**FINAL ASSIGNMENT**

### Title

**Enhancing Predictive Models for Customer Churn Using Telco Dataset: A Comprehensive Analysis**

### Author/s

גלית רייפר: 029073558

רני וונג איטיק: 314774043

יצחק שרון: 307930271

### Keywords

Customer Churn, Predictive Modeling, Decision Tree, Neural Network, SMOTE, PCA, Data Preprocessing, Hyperparameter Optimization

### Abstract

Customer churn prediction is essential for organizations seeking to enhance customer retention and reduce revenue losses. This study explores techniques for improving predictive models using the Telco Customer Churn dataset. A combination of preprocessing methods, such as outlier removal, SMOTE for class balancing, and PCA for dimensionality reduction, is employed. Both Decision Tree and Neural Network models are examined, with a focus on hyperparameter tuning and architectural optimization. Results reveal that advanced preprocessing and optimization techniques significantly enhance model accuracy and reduce overfitting. This paper presents a detailed methodology, analysis of results, and insights into future directions for improving churn prediction performance.

### Introduction

Customer churn, defined as the termination of a customer's relationship with a service provider, is a pressing concern for businesses across industries. Predicting churn accurately enables organizations to implement effective retention strategies, mitigating financial losses. This study investigates the Telco Customer Churn dataset, employing Decision Tree and Neural Network models to predict churn. Through extensive preprocessing, hyperparameter tuning, and architectural improvements, this paper aims to provide a comprehensive approach to optimizing predictive models for customer churn.

### Literature Review

Research in customer churn prediction has highlighted the importance of machine learning models, particularly Decision Trees and Neural Networks. Decision Trees are valued for their interpretability and efficacy with structured datasets, while Neural Networks excel in capturing complex relationships in data. Addressing class imbalance through methods like SMOTE and reducing dimensionality using PCA have been shown to improve model performance. Recent advancements in hyperparameter optimization have further enhanced the predictive power of these models. This paper builds upon existing literature by integrating these techniques into a unified framework for churn prediction.

### Methodology

#### 1)Dataset Description

The Telco Customer Churn dataset, sourced from Kaggle, contains approximately 7,000 rows and 20 columns. Features include demographic details, account information, and service usage patterns. The target variable is binary, indicating whether a customer has churned (Yes) or not (No).

The **Telco Customer Churn dataset** is a widely used dataset for understanding customer retention and predicting churn (when customers leave a service). Here's a comprehensive breakdown of the dataset:

**Dataset Overview**

* **Source**: Kaggle (Blastchar)
* **Use Case**: Predicting customer churn in a telecom company.
* **Dataset Focus**: Binary classification where the target variable is whether a customer has churned (Yes) or not (No).
* **Size**: Approximately **7,000 rows** and **20 columns**.

**Columns and Features**

The dataset includes customer demographics, account details, and service usage. Here are the key features:

**Demographic Information**

1. **CustomerID**: Unique identifier for each customer.
2. **Gender**: Male or Female.
3. **SeniorCitizen**: Binary indicator (0 = Not a senior citizen, 1 = Senior citizen).
4. **Partner**: Binary (Yes = Has a partner, No = No partner).
5. **Dependents**: Binary (Yes = Has dependents, No = No dependents).

**Service Subscription**

1. **PhoneService**: Binary (Yes = Has phone service, No = No phone service).
2. **MultipleLines**: Whether the customer has multiple phone lines (No, Yes, or No phone service).
3. **InternetService**: Type of internet service (DSL, Fiber optic, or No).
4. **OnlineSecurity**: Whether the customer has online security add-on (Yes, No, or No internet service).
5. **OnlineBackup**: Whether the customer has online backup add-on.
6. **DeviceProtection**: Device protection add-on subscription.
7. **TechSupport**: Tech support add-on subscription.
8. **StreamingTV**: Subscription to streaming TV service.
9. **StreamingMovies**: Subscription to streaming movie service.

**Account Information**

1. **Contract**: Type of contract (Month-to-month, One year, or Two year).
2. **PaperlessBilling**: Binary (Yes = Paperless billing, No = Printed bills).
3. **PaymentMethod**: Customer's method of payment (e.g., Electronic check, Credit card, Mailed check, Bank transfer).
4. **MonthlyCharges**: Monthly fee charged to the customer (continuous numerical feature).
5. **TotalCharges**: Total amount billed to the customer to date (continuous numerical feature).

#### 2) Basic Eda

The **Telco Customer Churn dataset** contains **21 features (columns)**.

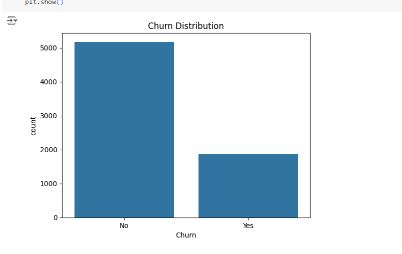
17 categorial and 3 numeric.

