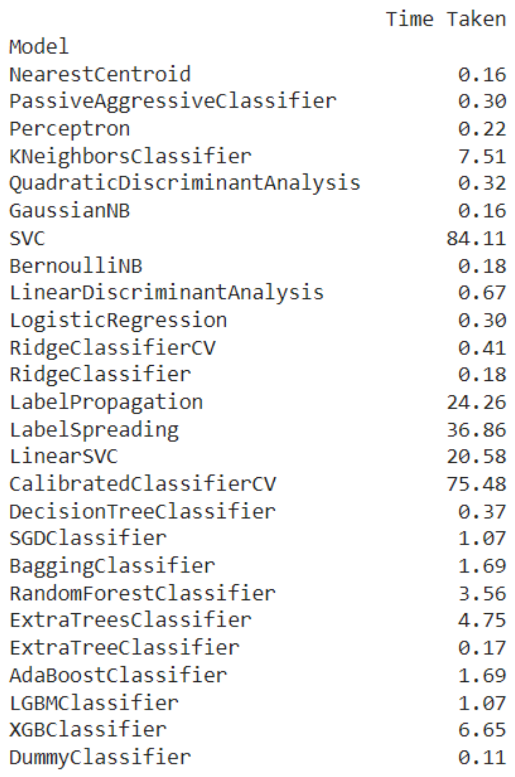
5.3 Using Feature Selection

Below is the result of Case 2 experiment.



```
100%|██████████| 29/29 [04:33<00:00,  9.42s/it]                              Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
NearestCentroid                    0.67               0.56     0.56      0.67
PassiveAggressiveClassifier        0.74               0.54     0.54      0.69
Perceptron                         0.71               0.53     0.53      0.68
KNeighborsClassifier               0.74               0.52     0.52      0.68
QuadraticDiscriminantAnalysis      0.76               0.51     0.51      0.67
GaussianNB                         0.76               0.51     0.51      0.67
SVC                                0.76               0.51     0.51      0.67
BernoulliNB                        0.76               0.51     0.51      0.67
LinearDiscriminantAnalysis         0.76               0.51     0.51      0.67
LogisticRegression                 0.76               0.51     0.51      0.67
RidgeClassifierCV                  0.76               0.51     0.51      0.67
RidgeClassifier                    0.76               0.51     0.51      0.67
LabelPropagation                   0.76               0.51     0.51      0.66
LabelSpreading                     0.76               0.51     0.51      0.66
LinearSVC                          0.76               0.51     0.51      0.66
CalibratedClassifierCV             0.76               0.51     0.51      0.66
DecisionTreeClassifier             0.76               0.51     0.51      0.66
SGDClassifier                      0.76               0.51     0.51      0.66
BaggingClassifier                  0.76               0.51     0.51      0.66
RandomForestClassifier             0.76               0.50     0.50      0.66
ExtraTreesClassifier               0.76               0.50     0.50      0.66
ExtraTreeClassifier                0.76               0.50     0.50      0.66
AdaBoostClassifier                 0.76               0.50     0.50      0.66
LGBMClassifier                     0.76               0.50     0.50      0.66
XGBClassifier                      0.76               0.50     0.50      0.66
DummyClassifier                    0.76               0.50     0.50      0.66
```

Table 5.3.a: Lazy Predict Evaluation Results for Case 2



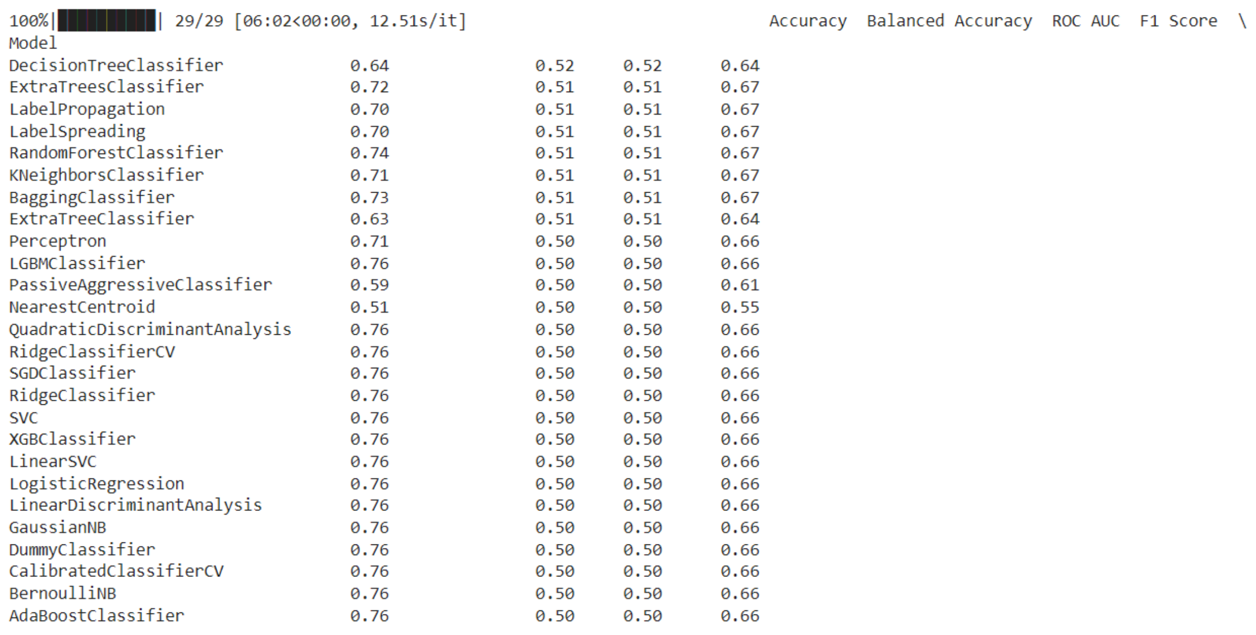
```
                               Time Taken
Model
NearestCentroid                      0.16
PassiveAggressiveClassifier          0.30
Perceptron                           0.22
KNeighborsClassifier                 7.51
QuadraticDiscriminantAnalysis        0.32
GaussianNB                           0.16
SVC                                 84.11
BernoulliNB                          0.18
LinearDiscriminantAnalysis           0.67
LogisticRegression                   0.30
RidgeClassifierCV                    0.41
RidgeClassifier                      0.18
LabelPropagation                    24.26
LabelSpreading                      36.86
LinearSVC                           20.58
CalibratedClassifierCV              75.48
DecisionTreeClassifier               0.37
SGDClassifier                        1.07
BaggingClassifier                    1.69
RandomForestClassifier               3.56
ExtraTreesClassifier                 4.75
ExtraTreeClassifier                  0.17
AdaBoostClassifier                   1.69
LGBMClassifier                       1.07
XGBClassifier                        6.65
DummyClassifier                      0.11
```

Table 5.3.b: Lazy Predict Runtime Results for Case 2

5.4 Using Dimensionality Reduction

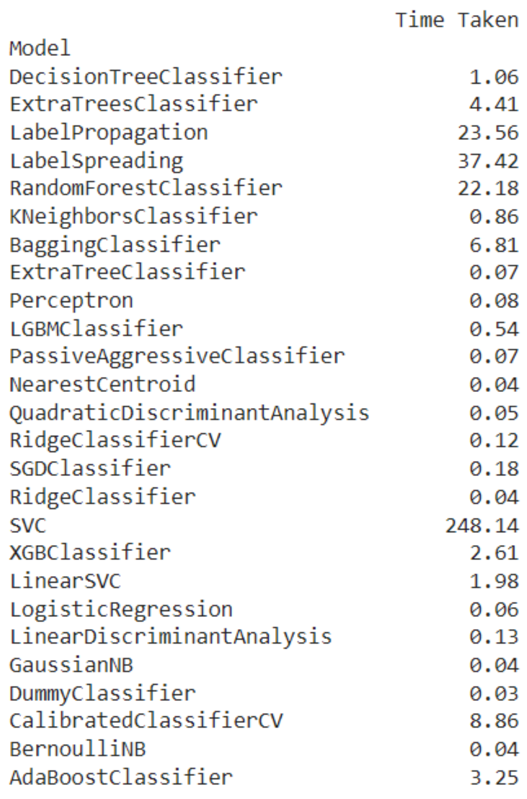
5.4.1 With PCA=10

Below is the result of Case 3a experiment.



```
100%|██████████| 29/29 [06:02<00:00, 12.51s/it]                    Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
DecisionTreeClassifier            0.64               0.52     0.52      0.64
ExtraTreesClassifier              0.72               0.51     0.51      0.67
LabelPropagation                  0.70               0.51     0.51      0.67
LabelSpreading                    0.70               0.51     0.51      0.67
RandomForestClassifier            0.74               0.51     0.51      0.67
KNeighborsClassifier              0.71               0.51     0.51      0.67
BaggingClassifier                 0.73               0.51     0.51      0.67
ExtraTreeClassifier               0.63               0.51     0.51      0.64
Perceptron                        0.71               0.50     0.50      0.66
LGBMClassifier                    0.76               0.50     0.50      0.66
PassiveAggressiveClassifier       0.59               0.50     0.50      0.61
NearestCentroid                   0.51               0.50     0.50      0.55
QuadraticDiscriminantAnalysis     0.76               0.50     0.50      0.66
RidgeClassifierCV                 0.76               0.50     0.50      0.66
SGDClassifier                     0.76               0.50     0.50      0.66
RidgeClassifier                   0.76               0.50     0.50      0.66
SVC                               0.76               0.50     0.50      0.66
XGBClassifier                     0.76               0.50     0.50      0.66
LinearSVC                         0.76               0.50     0.50      0.66
LogisticRegression                0.76               0.50     0.50      0.66
LinearDiscriminantAnalysis        0.76               0.50     0.50      0.66
GaussianNB                        0.76               0.50     0.50      0.66
DummyClassifier                   0.76               0.50     0.50      0.66
CalibratedClassifierCV            0.76               0.50     0.50      0.66
BernoulliNB                       0.76               0.50     0.50      0.66
AdaBoostClassifier                0.76               0.50     0.50      0.66
```

Table 5.4.1.a: Lazy Predict Evaluation Results for Case 3a

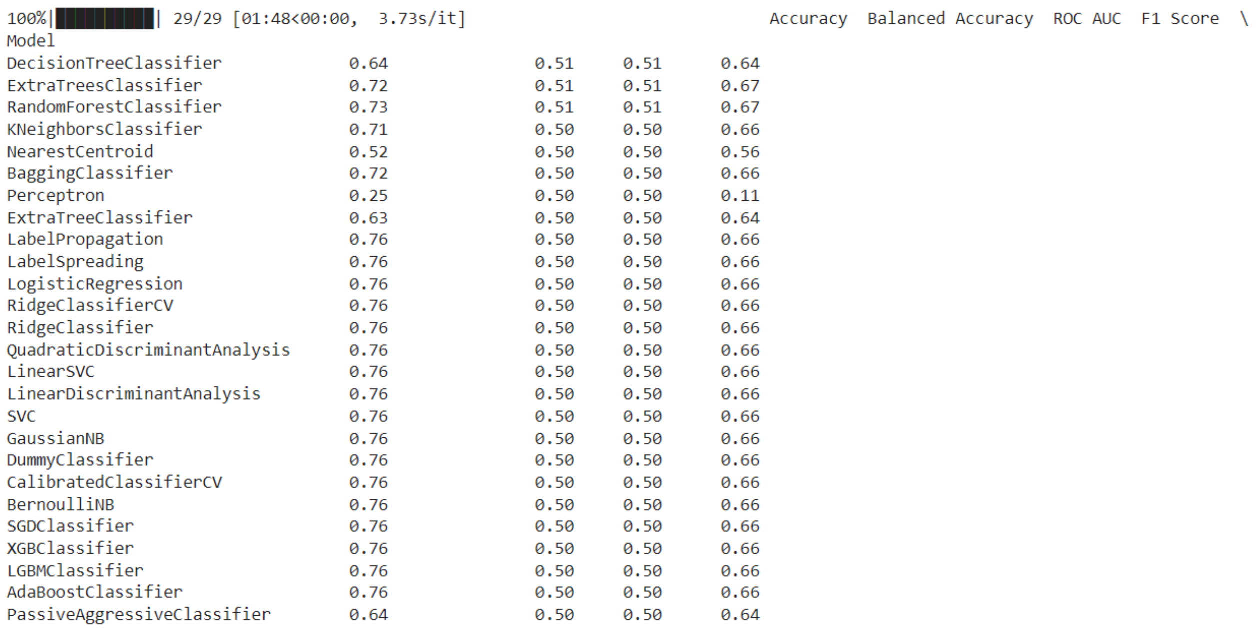


```
                               Time Taken
Model
DecisionTreeClassifier               1.06
ExtraTreesClassifier                 4.41
LabelPropagation                    23.56
LabelSpreading                      37.42
RandomForestClassifier              22.18
KNeighborsClassifier                 0.86
BaggingClassifier                    6.81
ExtraTreeClassifier                  0.07
Perceptron                           0.08
LGBMClassifier                       0.54
PassiveAggressiveClassifier          0.07
NearestCentroid                      0.04
QuadraticDiscriminantAnalysis        0.05
RidgeClassifierCV                    0.12
SGDClassifier                        0.18
RidgeClassifier                      0.04
SVC                                248.14
XGBClassifier                        2.61
LinearSVC                            1.98
LogisticRegression                   0.06
LinearDiscriminantAnalysis           0.13
GaussianNB                           0.04
DummyClassifier                      0.03
CalibratedClassifierCV               8.86
BernoulliNB                          0.04
AdaBoostClassifier                   3.25
```