Step 1: Overview of the dataset-explore: Dataset Shape, Columns and Data Types, Missing Values, Duplicate Rows. We also explore summary statistic of missing values.

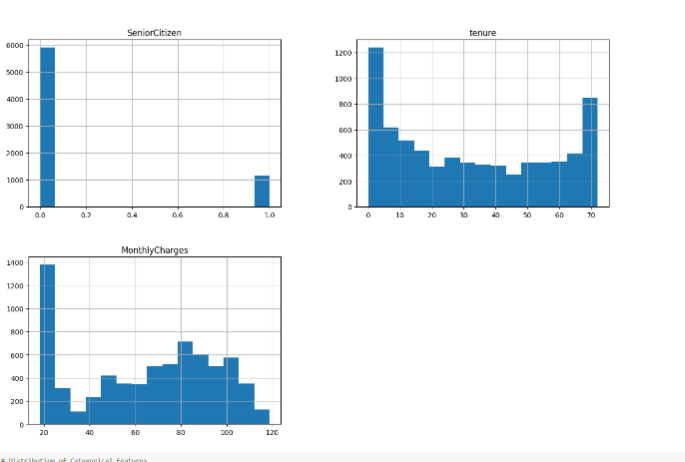
EDA DIAGRAMS:

(In the notebook Question 2)

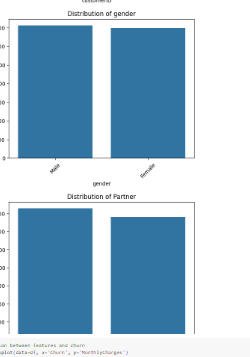
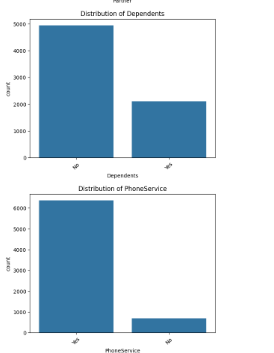
Distribution diagram: (target)

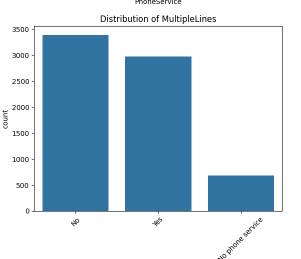
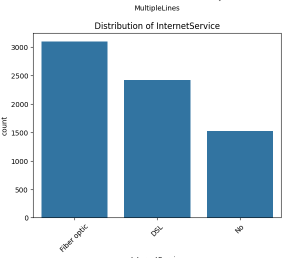


Numeric distributions:

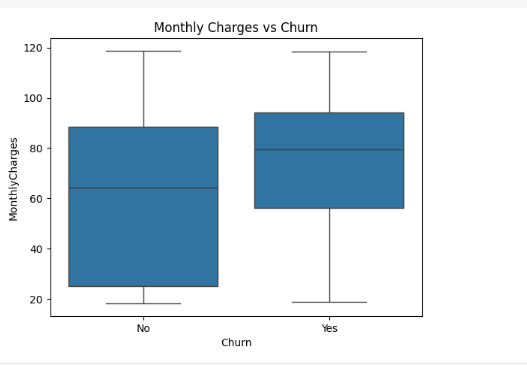


Categorial distribution: there are 18 diagram (look at the notebook). Here are some examples.

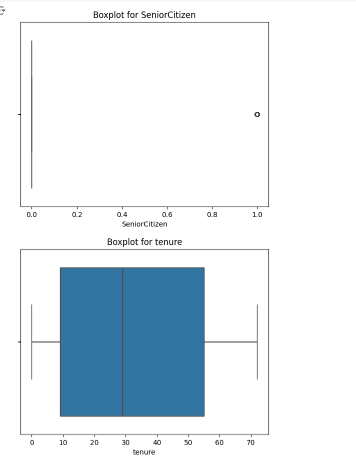


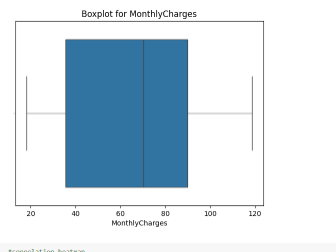


Relation between feature and churn



Outliers:





#### 3) Preprocessing:

(In the notebook Question 3)