Table 5.4.1.b: Lazy Predict Runtime Results for Case 3a

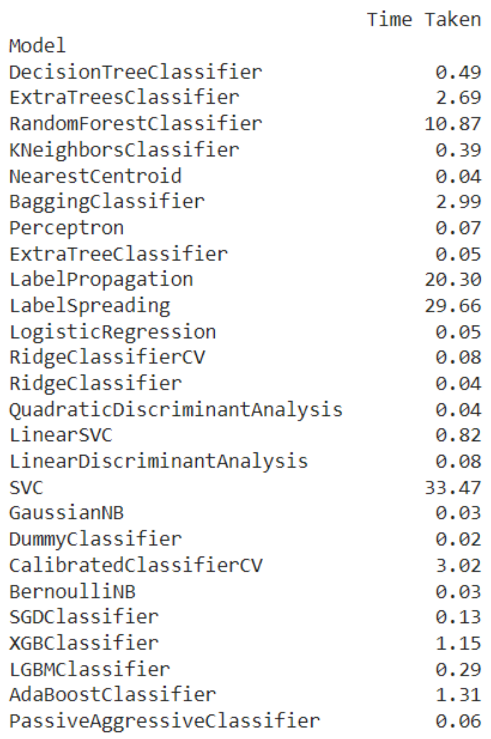
5.4.2 With PCA=3

Below is the result of Case 3b experiment.



```
100%|██████████| 29/29 [01:48<00:00,  3.73s/it]                              Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
DecisionTreeClassifier             0.64                0.51     0.51      0.64
ExtraTreesClassifier               0.72                0.51     0.51      0.67
RandomForestClassifier             0.73                0.51     0.51      0.67
KNeighborsClassifier               0.71                0.50     0.50      0.66
NearestCentroid                    0.52                0.50     0.50      0.56
BaggingClassifier                  0.72                0.50     0.50      0.66
Perceptron                         0.25                0.50     0.50      0.11
ExtraTreeClassifier                0.63                0.50     0.50      0.64
LabelPropagation                   0.76                0.50     0.50      0.66
LabelSpreading                     0.76                0.50     0.50      0.66
LogisticRegression                 0.76                0.50     0.50      0.66
RidgeClassifierCV                  0.76                0.50     0.50      0.66
RidgeClassifier                    0.76                0.50     0.50      0.66
QuadraticDiscriminantAnalysis      0.76                0.50     0.50      0.66
LinearSVC                          0.76                0.50     0.50      0.66
LinearDiscriminantAnalysis         0.76                0.50     0.50      0.66
SVC                                0.76                0.50     0.50      0.66
GaussianNB                         0.76                0.50     0.50      0.66
DummyClassifier                    0.76                0.50     0.50      0.66
CalibratedClassifierCV             0.76                0.50     0.50      0.66
BernoulliNB                        0.76                0.50     0.50      0.66
SGDClassifier                      0.76                0.50     0.50      0.66
XGBClassifier                      0.76                0.50     0.50      0.66
LGBMClassifier                     0.76                0.50     0.50      0.66
AdaBoostClassifier                 0.76                0.50     0.50      0.66
PassiveAggressiveClassifier        0.64                0.50     0.50      0.64
```

Table 5.4.2.a: Lazy Predict Evaluation Results for Case 3b

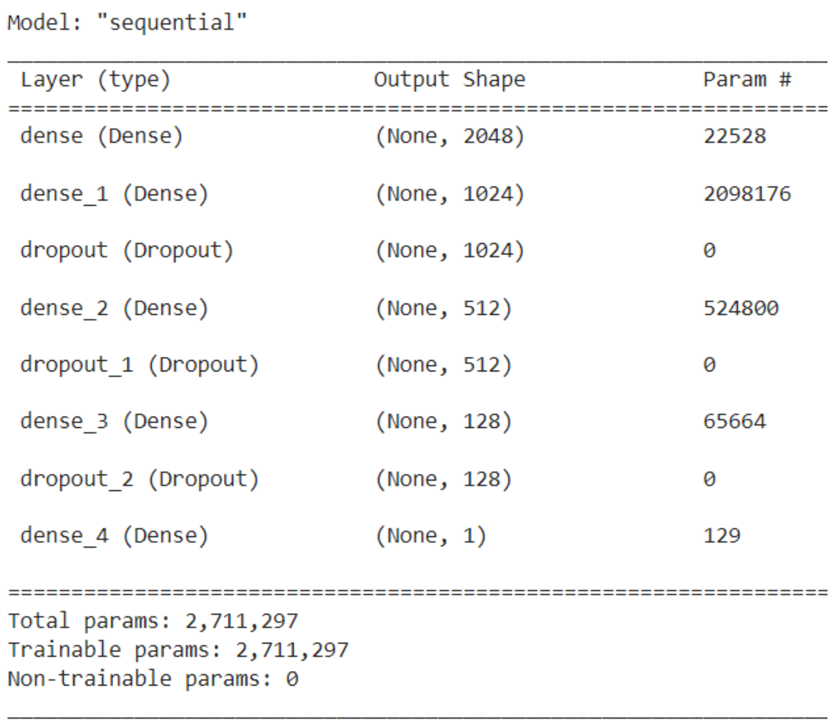


```
                                 Time Taken
Model
DecisionTreeClassifier                 0.49
ExtraTreesClassifier                   2.69
RandomForestClassifier                10.87
KNeighborsClassifier                   0.39
NearestCentroid                        0.04
BaggingClassifier                      2.99
Perceptron                             0.07
ExtraTreeClassifier                    0.05
LabelPropagation                      20.30
LabelSpreading                        29.66
LogisticRegression                     0.05
RidgeClassifierCV                      0.08
RidgeClassifier                        0.04
QuadraticDiscriminantAnalysis          0.04
LinearSVC                              0.82
LinearDiscriminantAnalysis             0.08
SVC                                   33.47
GaussianNB                             0.03
DummyClassifier                        0.02
CalibratedClassifierCV                 3.02
BernoulliNB                            0.03
SGDClassifier                          0.13
XGBClassifier                          1.15
LGBMClassifier                         0.29
AdaBoostClassifier                     1.31
PassiveAggressiveClassifier            0.06
```

Table 5.4.2.b: Lazy Predict Runtime Results for Case 3b

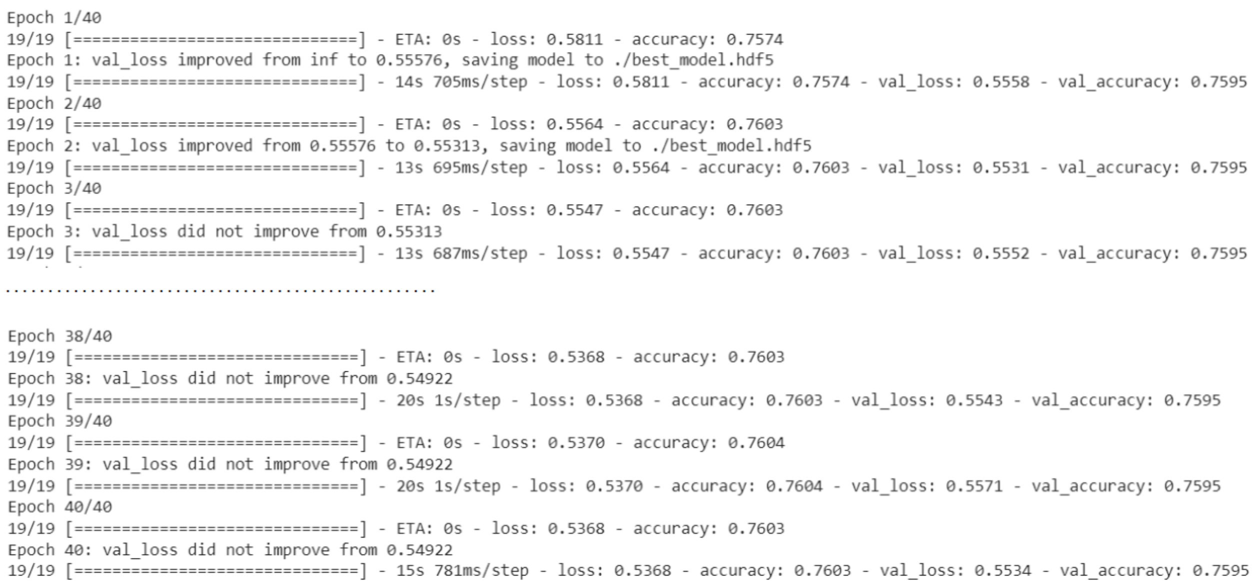
5.5 Using Deep Learning

5.5.1 Deep Learning with PCA=10



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2048)              22528

 dense_1 (Dense)             (None, 1024)              2098176

 dropout (Dropout)           (None, 1024)              0

 dense_2 (Dense)             (None, 512)               524800

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 128)               65664

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 1)                 129

=================================================================
Total params: 2,711,297
Trainable params: 2,711,297
Non-trainable params: 0
_________________________________________________________________
```

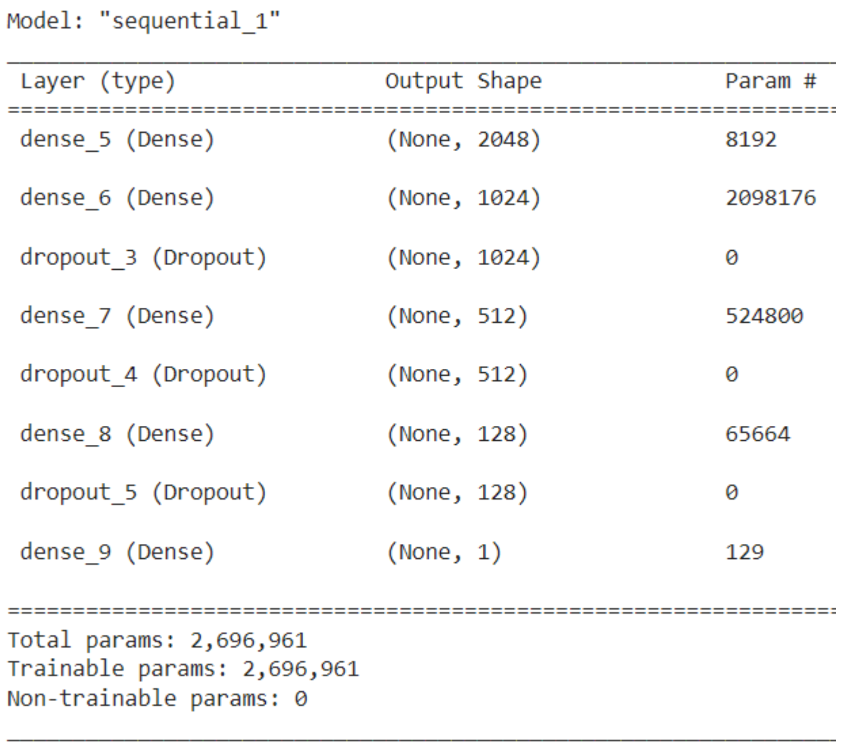
Table 5.5.1.a: CNN Model & Parameters for Case 4a



```
Epoch 1/40
19/19 [==============================] - ETA: 0s - loss: 0.5811 - accuracy: 0.7574
Epoch 1: val_loss improved from inf to 0.55576, saving model to ./best_model.hdf5
19/19 [==============================] - 14s 705ms/step - loss: 0.5811 - accuracy: 0.7574 - val_loss: 0.5558 - val_accuracy: 0.7595
Epoch 2/40
19/19 [==============================] - ETA: 0s - loss: 0.5564 - accuracy: 0.7603
Epoch 2: val_loss improved from 0.55576 to 0.55313, saving model to ./best_model.hdf5
19/19 [==============================] - 13s 695ms/step - loss: 0.5564 - accuracy: 0.7603 - val_loss: 0.5531 - val_accuracy: 0.7595
Epoch 3/40
19/19 [==============================] - ETA: 0s - loss: 0.5547 - accuracy: 0.7603
Epoch 3: val_loss did not improve from 0.55313
19/19 [==============================] - 13s 687ms/step - loss: 0.5547 - accuracy: 0.7603 - val_loss: 0.5552 - val_accuracy: 0.7595
..................................................