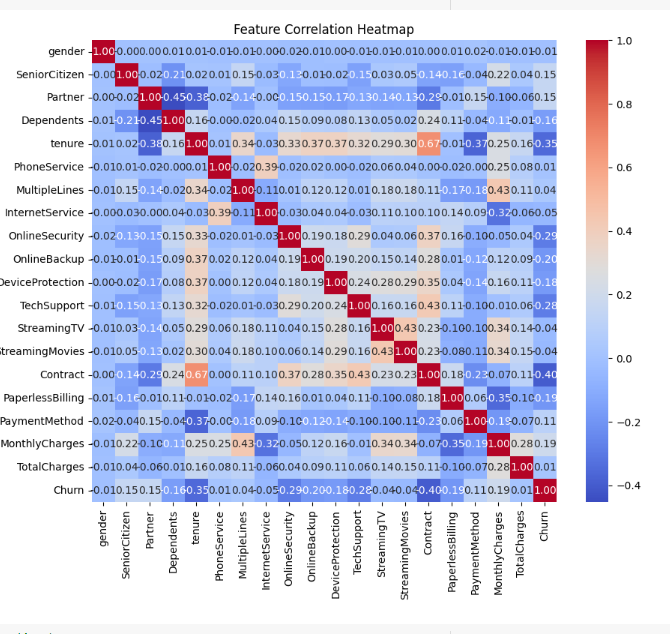
a. Cleaning data-Missing values were imputed using mean or median values, depending on the distribution of the feature.

b. Normalization of data-min max technique

c. Data balancing- Smote algorithm:

d. Extract irrelevant features- like customer id

e. Building matrix correlation (heatmap) and extract features.The code **removes columns** from the DataFrame (df) that have a **low correlation** (absolute correlation < 0.1) with the target variable Churn. It ensures that only features with meaningful linear relationships to Churn are kept for further analysis or modeling.

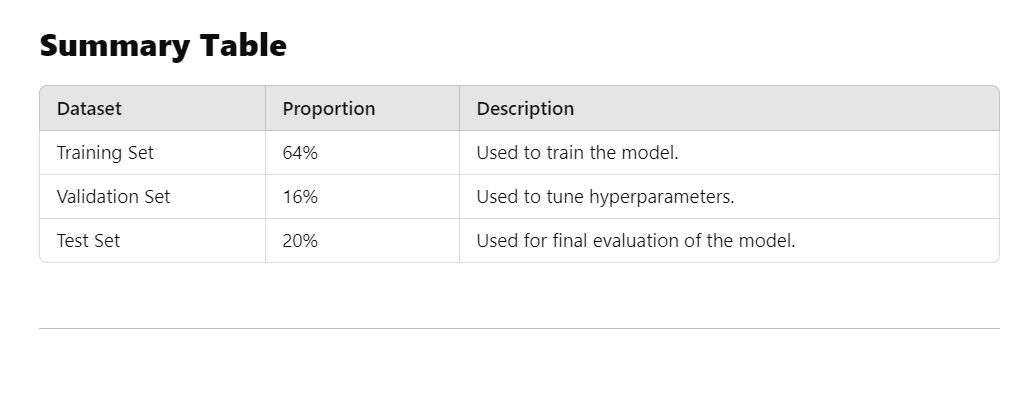


#### Model Development

#### 4) Splitting

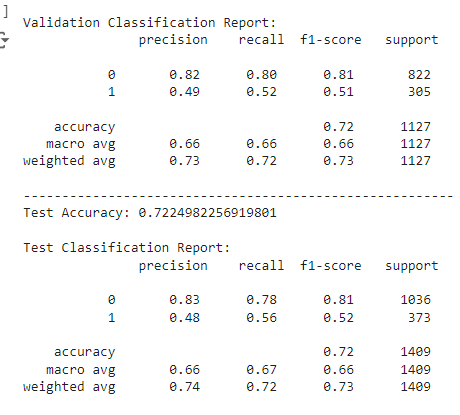
First we split the data to: 80% training and 20% testing.

Then we split the training: 80% to training and 20% to validation:



#### 5) Running decision tree

We run decision tree on this data and got accuracy of 72% on test.( In the notebook Question 5).

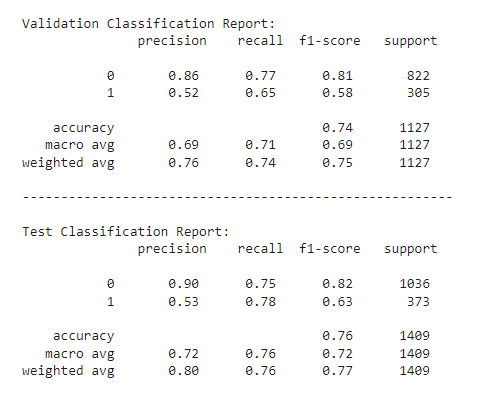


#### 6) Hyper parameters tuning:

Hyper parameters Classic Model

Best Parameters for DecisionTreeClassifier: {'min\_samples\_split': 10, 'min\_samples\_leaf': 4, 'max\_depth': 10} (In notebook Q-6).

The accuracy was improve by 4%.



**Key Observations After Hyperparameter Tuning:**

1. **Improved Accuracy**:
   * The accuracy increased to **74% (validation)** and **76% (test)** compared to the previous results (**72%**).
   * This indicates better performance and generalization.
   * In churn prediction we can also check recall. High recall is crucial because it minimizes FN (missing a churner can leads to lost in revenue.)
2. **Reduced Overfitting**:
   * Training accuracy is high (85%), but validation accuracy is much closer to test results (74%).
   * Hyperparameter tuning reduced overfitting observed in the original tree.

#### 7 ) Neural network

First we build a simple NN. (in the notebook Q-7).

1. The data is fed into the **input layer**.
2. The first hidden layer learns **low-level features** using 128 neurons and ReLU activation.
3. The second hidden layer further processes the features with 64 neurons.
4. The **output layer** outputs a probability score using the sigmoid activation.
5. The model uses **binary cross-entropy loss** to adjust weights and minimize prediction errors during training.
6. Predictions are later classified using a **threshold** (you set it to 0.4):
   * Probability > 0.4 → Class 1 (Positive).
   * Probability ≤ 0.4 → Class 0 (Negative).

num of epoch is 1 and batchsize-32,learning rate=0.001

We get accuracy of about 73% in test -the decision tree has better accuracy when we do it with hyperparameters tuning.

:

#### 8 ) Decision tree VS Neural network

The **Decision Tree** classifier (after hyperparameter tuning) outperformed this simple **Neural Network (NN)** a little. Here’s an explanation of why this can happened:

 **Structured Data**: Decision Trees are more suited to our dataset type.

 **Class Imbalance**: Trees handle imbalance better after SMOTE and hyperparameter tuning.

 **Training Data Size**: NNs need more data to perform well.

 **Optimization**: Our Decision Tree hyperparameters were better optimized.

 **Complexity**: Decision Trees are simpler and less prone to underfitting/overfitting on small datasets.

#### 9 ) Neural networks hyperparameters tuning

We choose :

a) basic NN variations - change regularization by variations of drop rates: drop\_rate in [0.1, 0.3, 0.5]

Conclusion : drop rates 0.3 and 0.5 improved slightly the test accuracy of the netwotk by 1%-2%.

b) basic NN variations - change first dense unites:

layer\_unites in [64, 128, 256]

Conclusion:in first layer units 128 and 256 give the best results

And improve acuuracy by 2%.

c) basic NN variations - change learning\_rate:

lr in [0.01, 0.001, 0.0001]: 0.0001 gives the best result

if we do grid search and choose the best version by hyper parameters:

 'learning\_rate': [0.01, 0.001, 0.0001],

    'batch\_size': [16, 32, 64],

    'epochs': [10, 20, 30]

We receive best params:

Learning Rate: 0.001, Batch Size: 16, Epochs: 30

(with one dropout of 0.3) and accuracy of test was improved by around 2%. Accuracy is 75%(see Q-9 on notebook)

#### 10 ) Increase accuracy by changing the data

Method 1:

We remove outlier and increase accuracy (by 3%), accuracy is 78%.

This code incorporates **outlier detection with Isolation Forest** to clean the training dataset.

In our code, **Isolation Forest** is used separately on each class (Churn = 0 and Churn = 1) to detect and remove **outliers** from the training data. It assumes **20% of the data points** are anomalies (contamination=0.2) and flags them based on how quickly they can be isolated in decision trees. Outliers are removed (prediction == -1), and only **inliers (prediction == 1)** are kept for both classes. The cleaned datasets are then **combined** and used for training the neural network, improving model performance by reducing noise and preventing overfitting. This approach ensures **fair and balanced outlier detection** without bias toward the majority class.

After removing outliers we run our deep learning network with best params ( best learning rate, best batch size and best epoch and one dropout of 0.3)and the network accuracy was improve by 3%. (see in notebook Q-10a).

Method 2:

We changed a bit the architecture of the network and also increase data by 50% (from about 7000 raws to about 10,000 raws) . we so improvement of accuracy (almost 4%- acuuracy is 78.28%). (see in notebook Q-10b and 10c).

#### 11). Reduce accuracy by changing the data

We remove data-focusing only on raws on churn=1. The accuracy was drop drastically (by 50%,also with best params). (see Q-11 in notebook).

The data training heavily skewed toward target = 1

 The dataset is **heavily skewed** towards the **minority class (Churn = 1)** by selecting only rows where Churn == 1.