Epoch 38/40
19/19 [==============================] - ETA: 0s - loss: 0.5368 - accuracy: 0.7603
Epoch 38: val_loss did not improve from 0.54922
19/19 [==============================] - 20s 1s/step - loss: 0.5368 - accuracy: 0.7603 - val_loss: 0.5543 - val_accuracy: 0.7595
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.5370 - accuracy: 0.7604
Epoch 39: val_loss did not improve from 0.54922
19/19 [==============================] - 20s 1s/step - loss: 0.5370 - accuracy: 0.7604 - val_loss: 0.5571 - val_accuracy: 0.7595
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.5368 - accuracy: 0.7603
Epoch 40: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 781ms/step - loss: 0.5368 - accuracy: 0.7603 - val_loss: 0.5534 - val_accuracy: 0.7595
```

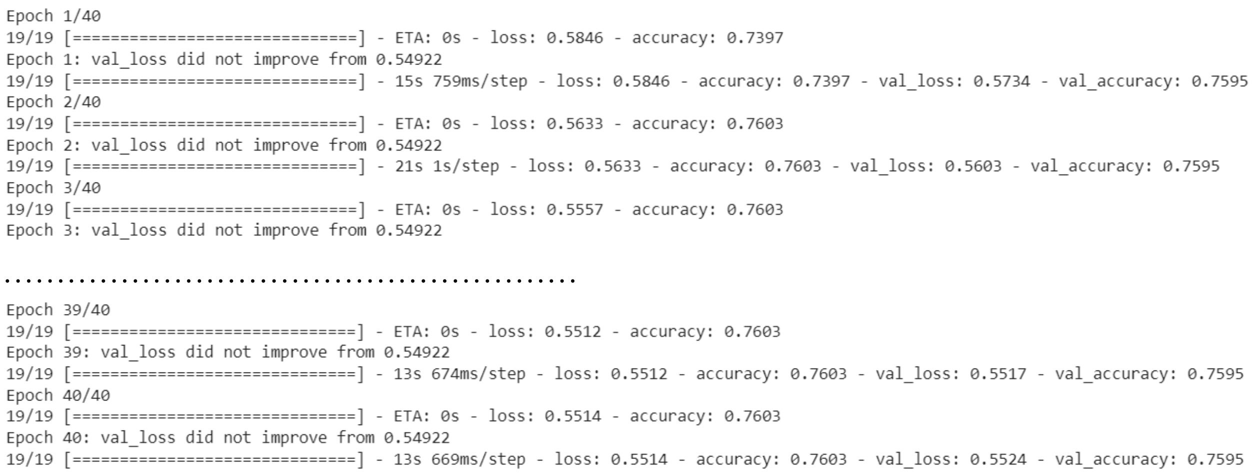
Table 5.5.1.b: CNN Model Training Epochs for Case 4a

5.5.2 Deep Learning with PCA=3



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 2048)              8192

 dense_6 (Dense)             (None, 1024)              2098176

 dropout_3 (Dropout)         (None, 1024)              0

 dense_7 (Dense)             (None, 512)               524800

 dropout_4 (Dropout)         (None, 512)               0

 dense_8 (Dense)             (None, 128)               65664

 dropout_5 (Dropout)         (None, 128)               0

 dense_9 (Dense)             (None, 1)                 129

=================================================================
Total params: 2,696,961
Trainable params: 2,696,961
Non-trainable params: 0
_________________________________________________________________
```

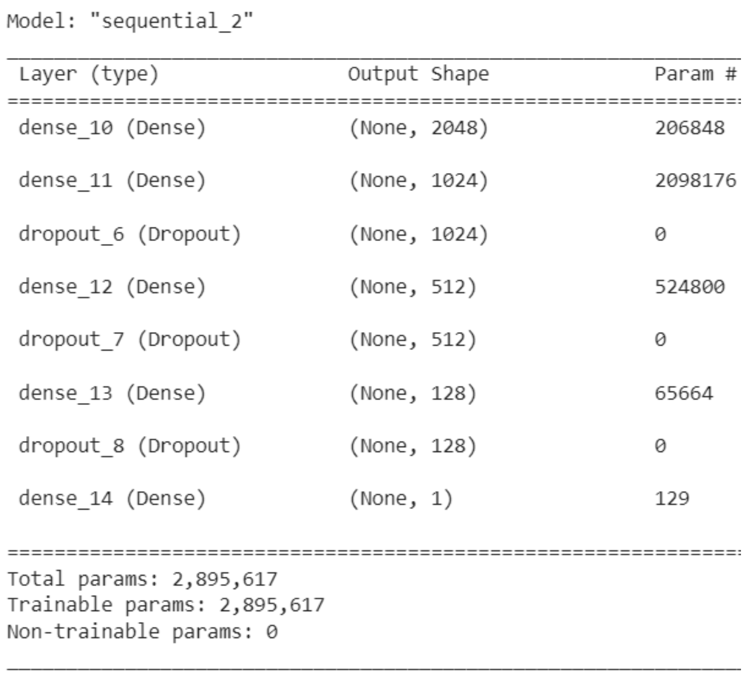
Table 5.5.2.a: CNN Model & Parameters for Case 4b



```
Epoch 1/40
19/19 [==============================] - ETA: 0s - loss: 0.5846 - accuracy: 0.7397
Epoch 1: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 759ms/step - loss: 0.5846 - accuracy: 0.7397 - val_loss: 0.5734 - val_accuracy: 0.7595
Epoch 2/40
19/19 [==============================] - ETA: 0s - loss: 0.5633 - accuracy: 0.7603
Epoch 2: val_loss did not improve from 0.54922
19/19 [==============================] - 21s 1s/step - loss: 0.5633 - accuracy: 0.7603 - val_loss: 0.5603 - val_accuracy: 0.7595
Epoch 3/40
19/19 [==============================] - ETA: 0s - loss: 0.5557 - accuracy: 0.7603
Epoch 3: val_loss did not improve from 0.54922

.....................................................

Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.5512 - accuracy: 0.7603
Epoch 39: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 674ms/step - loss: 0.5512 - accuracy: 0.7603 - val_loss: 0.5517 - val_accuracy: 0.7595
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.5514 - accuracy: 0.7603
Epoch 40: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 669ms/step - loss: 0.5514 - accuracy: 0.7603 - val_loss: 0.5524 - val_accuracy: 0.7595
```

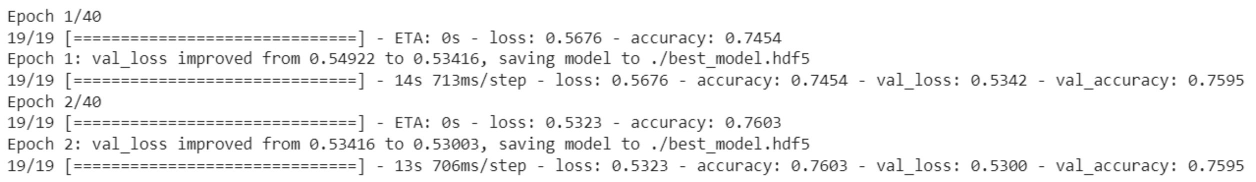
Table 5.5.2.b: CNN Model Training Epochs for Case 4b

5.5.3 Deep Learning with Feature Selection



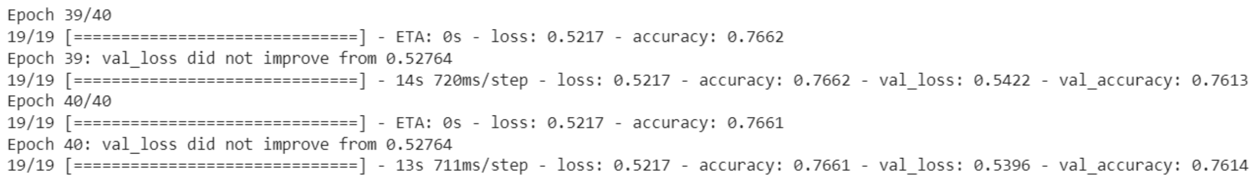
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 2048)              206848

 dense_11 (Dense)            (None, 1024)              2098176

 dropout_6 (Dropout)         (None, 1024)              0

 dense_12 (Dense)            (None, 512)               524800

 dropout_7 (Dropout)         (None, 512)               0

 dense_13 (Dense)            (None, 128)               65664

 dropout_8 (Dropout)         (None, 128)               0

 dense_14 (Dense)            (None, 1)                 129

=================================================================
Total params: 2,895,617
Trainable params: 2,895,617
Non-trainable params: 0
_________________________________________________________________
```

Table 5.5.3.a: CNN Model & Parameters for Case 4c



```
Epoch 1/40
19/19 [==============================] - ETA: 0s - loss: 0.5676 - accuracy: 0.7454
Epoch 1: val_loss improved from 0.54922 to 0.53416, saving model to ./best_model.hdf5
19/19 [==============================] - 14s 713ms/step - loss: 0.5676 - accuracy: 0.7454 - val_loss: 0.5342 - val_accuracy: 0.7595
Epoch 2/40
19/19 [==============================] - ETA: 0s - loss: 0.5323 - accuracy: 0.7603
Epoch 2: val_loss improved from 0.53416 to 0.53003, saving model to ./best_model.hdf5
19/19 [==============================] - 13s 706ms/step - loss: 0.5323 - accuracy: 0.7603 - val_loss: 0.5300 - val_accuracy: 0.7595
```

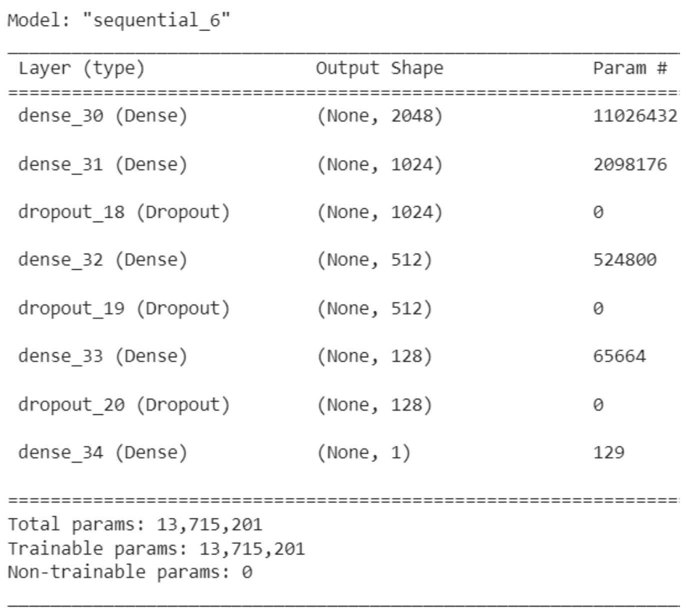
……………………………………………………..



```
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.5217 - accuracy: 0.7662
Epoch 39: val_loss did not improve from 0.52764
19/19 [==============================] - 14s 720ms/step - loss: 0.5217 - accuracy: 0.7662 - val_loss: 0.5422 - val_accuracy: 0.7613
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.5217 - accuracy: 0.7661
Epoch 40: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 711ms/step - loss: 0.5217 - accuracy: 0.7661 - val_loss: 0.5396 - val_accuracy: 0.7614
```

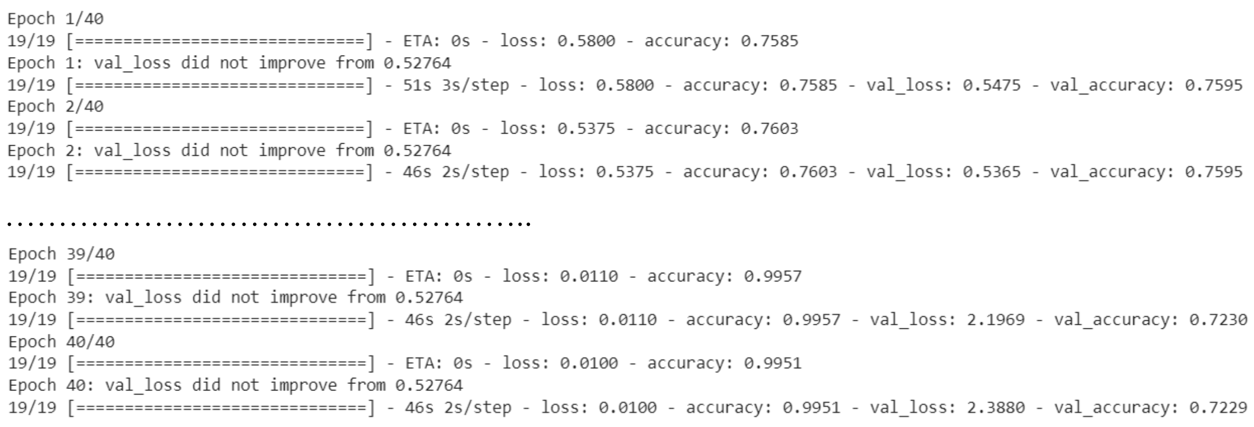
Table 5.5.3.b: CNN Model Training Epochs for Case 4c

5.5.4 Deep Learning with Feature Encoding



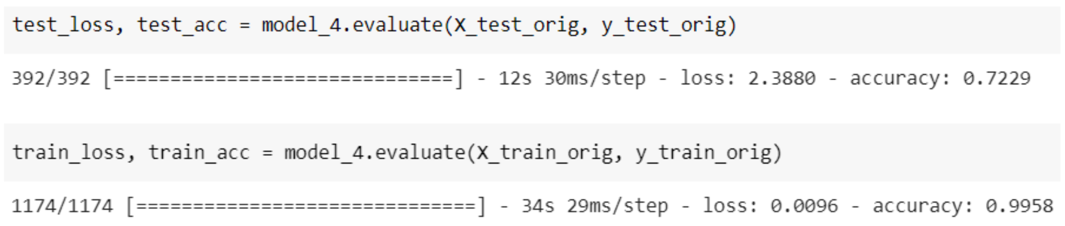
```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 2048)              11026432

 dense_31 (Dense)            (None, 1024)              2098176

 dropout_18 (Dropout)        (None, 1024)              0

 dense_32 (Dense)            (None, 512)               524800

 dropout_19 (Dropout)        (None, 512)               0

 dense_33 (Dense)            (None, 128)               65664

 dropout_20 (Dropout)        (None, 128)               0

 dense_34 (Dense)            (None, 1)                 129

=================================================================
Total params: 13,715,201
Trainable params: 13,715,201
Non-trainable params: 0
_________________________________________________________________
```

Table 5.5.4.a: CNN Model & Parameters for Case 4d