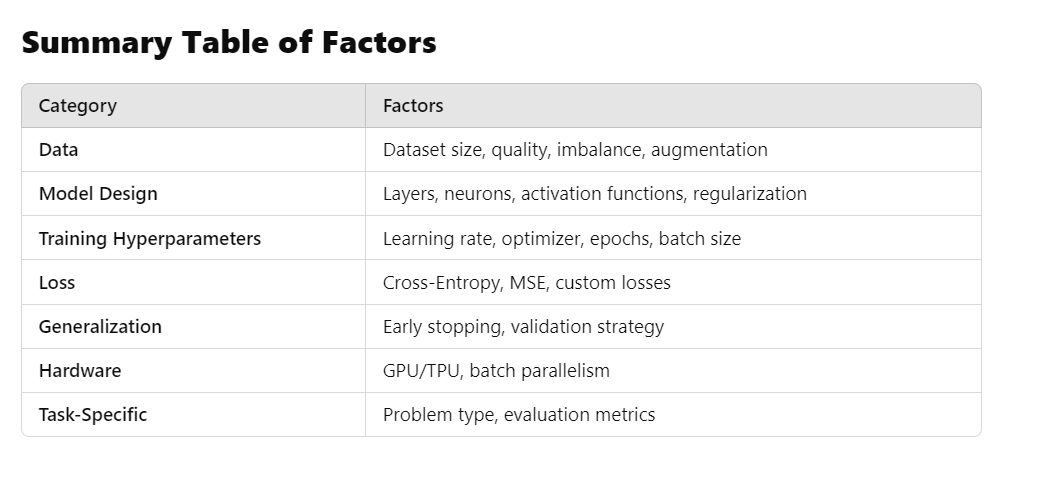
 **Why do this?**

* In highly imbalanced datasets, the minority class (Churn = 1) may not have enough representation for the model to learn effectively.
* By focusing on the minority class, the model gets exposed to more positive examples and reduces **bias toward the majority class**.

The neural network was trained **only on the minority class (Churn=1)**, aiming to improve its ability to detect churn cases. While the model learned patterns specific to churned customers, it **never encountered non-churn (Churn=0) examples** during training. As a result, during evaluation on a balanced dataset, the model became **biased towards predicting Churn=1**, leading to poor **accuracy** and low **precision**, despite possibly high **recall**. This approach caused the model to **overfit to the minority class patterns** and struggle with generalization. To address this, techniques like **SMOTE (Synthetic Minority Oversampling Technique)**, **class weights adjustment**, and **better evaluation metrics (e.g., Precision, Recall, F1-Score, ROC-AUC)** should be applied for balanced training and fair performance evaluation.

#### 12) Changing architecture

Designing a **deep learning architecture** involves several factors that influence model performance, training, and generalization. These factors can be categorized into **hyperparameters**, **model design choices**, and **data-related considerations**.

Below is a comprehensive list of factors: 

Suggest a architectural change-option 1

(in notebook see Q-12a)

leakyrelu\_model = Sequential([

    Input(shape=(x\_train.shape[1],)),

    Dense(128),

    LeakyReLU(alpha=0.1),

    Dense(64),

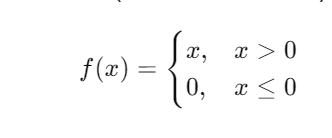
    LeakyReLU(alpha=0.1),

    Dropout(0.3),

    Dense(1, activation='sigmoid')

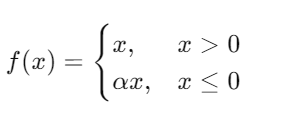
We add LeakyReLU layer. The **Problem with ReLU:** In a standard **ReLU (Rectified Linear**

**Unit)** activation function:



If a neuron's output becomes zero (or negative) during training, its gradient becomes zero, and it may **"die"** — meaning it stops learning.

Solution by LeakyReLU



When the input is negative, instead of returning zero, it returns a **small negative value** scaled by a factor α (e.g., 0.1).

This ensures the neuron can still **learn and update its weights** even when the output is negative.

Key Characteristics of LeakyReLU:

 **Non-Linearity:** Allows the network to learn complex mappings.

 **Small Negative Slope (α):** Prevents neurons from getting "stuck" during training.

Then we run our deep learning network with best params ( best learning rate, best batch size and best epoch and one dropout of 0.3)and the network accuracy was not improved so much.

We then suggest another architecture:

Suggest a architectural option 2 (in notebook Q -12b)

model = Sequential([

    Input(shape=(x\_train.shape[1],)),

    Dense(128),

    BatchNormalization(),

    LeakyReLU(alpha=0.1),

    Dropout(0.3),

    Dense(64),

    BatchNormalization(),

    LeakyReLU(alpha=0.1),

    Dropout(0.2),

    Dense(32),

    BatchNormalization(),

    LeakyReLU(alpha=0.1),

    Dropout(0.2),

    Dense(1, activation='sigmoid')

])

# Optimizer and Compilation

optimizer = Adam(learning\_rate=1e-3)

model.compile(

    optimizer=optimizer,

    loss='binary\_crossentropy',

    metrics=['accuracy', tf.keras.metrics.AUC(name='auc')]

)

# EarlyStopping Callback

early\_stopping = EarlyStopping(

    monitor='val\_loss',

    patience=10,

    restore\_best\_weights=True

)

And we saw the accuracy was a bit improved ( almost by 2%-accuracy -76.5%)

This architecture comprise of :

 **BatchNormalization:** Improves training stability and speed.

Batch Normalization (BN) is a technique used in neural networks to standardize the inputs of each layer. It normalizes the output of the previous layer before passing it to the next one, which helps stabilize and accelerate the training process.

 **LeakyReLU:** Prevents dying neurons by allowing small gradients for negative values.

 **Dropout:** Reduces overfitting by randomly deactivating neurons.

 **EarlyStopping:** Prevents overtraining and restores optimal weights.

#### 13) Evaluation metrics

We will inspect:

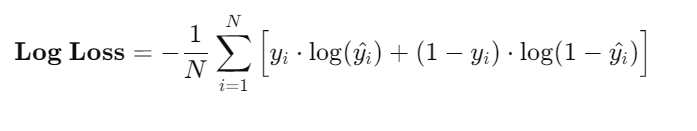
a)The Log Loss:

Log Loss measures the **uncertainty of predictions** made by a classification model. It calculates the **difference between the predicted probability and the actual label**.

 If a prediction is close to the actual label, the log loss will be **low**.

 If a prediction is far from the actual label, the log loss will be **high**.

For **binary classification**:



**Log Loss as a Loss Function:**

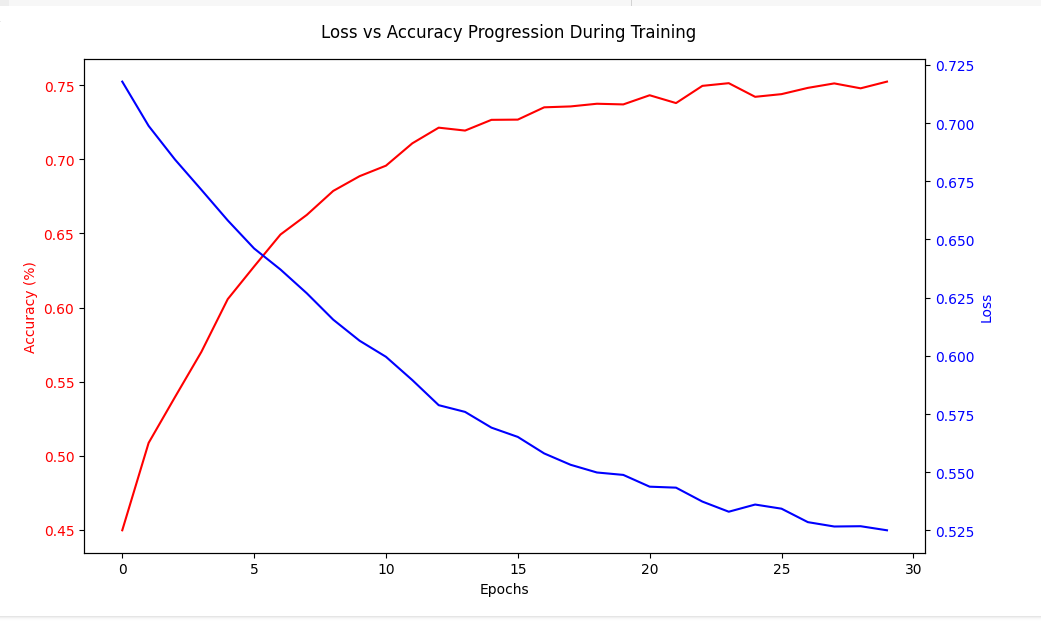
* Used during **training** to optimize the model.

**Log Loss as a Metric:**

* Can also be used to **evaluate the performance** of a trained model on test data.

We run our deep learning model with best hyper parameters (learning rate, batch size and epoch) with loss='binary\_crossentropy

And get this diagram:



The plot displays the progression of **accuracy** and **loss** across **epochs** during the training of a deep learning model. **Accuracy:** Improved consistently, **Loss:** Decreased consistently.

**Overall Model Health:** The model shows good convergence without signs of overfitting or underfitting. (see in notebook Q-13).

b) We also inspect Roc diagram. In question 12 we was trying improving our network accuracy by selecting special architecture. We then run **Roc diagram.**

**(see Q-12b in notebook)**

**ROC-AUC Optimization:** Fine-tunes the threshold for maximum classification performance.

he **ROC (Receiver Operating Characteristic) curve itself is not a single metric**, but rather a **visualization tool** used to evaluate and compare the performance of **binary classification models** across different thresholds.