```
Epoch 1/40
19/19 [==============================] - ETA: 0s - loss: 0.5800 - accuracy: 0.7585
Epoch 1: val_loss did not improve from 0.52764
19/19 [==============================] - 51s 3s/step - loss: 0.5800 - accuracy: 0.7585 - val_loss: 0.5475 - val_accuracy: 0.7595
Epoch 2/40
19/19 [==============================] - ETA: 0s - loss: 0.5375 - accuracy: 0.7603
Epoch 2: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.5375 - accuracy: 0.7603 - val_loss: 0.5365 - val_accuracy: 0.7595

………………………………………………..
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.0110 - accuracy: 0.9957
Epoch 39: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0110 - accuracy: 0.9957 - val_loss: 2.1969 - val_accuracy: 0.7230
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.0100 - accuracy: 0.9951
Epoch 40: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0100 - accuracy: 0.9951 - val_loss: 2.3880 - val_accuracy: 0.7229
```

Table 5.5.4.b: CNN Model Training Epochs for Case 4d



```
test_loss, test_acc = model_4.evaluate(X_test_orig, y_test_orig)

392/392 [==============================] - 12s 30ms/step - loss: 2.3880 - accuracy: 0.7229

train_loss, train_acc = model_4.evaluate(X_train_orig, y_train_orig)

1174/1174 [==============================] - 34s 29ms/step - loss: 0.0096 - accuracy: 0.9958
```

Table 5.5.4.c: CNN Model Training Results for Case 4d

Chapter-6: Results

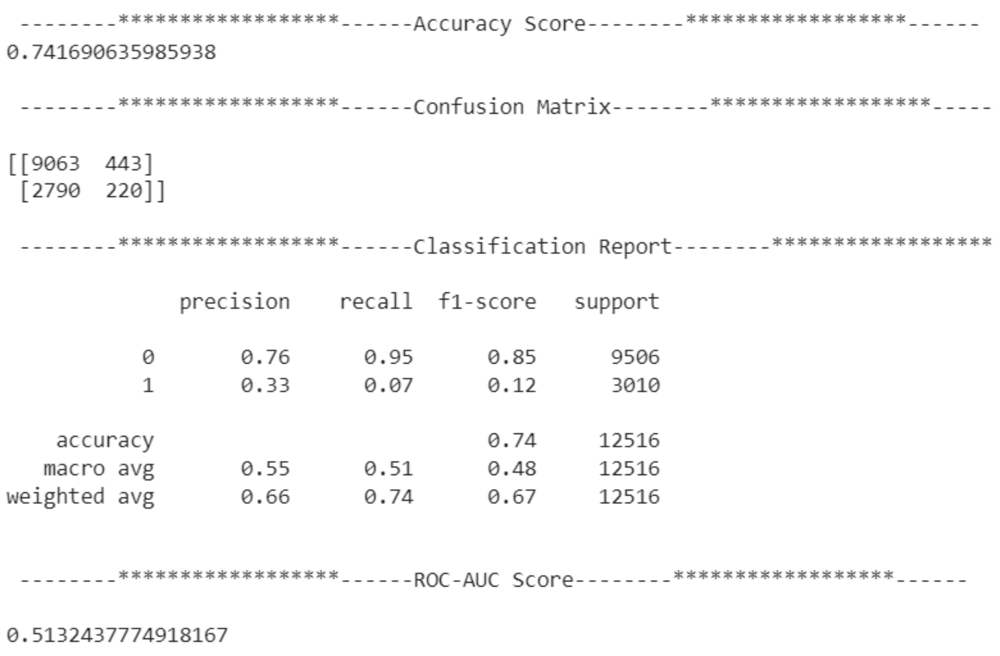
6.1 Final Model Selection

The final model selected for this dataset was the Passive Aggressive Classifier.

The rationale for selection is as below:

- The approach chosen for selection is an optimal one.
- Optimum Processing – Intensive operations like PCA and Deep Learning are not worth it unless they gave superior results. But as we can see from Chapter 5, that is not the case. Thus, we can cut out these.
- Optimum Features – Processing the fully encoded dataset would consume significant resources. Thus, feature selection would be done. Hence, we would need a model that had high performance after feature selection.
- Optimum Performance – Since we will be processing large amounts of data, it is better to have good performance.
- Optimum Results – Accuracy alone cannot be the yardstick for evaluation. Optimum values are desired for other metrics like ROC score, F1-score, etc.

6.2 Model Training Results



```
 --------******************------Accuracy Score--------******************------
0.741690635985938

 --------******************------Confusion Matrix--------******************-----

[[9063  443]
 [2790  220]]

 --------******************------Classification Report--------******************

              precision    recall  f1-score   support

           0       0.76      0.95      0.85      9506
           1       0.33      0.07      0.12      3010

    accuracy                           0.74     12516
   macro avg       0.55      0.51      0.48     12516
weighted avg       0.66      0.74      0.67     12516


 --------******************------ROC-AUC Score--------******************------

0.5132437774918167
```

Table 6.2.a: Classification Metrics for Model Training

```
 --------******************------ROC Curve--------******************------
```

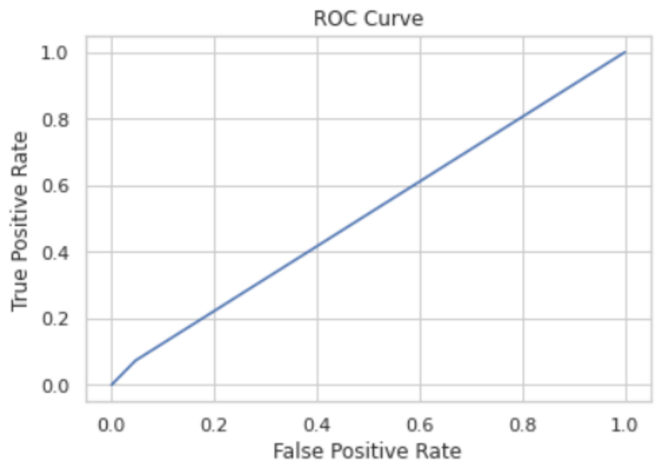


Figure 6.2.b: ROC Curve for Model Training

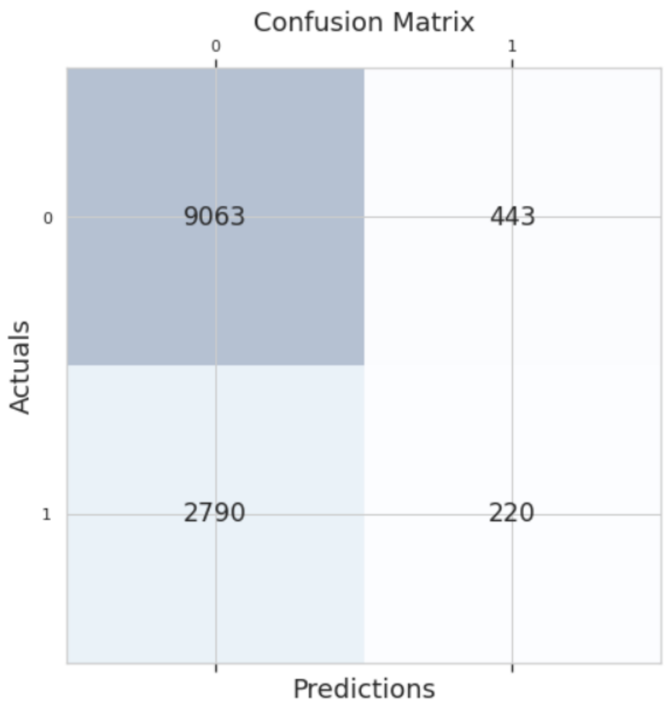
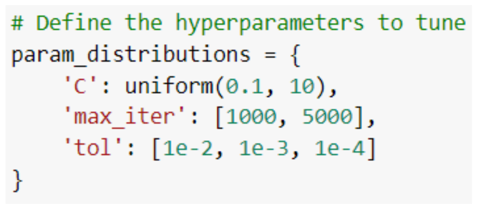


Table 6.2.c: Confusion Matrix for Model

6.3 Hyperparameter Tuning

The RandomizedSearchCV method was used to perform Hyperparameter Tuning.

The parameter space was as below:



```
# Define the hyperparameters to tune
param_distributions = {
    'C': uniform(0.1, 10),
    'max_iter': [1000, 5000],
    'tol': [1e-2, 1e-3, 1e-4]
}
```

Table 6.3.a: Parameter Space for Hyperparameter Tuning

The results of hyperparameter tuning are as below.

```
Best hyperparameters:  {'C': 0.5375406449743224, 'max_iter': 1000, 'tol': 0.001}
Best score:  0.7507641683869299
```

Figure 6.3.b: Hyperparameter Tuning Results

6.4 Final Model Evaluation

```
 --------******************------Accuracy Score--------******************------
0.7605465004793864

 --------******************------Confusion Matrix--------******************-----

[[9437   69]
 [2928   82]]

 --------******************------Classification Report--------******************

              precision    recall  f1-score   support

           0       0.76      0.99      0.86      9506
           1       0.54      0.03      0.05      3010

    accuracy                           0.76     12516
   macro avg       0.65      0.51      0.46     12516
weighted avg       0.71      0.76      0.67     12516


 --------******************------ROC-AUC Score--------******************------

0.5099919756922188
```

Table 6.4.a: Classification Report for Final Model

```
 --------******************------ROC Curve--------******************------
```

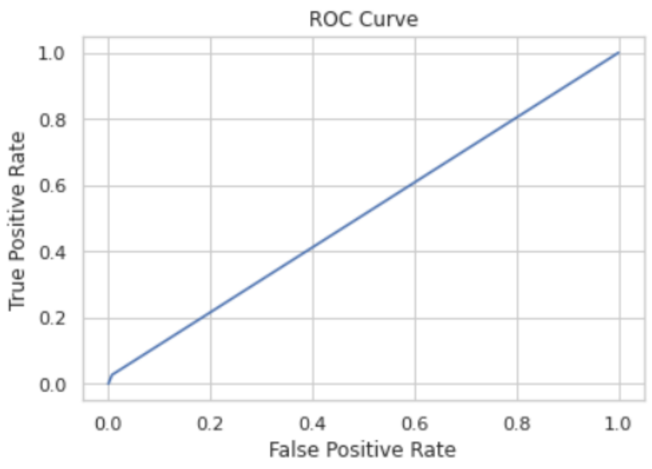
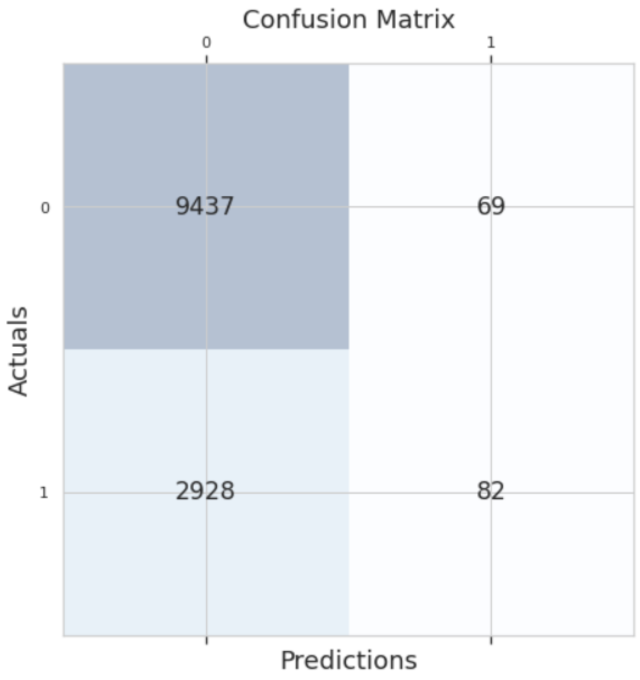


Figure 6.4.b: ROC Curve for Final Model



Confusion Matrix

Actuals \ Predictions	0	1
0	9437	69
1	2928	82

Table 6.4.c: Confusion Matrix for Final Model

Conclusions & Recommendations

The results obtained can be summarized as below.

Empirical Result Evaluation

- Final Model was built using Passive Aggressive Classifier.
- The accuracy score of the final model was 76%.
- The other evaluation parameters like precision, recall and F1-score for the final model were 0.71, 0.76 and 0.67 respectively.

Recommendations

The model performed best when implemented with Feature selection. A more thorough, robust, and sophisticated approach to feature engineering may be expected to give even better results.

Limitations

The precision, recall and F1-score of the positive respondents in the final model were 0.54, 0.03 and 0.05 respectively. This caused the overall ROC Score to drop to 0.51.

From a result standpoint alone, this makes the model sub-optimal for predictive purposes. However, the results from other similar projects may also be pointing to a certain level of inadequacy in the underlying dataset used for the project.

Comparative Summary & Conclusion

Strictly in the context of empirical parameters, this project has comparable results as the other experiments cited in the literature survey.

While this project used different preprocessing methods, algorithms, packages and model experimentation methods, the main limitation faced by the other authors of the previous projects, namely the sub-optimal performance on the smaller target class of positive respondents, was encountered in this project as well.

However, since this project focused on the reusable aspects of the process and deployment of a similar project with parameters and data that are expected to have more predictive power than the dataset used here, we can conclude that we have found a reasonable degree of success in this endeavor.

Directions for Future Work

The below are the ideas that can be implemented in future projects that are related to this dataset:

- Focus more on feature engineering. More sophisticated feature engineering techniques may help squeeze more predictive potential out of this particular dataset. Perhaps look more in-depth into the feature engineering methods used in one of the reference projects[23].
- Perform more experiments with feature selection as well. Try feature selection methods other than filter methods that I used in this project.

The below are the ideas that can be considered when implementing a similar project using a different dataset with more predictive potential:

Consult with the business teams on what type of error (type 1 or type 2) should be reduced and tune the model accordingly. For instance, what is the primary objective of lead prediction for the organization? Are we trying to reduce the number of calls to potential negative respondents that have to be made by the sales team? Or are we trying to maximize the no. of positive respondent leads that come out of the model? The ideal answer will be somewhere in between. Understand your cut-off point when tuning the model.

Location data like regions and cities could be excluded as a feature. This is because while this is important for macro-analytic purposes, it may not be as useful for lead viability prediction. The reason for this is because leads are very often followed up by branches at regional level. Thus, the model must compartmentalize for different regions and perform the lead prediction separately for different regions to maximize the efficacy. Another aspect that points in this direction is the results of feature selection from this experiment. We can see that most of the predictors belong to the region / city variables. In the data exploration phase as well, it was observed that region distribution is skewed in favor of specific regions. This effect should be separately analyzed.

Bibliography

1) Gupta, Sparsh. 'Health Insurance Lead Prediction - JOB-A-THON - AV.' Kaggle, https://www.kaggle.com/code/imsparsh/health-insurance-lead-prediction-job-a-thon-av/notebook (Accessed: 12 Mar 2023)

2) Thuring, F., Nielsen, J.P., Guillén, M. and Bolancé C. (2012) 'Selecting prospects for cross-selling financial products using multivariate credibility', Expert Systems with Applications, Volume 39, Issue 10, Pages 8809-8816, ISSN 0957-4174. Available at: https://doi.org/10.1016/j.eswa.2012.02.011

3) Scikit-learn, 'sklearn.feature_selection.SelectKBest'. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (Accessed: 12 Mar 2023)

4) Scikit-learn, 'sklearn.preprocessing.MinMaxScaler'. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (Accessed: 11 Mar 2023)

5) PyPi (2022), 'lazypredict 0.2.12'. Available at: https://pypi.org/project/lazypredict/ (Accessed: 12 Mar 2023)

6) Tutorial for Beginner, 'Classification Algorithm in Machine Learning'. Available at: https://tutorialforbeginner.com/classification-algorithm-in-machine-learning (Accessed: 12 Mar 2023)

7) Prof. Kernler, D., Elgin Community College, 'Section 3.5: The Five-Number Summary and Boxplots'. Available at: https://faculty.elgin.edu/dkernler/statistics/ch03/3-5.html (Accessed: 12 Mar 2023)

8) Harshit, K. (2018), 'Scaling vs Normalization'. Available at: https://kharshit.github.io/blog/2018/03/23/scaling-vs-normalization (Accessed: 12 Mar 2023)

9) Devopedia, 'Principal Component Analysis'. Available at: https://devopedia.org/principal-component-analysis (Accessed: 12 Mar 2023)

10) Machine Learning Journey, 'Adam Optimizer'. Available at: https://machinelearningjourney.com/index.php/2021/01/09/adam-optimizer/ (Accessed: 12 Mar 2023)

11) Shah, S. (2022), 'Convolutional Neural Network: An Overview'. Available at: https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/. (Accessed: 12 Mar 2023)

12) Wikipedia contributors. Receiver operating characteristic. Wikipedia, The Free Encyclopedia. March 3, 2023, 09:53 UTC. Available at: https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=1142602827. (Accessed March 12, 2023).

13) Inria Academy (2022), 'Hyperparameter tuning by randomized-search'. Available at: https://inria.github.io/scikit-learn-mooc/python_scripts/parameter_tuning_randomized_search.html. (Accessed: 12 Mar 2023)

14) Ragan, A. (2022), 'Taking the Confusion Out of Confusion Matrices'. Available at: https://towardsdatascience.com/taking-the-confusion-out-of-confusion-matrices-c1ce054b3d3e. (Accessed: 12 Mar 2023)

15) Hutson, G. (2021), 'Feature encoding methods – the Pandas way'. Available at: https://python-bloggers.com/2021/04/feature-encoding-methods-the-pandas-way/. (Accessed: 12 Mar 2023)

16) Great Learning Team (2022), 'Label Encoding in Python Explained'. Available at: https://www.mygreatlearning.com/blog/label-encoding-in-python/. (Accessed: 12 Mar 2023)

17) How to Learn Machine Learning, 'An Introduction to Feature Selection in Machine Learning'. Available at: https://howtolearnmachinelearning.com/articles/an-introduction-to-feature-selection-in-machine-learning/. (Accessed: 12 Mar 2023)

18) Statistics How To, 'Correlation Coefficient: Simple Definition, Formula, Easy Steps'. Available at: https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/. (Accessed: 12 Mar 2023)

19) Godoy, D. (2018), 'Understanding binary cross-entropy / log loss: a visual explanation'. Available at: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. (Accessed: 12 Mar 2023)

20) Harshit, K. (2018), 'A visual introduction to eigenvectors and eigenvalues'. Available at: https://kharshit.github.io/blog/2018/05/11/a-visual-introduction-to-eigenvectors-and-eigenvalues (Accessed: 12 Mar 2023)

21) Hisamoto, S. (2013), 'Perceptron, Support Vector Machine, and Passive Aggressive Algorithm.'. Available at: https://speakerdeck.com/sorami/perceptron-support-vector-machine-and-passive-aggressive-algorithm?slide=56 (Accessed: 12 Mar 2023)

22) Santana, Miguel. "Health Insurance Lead Prediction." GitHub, https://github.com/miguelangelsantana/Health-Insurance-Lead-Prediction. (Accessed: 12 Mar 2023)

23) Jose, Basil. "Health-Insurance-Lead-Prediction." GitHub, https://github.com/basilkjose/Health-Insurance-Lead-Prediction. (Accessed: 12 Mar 2023)

24) Atkin, Gabriel. "Health Insurance Customer Response Prediction." Kaggle, https://www.kaggle.com/code/gcdatkin/health-insurance-customer-response-prediction. (Accessed: 12 Mar 2023)

Appendix

Appendix A - Theoretical Concepts

A.1. Outlier Detection with Five-Number Summary

Outlier detection is the process of identifying data points that are significantly different from the other data points in a dataset. One commonly used approach for outlier detection is to use the five-number summary.

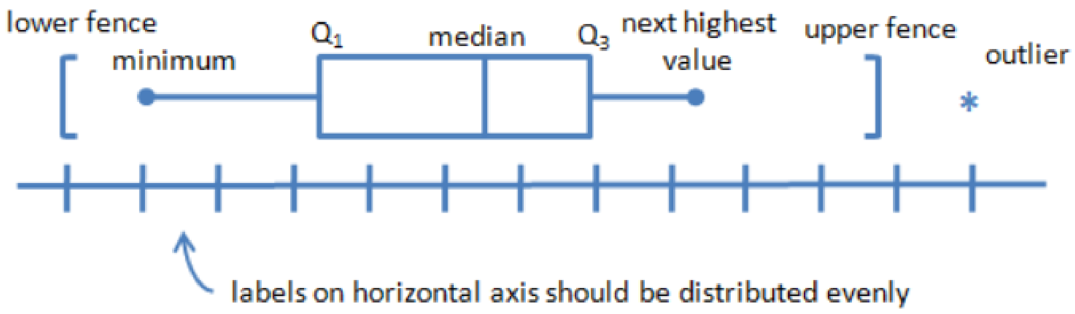


Figure A.1.a: Five Number Summary[7]

The five-number summary consists of the minimum value, the maximum value, the median (or middle value), and the first and third quartiles. These values are used to define the interquartile range (IQR), which is the range of values that contain the middle 50% of the data. Any data point that falls outside of the IQR is considered an outlier.

The process for using the five-number summary for outlier detection is as follows:

- Calculate the minimum and maximum values in the dataset.
- Calculate the median (middle value) of the dataset.
- Calculate the first quartile (Q1), which is the value that separates the bottom 25% of the data from the top 75% of the data.
- Calculate the third quartile (Q3), which is the value that separates the bottom 75% of the data from the top 25% of the data.
- Calculate the interquartile range (IQR) by subtracting Q1 from Q3.
- Define the upper outlier limit (UOL) as Q3 + 1.5IQR and the lower outlier limit (LOL) as Q1 - 1.5IQR.
- Any data point that falls above the UOL or below the LOL is considered an outlier.

By using the five-number summary and the IQR, outlier detection can be performed in a robust and efficient manner. However, it is important to note that the five-number summary approach assumes that the data is normally distributed, and may not work as well for datasets with skewed distributions or other non-normal characteristics. Additionally, it is important to carefully consider the context of the data and the potential causes of outliers before taking any action based on outlier detection results.

A.2. Min-Max Scaling

Min-max scaling is a type of data normalization technique used in machine learning and data analysis to rescale the values of a numerical feature to a fixed range between 0 and 1.

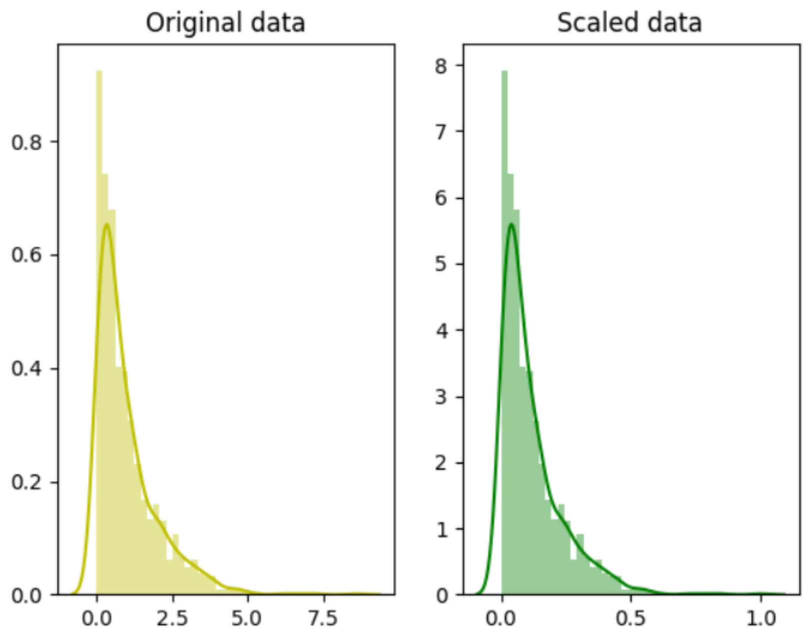


Figure A.2.a: Original vs. Scaled[8]

This technique works by subtracting the minimum value of the feature and then dividing by the range (the difference between the maximum and minimum values).

The formula for min-max scaling is:

$$\boxed{\mathbf{x\_scaled = (x - min(x)) / (max(x) - min(x))}}$$

where x is the original value of the feature, x_scaled is the rescaled value, min(x) is the minimum value of the feature, and max(x) is the maximum value of the feature.

Min-max scaling has several advantages in data analysis and machine learning:

- Min-max scaling preserves the shape of the original distribution while rescaling the values to a common scale, making it a non-distorting normalization technique.
- Min-max scaling is easy to implement and computationally efficient, making it a popular choice for data preprocessing.
- Min-max scaling is effective in reducing the impact of outliers in the data. Since the range is based on the difference between the maximum and minimum values, outliers have less influence on the final rescaled values.
- Min-max scaling is useful when the absolute values of the features are not as important as their relative values.

However, min-max scaling may not be appropriate in all situations.

- For example, if the dataset contains significant outliers, min-max scaling may not be able to capture the full range of the data.
- Additionally, if the range of the data is already relatively small, min-max scaling may not provide much benefit. In these cases, other normalization techniques such as z-score normalization may be more appropriate.

A.3. Principal Component Reduction

Principal Component Analysis (PCA) is a technique used in machine learning and data analysis to reduce the dimensionality of a large dataset by identifying patterns and relationships between variables.

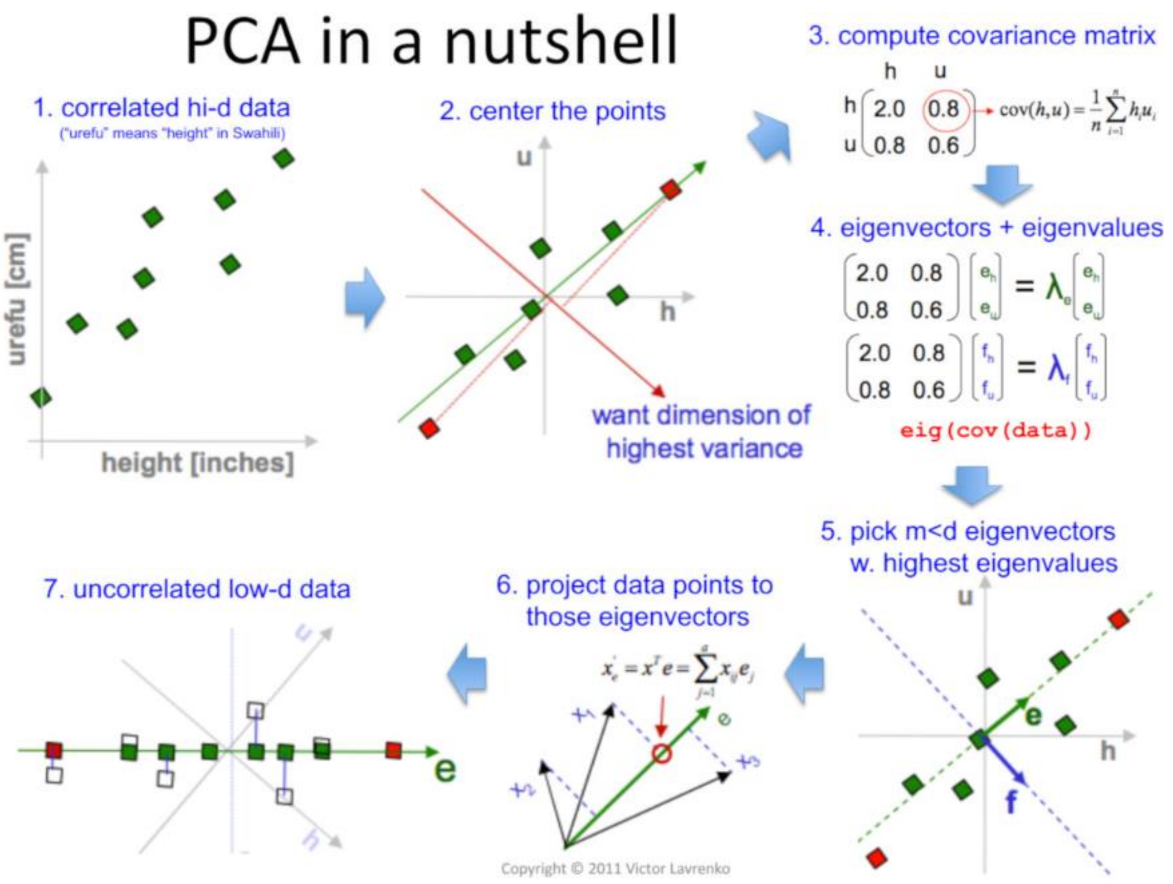

PCA in a nutshell
1. correlated hi-d data
("urefu" means "height" in Swahili)
urefu [cm]
height [inches]
2. center the points
u
h
want dimension of highest variance
3. compute covariance matrix
4. eigenvectors + eigenvalues
eig(cov(data))
5. pick m<d eigenvectors w. highest eigenvalues
e
f
6. project data points to those eigenvectors
7. uncorrelated low-d data
Copyright © 2011 Victor Lavrenko

Figure A.3.a: Principal Component Analysis[9]

PCA works by transforming a set of correlated variables into a set of uncorrelated variables called principal components. These principal components are linear combinations of the original variables and are calculated in such a way that the first principal component captures the maximum amount of variance in the data, the second principal component captures the maximum amount of variance remaining after the first component has been removed, and so on.

The steps involved in PCA are:

- **<u>Standardize the data:</u>** The variables in the dataset are standardized by subtracting the mean and dividing by the standard deviation, so that all variables have the same scale.
- **<u>Calculate the covariance matrix:</u>** The covariance matrix is calculated for the standardized variables. The covariance matrix describes the relationship between variables and is used to calculate the principal components.
- **<u>Calculate the eigenvectors and eigenvalues:</u>** The eigenvectors and eigenvalues of the covariance matrix are calculated. The eigenvectors represent the directions of maximum variance in the data, and the eigenvalues represent the amount of variance explained by each eigenvector.

- **<u>Choose the number of principal components:</u>** The number of principal components to retain is chosen based on the amount of variance explained by each component. Typically, a threshold value (such as 80% or 90% of the total variance) is used to determine the number of principal components to retain.
- **<u>Transform the data:</u>** The original data is transformed into the new coordinate system defined by the principal components.

PCA has several applications in data analysis and machine learning, including:

- **<u>Dimensionality reduction</u>**: PCA can be used to reduce the dimensionality of a dataset while retaining most of the important information.
- **<u>Data visualization:</u>** PCA can be used to visualize high-dimensional data in a lower-dimensional space.
- **<u>Noise reduction:</u>** PCA can be used to remove noise and other sources of variability in a dataset.

PCA is a powerful technique for exploring and analyzing high-dimensional datasets and can provide valuable insights into the underlying structure of the data.

A.4. Adam Optimizer

Adam (Adaptive Moment Estimation) is a popular optimization algorithm used in machine learning for stochastic gradient descent (SGD), which is a method used to update the weights and biases of a neural network during training.

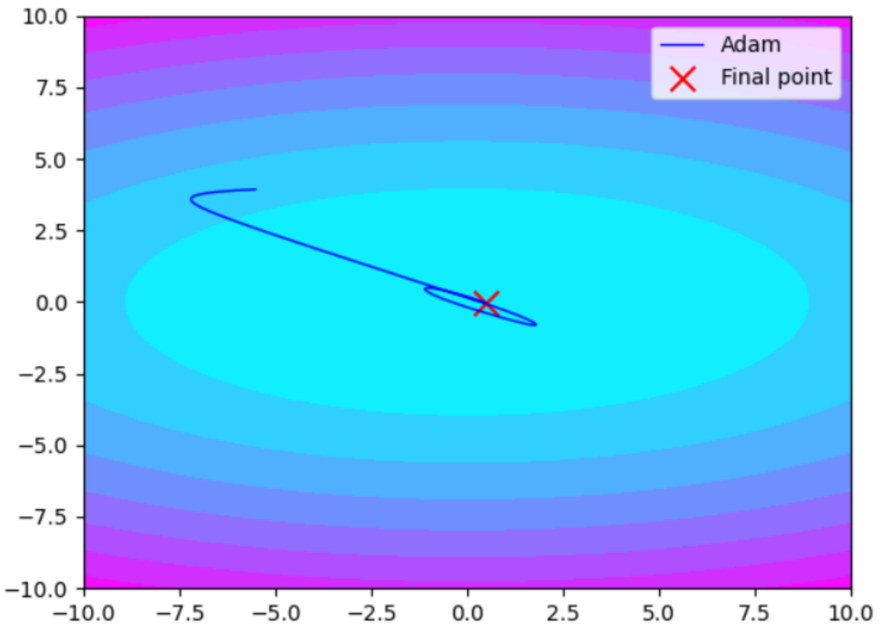


Figure A.4.a: Adam Optimizer[10]

Adam optimizer is a combination of two other optimization algorithms: AdaGrad and RMSProp.

Like AdaGrad, Adam optimizer adapts the learning rate for each parameter individually, based on the historical gradients. Like RMSProp, Adam optimizer also uses an exponentially decaying average of past squared gradients to calculate the learning rate.

The steps involved in Adam optimizer are as follows:

- **Initialization:** The initial values of the parameters, such as the learning rate, momentum, and decay rates, are set.
- **Compute gradients:** The gradients of the loss function with respect to the parameters are computed using backpropagation.
- **Calculate moving averages of past gradients:** Adam optimizer calculates two exponentially decaying averages of past gradients, one for the gradient and one for the square of the gradient.
- **Update parameters:** The parameters are updated using a weighted sum of the moving averages of past gradients, with weights that are determined by the learning rate and the momentum.
- **Repeat:** The process of computing gradients, calculating moving averages, and updating parameters is repeated until the desired level of accuracy is achieved.

Adam optimizer has several advantages over other optimization algorithms, such as:

- **Faster convergence:** Adam optimizer is known to converge faster than other optimization algorithms, such as stochastic gradient descent and Adagrad.
- **Adaptive learning rates:** Adam optimizer adapts the learning rate for each parameter individually, which helps to prevent the algorithm from getting stuck in local minima.
- **Robustness to noisy gradients:** Adam optimizer is robust to noisy gradients, which can occur in large datasets or in the presence of outliers.

Adam optimizer is a powerful optimization algorithm that has become a popular choice for training deep neural networks in machine learning due to its fast convergence and adaptive learning rates.

A.5. Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of artificial neural network that are designed to process and analyze images, videos, and other multidimensional data with spatial relationships.

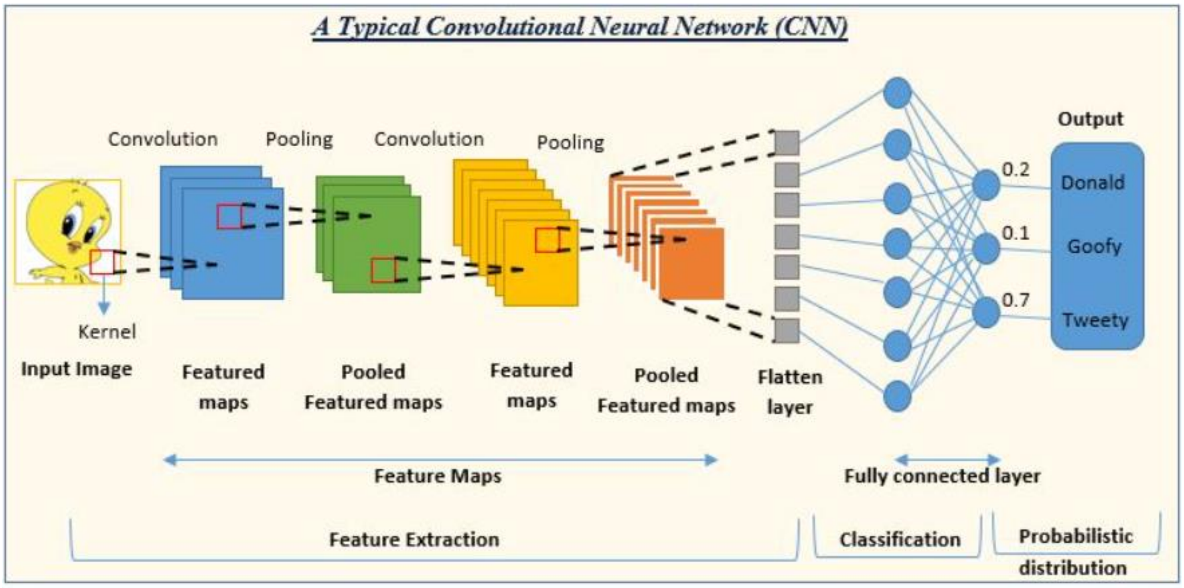


Figure A.5.a: Convolutional Neural Network[11]

CNNs use a special type of layer called a convolutional layer that applies a set of filters or kernels to the input image. Each filter detects a specific feature or pattern in the input image, such as edges, corners, or textures. The filters are applied to the input image using a sliding window approach, where the filter moves across the image in small steps and computes the dot product between the filter weights and the corresponding pixel values in the image.

The output of the convolutional layer is a set of feature maps, where each feature map corresponds to a specific filter and contains the activations of the filter across the input image. The feature maps are then passed through a non-linear activation function, such as ReLU, to introduce non-linearity into the model.

CNNs also typically include pooling layers, which downsample the feature maps by taking the maximum or average value within a small window. Pooling helps to reduce the spatial dimensions of the feature maps, while preserving the most important features.