**AUC (Area Under Curve):** 0.8274

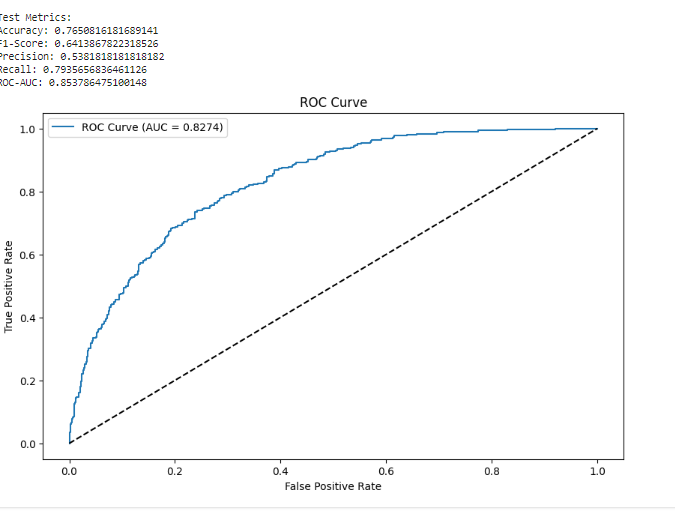
* This indicates **good discrimination ability**. Our model can separate positive and negative classes effectively.

**From our displayed metrics:**

* **Accuracy:** 0.7658 (Decent overall performance)
* **F1-Score:** 0.6414 (Balance between Precision and Recall)
* **Precision:** 0.5381 (Relatively low)
* **Recall:** 0.7936 (High sensitivity)

**Interpretation:**

* Our current threshold favors **Recall** over **Precision**.
* This suggests our model is optimized for scenarios where **missing a positive (false negative) is more costly than falsely predicting a positive (false positive)**.



#### 14) Smote

We run smote with our best params (in notebook Q-14).

smote\_ratios = [0.5, 0.75, 1.0]

**SMOTE (Synthetic Minority Over-sampling Technique)** is a technique used to handle **imbalanced datasets** by **synthetically generating samples for the minority class**. The smote\_ratios parameter determines the **level of oversampling** applied to the minority class.

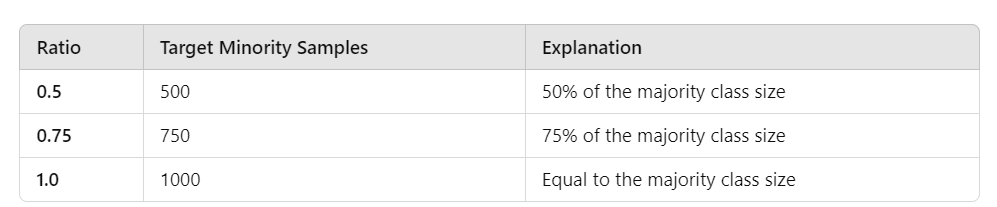
 **Purpose:** Address class imbalance by creating synthetic samples for the minority class.

 **How It Works:** SMOTE generates new samples by interpolating between existing minority class samples.

 **Goal:** Improve model performance, especially in metrics like **Recall** and **F1-Score**.

**Example Scenario:**

* **Original Dataset:**
  + Majority Class (Class 0): 1000 samples
  + Minority Class (Class 1): 100 samples



We get the best result with smote 0.5: (increase by 2% accuracy-accuracy is 77%)

Reasons:

Overfittinfg with higher ratio

1.  **Problem:** When the SMOTE ratio is increased to 0.75 or 1.0, the model might have started **overfitting** on the synthetic minority class samples.

 **Why?** Synthetic data generated by SMOTE isn't as diverse or representative as real-world data. Excessive oversampling can create **redundant or unrealistic samples**.

 **Result:** The model performs well on the training data but poorly on unseen data (validation/test sets).

**Training Data Imbalance vs. Real-World Distribution**

* Real-world datasets often have some level of **natural imbalance**.
* A **perfectly balanced dataset (1:1)** might not reflect real-world conditions, leading to **misleading performance** during evaluation.
* A ratio of **0.5** can be more aligned with the natural imbalance in your original dataset.