The final layers of a CNN are typically fully connected layers, which are similar to the layers in a traditional neural network. These layers combine the features from the previous layers and produce the final output of the network.

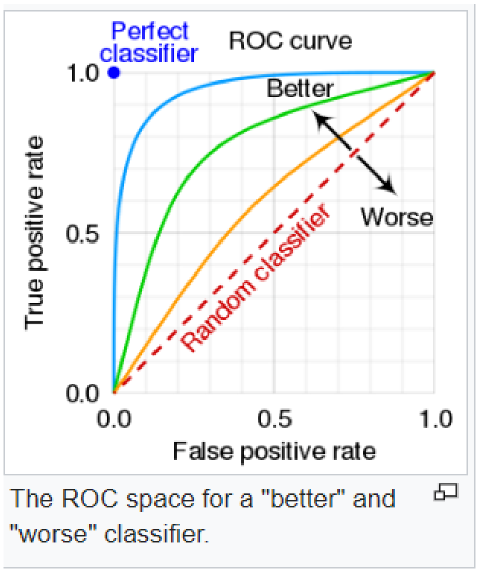
CNNs have several advantages over traditional neural networks for image processing and analysis, including:

- **Local connectivity:** CNNs exploit the local spatial correlations in the input image, which makes them well-suited for image analysis tasks.
- **Parameter sharing:** CNNs share the weights of the filters across the input image, which reduces the number of parameters and makes the model more efficient.
- **Translation invariance:** CNNs are invariant to translations in the input image, which means that they can detect the same features regardless of their position in the image.

CNNs have revolutionized the field of computer vision and are widely used in a variety of applications, such as object detection, image segmentation, and facial recognition.

A.6. ROC Curve

A Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model, such as a logistic regression or a support vector machine. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds.



The ROC space for a "better" and "worse" classifier.

Figure A.6.a: ROC Curve[12]

The TPR is the proportion of true positives (i.e., correctly classified positive samples) among all the actual positive samples, while the FPR is the proportion of false positives (i.e., incorrectly classified negative samples) among all the actual negative samples.

To create an ROC curve, the model predictions for a set of test samples are first sorted in descending order by their predicted probabilities or scores. The classification threshold is then varied from 0 to 1, and at each threshold, the TPR and FPR are calculated.

The ROC curve is obtained by plotting the TPR on the y-axis against the FPR on the x-axis. A random classifier would produce an ROC curve that is a straight line from the origin to (1,1), while a perfect classifier would produce an ROC curve that passes through the top left corner of the plot.

The area under the ROC curve (AUC) is a commonly used metric to evaluate the performance of a binary classification model. The AUC ranges from 0 to 1, where an AUC of 0.5 indicates that the classifier is no better than a random classifier, while an AUC of 1 indicates perfect classification performance.

A high AUC indicates that the classifier is able to distinguish between positive and negative samples with high accuracy, and a low AUC indicates that the classifier is not able to separate the two classes well.

An ROC curve provides a visual representation of the performance of a binary classification model at different classification thresholds, and the AUC is a summary metric that quantifies the overall performance of the model.

A.7. Hyperparameter Tuning with Randomized Search

RandomizedSearchCV is a technique for hyperparameter tuning in machine learning that helps to find the optimal set of hyperparameters for a given model by searching a defined parameter space.

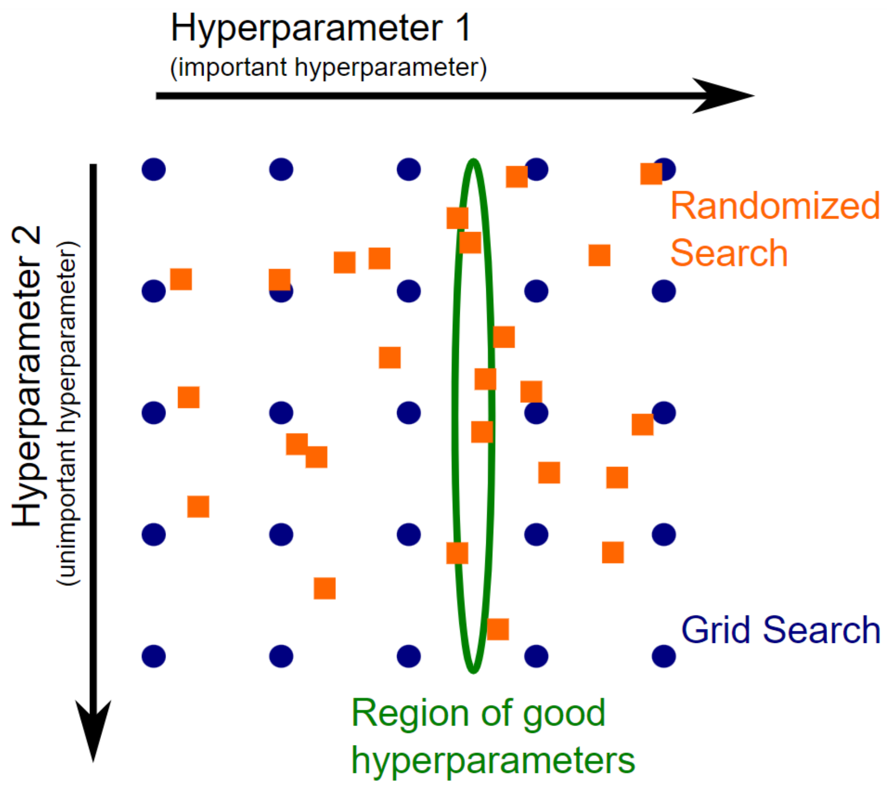

Hyperparameter 1
(important hyperparameter)
Hyperparameter 2
(unimportant hyperparameter)
Randomized Search
Grid Search
Region of good hyperparameters

Figure A.7.a: Hyperparameter Tuning[13]

Hyperparameters are model configuration parameters that cannot be learned directly from the data, such as learning rate, regularization strength, and number of hidden layers. The optimal values of these hyperparameters depend on the dataset and the specific task, and can significantly affect the performance of the model.

RandomizedSearchCV is an extension of the grid search technique, which exhaustively searches all possible combinations of hyperparameters within a defined parameter grid. However, RandomizedSearchCV differs from grid search in that it does not perform an exhaustive search, but rather samples a defined number of hyperparameter configurations at random from the parameter space.

To use RandomizedSearchCV, the user must first define a parameter space, which is a dictionary that specifies the hyperparameters to be tuned and their possible values. The user also specifies the number of parameter settings that will be sampled, as well as the number of cross-validation folds to be used in evaluating the performance of each configuration.

RandomizedSearchCV then generates a specified number of random hyperparameter configurations and evaluates the performance of each configuration using cross-validation. The performance metric used to evaluate each configuration can be specified by the user, and is typically a measure of the model's accuracy, such as F1 score, precision, or recall.

At the end of the search process, RandomizedSearchCV returns the hyperparameter configuration that achieved the best performance on the validation set, as well as the corresponding performance metric. The optimal hyperparameters can then be used to train the model on the entire training set and evaluate its performance on the test set.

RandomizedSearchCV is a useful technique for hyperparameter tuning because it reduces the computational cost and search space required for finding the optimal set of hyperparameters, while still providing a good chance of finding the optimal configuration.

A.8. Passive Aggressive Classifier

The Passive-Aggressive (PA) algorithm is a type of online learning algorithm that is commonly used for binary classification problems. The algorithm is particularly useful in situations where the data is large and streaming, and the underlying distribution of the data may change over time.

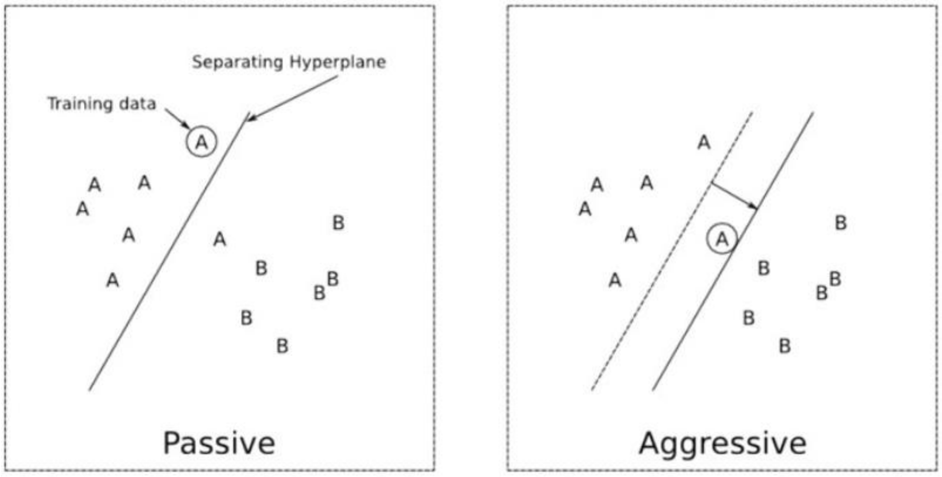


Figure A.8.a: Passive Aggressive Classifier Process[21]

The PA algorithm works by maintaining a weight vector that is used to classify incoming data points. The algorithm updates the weight vector iteratively, with each update based on the most recent data point. The update rule is designed to minimize the classification error while still maintaining the current performance of the classifier.

The name "Passive-Aggressive" comes from the fact that the algorithm can be either passive or aggressive in its updates, depending on whether the most recent data point is correctly classified or not. If the data point is correctly classified, the algorithm remains passive and does not update the weight vector. However, if the data point is misclassified, the algorithm becomes aggressive and updates the weight vector in a way that minimizes the error.

There are several variants of the PA algorithm, including the PA-I, PA-II, and PA-III algorithms, which differ in their update rules and aggressiveness. In general, the PA algorithm is known for its simplicity, efficiency, and ability to handle large and streaming datasets.

One drawback of the PA algorithm is that it can be sensitive to noise and outliers in the data, which can lead to overfitting. Therefore, it is important to preprocess the data and perform feature selection or feature extraction before applying the PA algorithm.

In summary, the Passive-Aggressive algorithm is a type of online learning algorithm that is used for binary classification problems, particularly in situations where the data is large and streaming. The algorithm updates the weight vector iteratively based on the most recent data point, with the update rule designed to minimize the classification error while still maintaining the current performance of the classifier.

A.9. Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) that the model predicted for a given set of data.

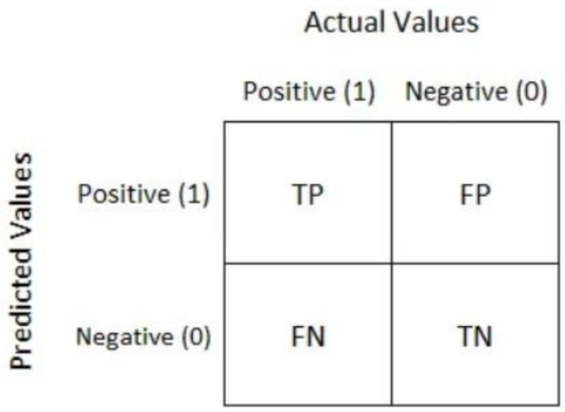


Figure A.9.a: Confusion Matrix[14]

The rows of the confusion matrix represent the actual labels of the data, while the columns represent the predicted labels. The diagonal of the matrix represents the correct predictions (true positives and true negatives), while the off-diagonal elements represent the incorrect predictions (false positives and false negatives).

A confusion matrix can be used to calculate several performance metrics for a classification model, including accuracy, precision, recall, and F1 score.

These metrics can be defined as follows:

- **Accuracy:** the proportion of correct predictions out of the total number of predictions. It is calculated as (TP+TN)/(TP+TN+FP+FN).
- **Precision:** the proportion of true positives out of the total number of positive predictions. It is calculated as TP/(TP+FP).
- **Recall** (also known as **sensitivity**): the proportion of true positives out of the total number of actual positives. It is calculated as TP/(TP+FN).
- **F1 score:** the harmonic mean of precision and recall. It is calculated as 2*precision*recall/(precision+recall).

The confusion matrix can also be visualized using a heatmap, with the diagonal cells highlighted in a different color to draw attention to the correct predictions. This can help to identify which classes the model is performing well on and which classes it is struggling with.

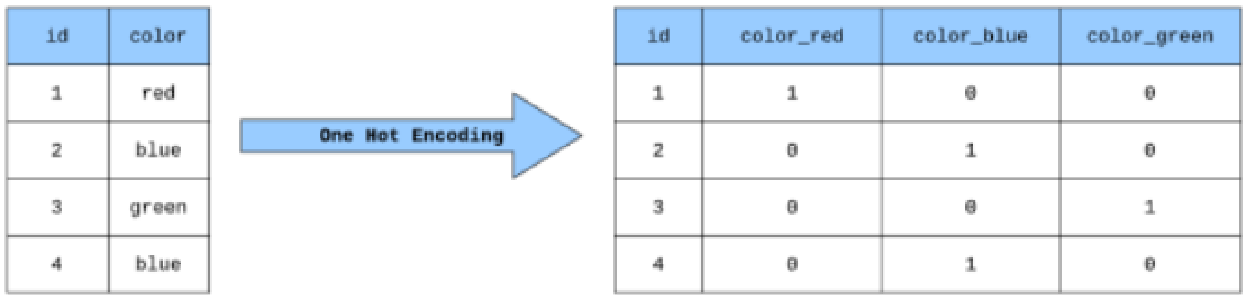
In summary, a confusion matrix is a table that shows the true positives, true negatives, false positives, and false negatives that a classification model predicted for a given set of data. It can be used to calculate several performance metrics for the model, including accuracy, precision, recall, and F1 score, and can be visualized using a heatmap.

A.10. Feature Encoding

Feature encoding in machine learning is the process of converting categorical variables (such as text, strings or non-numerical data) into a numerical format that can be used in machine learning models. This is important because most machine learning algorithms are designed to work with numerical data and cannot handle categorical variables directly.

There are several methods for feature encoding in machine learning, including:

- **One-Hot Encoding:** This is a method that converts each category into a binary vector where each element represents the presence or absence of that category. For example, if we have a categorical variable "color" with three categories (red, green, blue), one-hot encoding would create three new binary variables (red, green, blue) with a value of 1 for the category and 0 for all other categories.



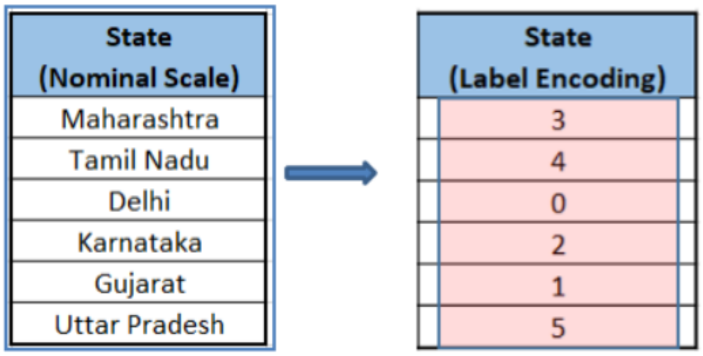
id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Figure A.10.a: One-Hot Encoding[15]

- **Label Encoding:** This is a method that assigns a unique integer value to each category. For example, if we have a categorical variable "color" with three categories (red, green, blue), label encoding would assign the values 1, 2 and 3 to the categories.



State (Nominal Scale)
Maharashtra
Tamil Nadu
Delhi
Karnataka
Gujarat
Uttar Pradesh

State (Label Encoding)
3
4
0
2
1
5

Figure A.10.b: Label Encoding[16]

- **Binary Encoding:** This is a method that converts each category into a binary format. For example, if we have a categorical variable "color" with three categories (red, green, blue), binary encoding would create two new binary variables where each variable represents one of the categories. For example, red would be encoded as 01, green as 10 and blue as 11.
- **Frequency Encoding:** This is a method that replaces each category with its frequency in the dataset. For example, if we have a categorical variable "color" with three categories (red, green, blue) and the dataset contains 10 red, 5 green and 3 blue, frequency encoding would replace each category with its respective frequency (10, 5, 3).
- **Target Encoding:** This is a method that replaces each category with the mean target value (or other statistical metric) for that category. For example, if we have a categorical variable "color" and a binary target variable, target encoding would replace each category with the mean target value for that category.

Feature encoding is an important step in machine learning as it can significantly affect the performance of a model. Choosing the appropriate method of feature encoding depends on the nature of the data and the problem being solved.

A.11. Feature Selection

Feature selection is the process of selecting a subset of relevant features (or variables) from a larger set of features that are available in a dataset. This is an important step in many machine learning classification tasks, as it can improve the accuracy and performance of a model by reducing the complexity of the input data and focusing on the most informative features.

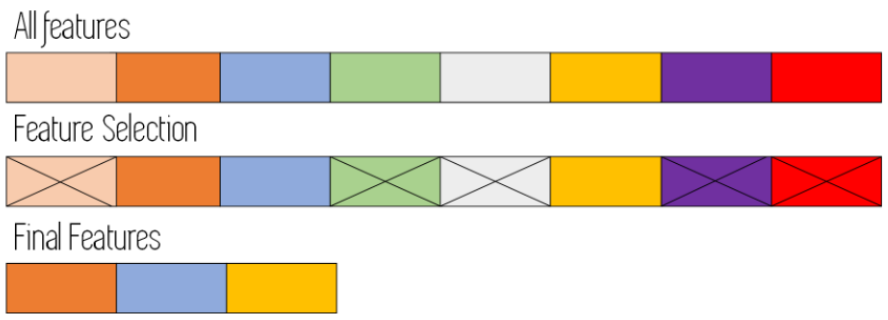


Figure A.11.a: Feature Selection[17]

There are several methods for feature selection in machine learning classification, including:

Filter Methods: These methods select features based on statistical measures such as correlation, mutual information, and chi-square tests. Features are ranked according to their relevance to the target variable, and a threshold is set to select the top features.

Wrapper Methods: These methods evaluate the performance of a model by training it on different subsets of features. They use a search algorithm to find the best subset of features that maximizes the model's accuracy.

Embedded Methods: These methods select features as part of the model training process. They are often used in algorithms like decision trees and linear regression, where features are assigned weights that indicate their importance.

Dimensionality Reduction Methods: These methods transform the input data into a lower-dimensional space while preserving the most important information. Techniques like principal component analysis (PCA) and linear discriminant analysis (LDA) can be used to reduce the number of features and improve the performance of a model.

It's important to note that feature selection is a critical step in machine learning classification, as it can significantly affect the accuracy and performance of a model. Therefore, it's important to carefully consider which method to use and ensure that the selected features are relevant and informative for the task at hand.

Appendix B - Mathematical Concepts

B.1. Correlation Coefficient

Correlation coefficient is a statistical measure that indicates the strength and direction of the relationship between two variables. In machine learning, correlation coefficient is often used to assess the linear relationship between features and target variables.

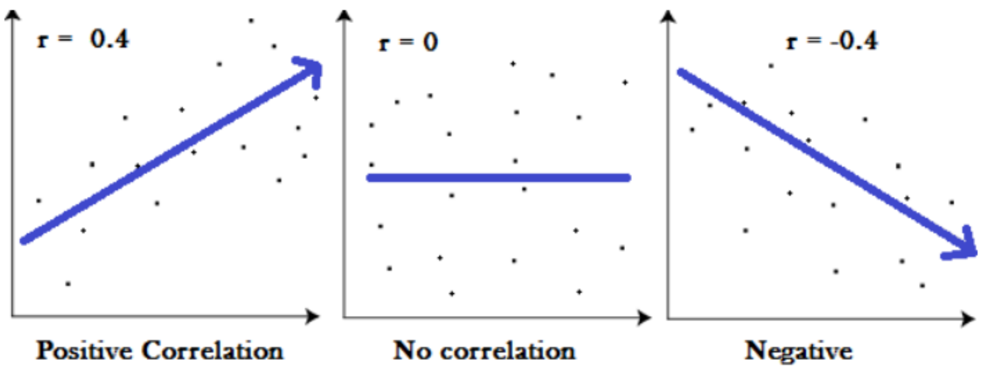


Figure B.1.a: Correlation Types[18]

The correlation coefficient can range from -1 to 1, with a value of -1 indicating a perfect negative correlation (as one variable increases, the other decreases), a value of 0 indicating no correlation, and a value of 1 indicating a perfect positive correlation (as one variable increases, the other increases).

In machine learning, the correlation coefficient can be used to identify features that are strongly correlated with the target variable. Highly correlated features may be redundant and can cause problems such as overfitting or numerical instability. Therefore, it is often desirable to remove highly correlated features from the dataset or to use feature selection methods to select a subset of features that are not highly correlated.

Correlation coefficient can be used to identify highly correlated features and to remove them to prevent problems such as overfitting or numerical instability.

B.2. Binary Cross-Entropy

Binary cross-entropy is a loss function commonly used in binary classification tasks in machine learning. It measures the difference between the predicted probabilities of the positive class

(class 1) and the actual labels (either 0 or 1). The goal of the binary cross-entropy loss function is to minimize this difference during training.

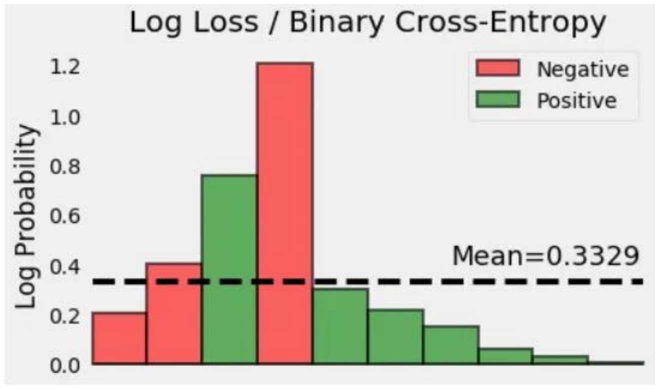


Figure B.2.a: Binary Cross Entropy[19]

Mathematically, the binary cross-entropy loss function is defined as follows:

For a single instance, the loss function is given by:

$$\boxed{\mathbf{L(y, \hat{y}) = - (y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y}))}}$$

where y is the actual binary label (either 0 or 1) and $\hat{y}$ is the predicted probability of the positive class (class 1).

For a dataset with multiple instances, the loss function is the average of the loss over all instances:

$$\boxed{\mathbf{L(Y, \hat{Y}) = - (1/m) * sum(y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y}))}}$$

where Y is the vector of actual binary labels, $\hat{Y}$ is the vector of predicted probabilities, and m is the number of instances in the dataset.

The binary cross-entropy loss function is used in binary classification tasks to optimize the weights of the neural network during training. During training, the model attempts to minimize the value of the loss function by adjusting the weights in order to make better predictions. The smaller the loss function, the better the model's predictions.

The goal is to minimize this difference during training in order to improve the model's predictions.

B.3. Eigenvalues & Eigenvectors

Eigenvalues and eigenvectors are mathematical concepts that are commonly used in linear algebra and machine learning. They are important in understanding the behavior of linear transformations and matrices.

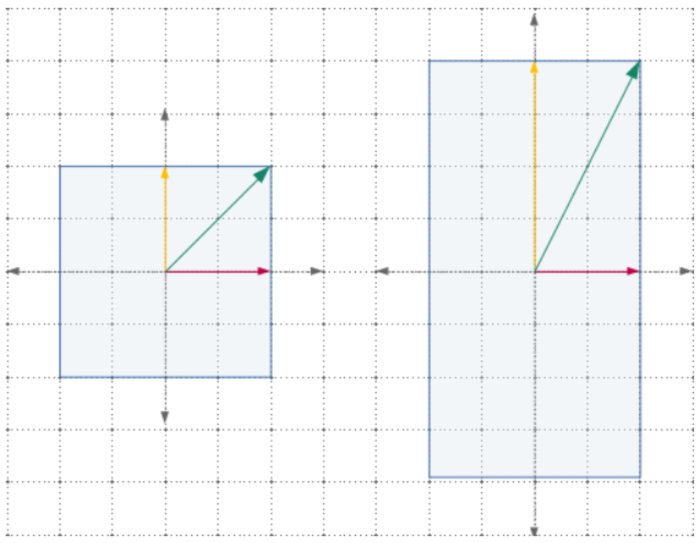


Figure B.3.a: Eigenvector[20]

An eigenvector is a vector that, when a linear transformation is applied to it, only changes in magnitude (i.e., it may change direction but its length remains the same). More formally, an eigenvector v of a square matrix A is a non-zero vector that satisfies the following equation:

$$\boxed{\mathbf{A * v = \lambda * v}}$$

where λ is a scalar value known as the eigenvalue associated with the eigenvector v. In other words, when matrix A is applied to the eigenvector v, the result is a new vector that is a multiple of the original eigenvector, with the scalar multiple λ. Eigenvectors and eigenvalues always occur in pairs, and the set of all eigenvectors associated with a matrix is known as the eigenspace of that matrix.

Eigenvalues represent the factor by which an eigenvector is scaled when a linear transformation is applied to it. They are scalar values that are associated with a matrix and are often used to describe the behavior of the matrix under various transformations. For example, the eigenvalues of a covariance matrix in a principal component analysis (PCA) are used to determine the amount of variance in the data that is explained by each principal component.

In machine learning, eigenvalues and eigenvectors are often used in feature extraction and dimensionality reduction techniques such as PCA. By finding the eigenvectors and eigenvalues of a dataset's covariance matrix, it is possible to identify the most important features or dimensions that explain the most variance in the data.