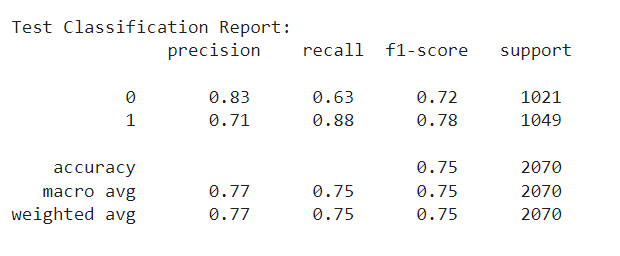
#### 15) Pca

We choose PCA.(in notebook Q-15)

**Principal Component Analysis (PCA)** is a **dimensionality reduction technique** that transforms high-dimensional data into a smaller set of **uncorrelated variables (principal components)** while retaining as much **variance** as possible. It works by identifying the **directions (eigenvectors)** that capture the **maximum variance** in the data and projecting the data onto those directions.

After running PCA on our **basic NN** the accuracy of the network was improved by 2%:

(in the basic NN accuracy was 73%).



#### :

Potential reasons why PCA improve our BasicNeuralNetwork?

## 1. Reduced Overfitting (Curse of Dimensionality)

* **Problem:** In high-dimensional data, models can overfit because they learn patterns specific to noise rather than general trends.
* **How PCA Helps:** PCA reduces dimensions by **removing less informative features** (those with low variance).
* **Result:** The model focuses on **important patterns** rather than noisy or redundant information, improving generalization.

## 2.Noise Reduction

* **Problem:** Raw datasets often contain noise, which can mislead deep learning models.
* **How PCA Helps:** PCA filters out **low-variance features**, which are often associated with noise.
* **Result:** The data becomes cleaner, leading to more **robust and stable training**.

**3.Improved Convergence During Training**

* **Problem:** High-dimensional data increases computational complexity and slows model training.
* **How PCA Helps:** By reducing dimensions, PCA simplifies the data structure.
* **Result:** Training becomes **faster**, and the optimizer converges more efficiently.

**4.Feature Correlation Reduction**

* **Problem:** Highly correlated features can cause instability and poor weight updates.
* **How PCA Helps:** PCA creates **orthogonal principal components**, reducing correlation among features.
* **Result:** The model can **learn more effectively** from independent features.

**5.Highlighting the Most Important Features**

* **Problem:** In raw data, important features might be buried under irrelevant or less significant features.
* **How PCA Helps:** PCA ensures the **principal components** capture the most **significant variance** in the dataset.
* **Result:** The model focuses on the **most informative features**, improving accuracy.

### Results

#### Decision Tree

* Baseline accuracy: 72%.
* Optimized accuracy: 76% (test), 74% (validation).
* Hyperparameter tuning reduced overfitting, as evidenced by the closer alignment of training and validation accuracies.

#### Neural Network

* Baseline accuracy: 73%.
* Best configuration (batch size = 16, learning rate = 0.001, dropout = 0.3) achieved 75% accuracy.

#### Architecture

* Architecture option 2 improved accuracy of our network in 2%.

#### Data change

* By removing outliers the accuracy of the model improve by 3%. Another method was increasing data by 50% -the model accuracy increase by 4%.
* By remove data-focusing only on rows on churn=1. The accuracy was drop drastically (by 50%,also with best params)

#### Smote

* The optimal SMOTE ratio was 0.5, balancing improved recall with minimized overfitting.

#### Pca

* Incorporating PCA improved accuracy by 2%, when running simple neural network underscoring the importance of dimensionality reduction.

#### Metrics

* We use metrics like: accuracy,precision,recall,f1 score and Roc diagram.

### Discussion and Conclusions

This study demonstrates the efficacy of preprocessing and optimization techniques in improving predictive models for customer churn.

-Decision Trees, with hyper parameters tuning and Neural Networks with hyper params tuning performance is almost the same .Neural networks excel at capturing complex, non linear relationships, if our data lacks such patterns it may perform similarly to decision tree.

-Changing the data (increase /decrease) can increase or decrease performance.

-Some NN architectures outperforms others.

-SMOTE with ratio 0.5 improve performances.

-PCA contributed to better generalization by reducing noise and focusing on relevant features when running it on simple NN.

### Future Work

1. **Advanced Models**: Explore ensemble methods, such as Random Forests and Gradient Boosting, for enhanced performance.
2. **Deep Learning**: Investigate advanced architectures, such as Long Short-Term Memory (LSTM) networks, to capture temporal patterns.
3. **Feature Engineering**: Develop domain-specific features to provide additional insights.
4. **Real-World Applications**: Test models on larger, diverse datasets to ensure scalability.
5. **Cost-Sensitive Learning**: Incorporate business impacts into model evaluation and optimization.

### References

1. Blastchar. Telco Customer Churn Dataset. Kaggle. Retrieved from https://www.kaggle.com/blastchar/telco-customer-churn.
2. He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284.
3. Hinton, G., Srivastava, N., & Krizhevsky, A. (2012). Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. arXiv preprint arXiv:1207.0580.
4. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.