

Telco Customer Churn Project

Report

Contents

Executive Summary.....	4
1. Introduction.....	5
2. Corpus Preparation	5
2.1. Dataset Source	5
2.2. Dataset Structure	5
2.3. Data cleaning	5
2.4. Data Preprocessing	5
3. Exploratory Data Analysis.....	6
3.1. Missing Values.....	6
3.2. Churn Distribution.....	6
3.3. Numerical Feature Insights	6
3.4. Categorical Feature Insights	6
3.5. Correlation Analysis	6
4. Solution Methodology	7
4.1. Preprocessing.....	7
4.2. Class Imbalance Handling	7
4.3. Models Implemented.....	7
4.3.1. Decision Tree Classifier	7
4.3.2. Neural Network	7
5. Evaluation Criteria	8
6. Experimental Results.....	8
6.1. Decision Tree Test Results.....	8
6.2. Neural network.....	8
7. Model Comparison	9
8. Ethical Considerations	9
8.1. Bias and Fairness.....	9
8.2. Transparency	10
8.3. Privacy considerations.....	10
8.4. Responsible Model Use	10
9. Post-Deployment Strategy.....	10
9.1. Monitoring.....	10
9.2. Explainability.....	10
9.3. Updating and Recalibration.....	10
9.4. Ethical Audit.....	10

9.5. Human Oversight	11
10. Limitations.....	11
11. Future Enhancements.....	11
12. Tools, Libraries and Frameworks Used.....	11
13. GitHub Repository	11
14. Appendix	11

Executive Summary

This project predicts customer churn using the Telco Customer Churn dataset, aiming to identify customers likely to leave a telecommunications service. After preparing and cleaning the dataset an exploratory data analysis (EDA) was conducted, revealing key churn indicators such as short customer tenure, higher monthly charges and month-to-month contracts.

The data was preprocessed using numeric scaling and one-hot encoding before training two models: a Decision Tree classifier and a Neural network. Both models were tuned and evaluated using accuracy, precision, recall, F1-score, ROC-AUC and PR-AUC. Experimental results show that the neural network provides stronger predictive performance, while the decision tree offers easier interpretability.

Ethical considerations including bias reduction, fairness, transparency and responsible deployment were applied throughout model development. A post-deployment strategy is proposed, covering monitoring, retraining, drift detection and explainability.

The report concludes with limitations and recommendations for future improvement, such as exploring ensemble approaches and enhancing interpretability tools.

1. Introduction

Customer churn represents a critical business challenge for subscription-based companies such as telecommunications providers. Predicting whether a customer is likely to discontinue their service enables companies to improve retention and reduce revenue loss.

This project develops two machine learning models, a Decision Tree classifier and a Neural Network model to perform binary classification on the Telco Customer Churn dataset (from Kaggle). The project pipeline includes dataset preparation, exploratory data analysis (EDA), preprocessing, model development with hyperparameter tuning, evaluation, comparison, ethical considerations and post-deployment strategies.

2. Corpus Preparation

2.1. Dataset Source

The dataset used is the Telco Customer Churn dataset, publicly available on Kaggle. It contains customer-level information from a telecommunications company with the objective of predicting whether a customer will churn.

- Telco Customer Churn dataset (Kaggle): <https://www.kaggle.com/datasets/blastchar/telco-customer-churn>

2.2. Dataset Structure

- Total samples: 7043
- Total attributes: 21 columns
- Target variable: Churn (Yes/No)
- Attribute types:
 - Numerical: tenure, Monthlycharges, TotalCharges (becomes numerical after cleaning)
 - Categorical: contract type, payment method, service subscriptions, etc.

2.3. Data cleaning

The following preprocessing steps were applied:

- Conversion of TotalCharges to numeric and removed rows where conversion produced missing values.
- Dropping the identifier customerID since it had no predictive value
- Mapping Churn from Yes/No to 1/0

2.4. Data Preprocessing

- Applied ColumnTransformer:
 - Numerical features – StandardScaler

- Categorical features – OneHotEncoder
- Split data into 80% training and 20% testing using stratified sampling to preserve class imbalance.
- Computed class weights to address imbalance, ensuring fairer model performance.
- Prepared final feature matrices for use in Scikit-Learn and Keras models.

3. Exploratory Data Analysis

3.1. Missing Values

- Only TotalCharges contained missing values.
- After conversion to numeric, 11 rows had NaN values and they were removed.

3.2. Churn Distribution

- Churn class imbalance observed:
 - No: 73.5%
 - Yes: 26.5%
- Indicates the need for class balancing techniques which are addressed later via class weights.

3.3. Numerical Feature Insights

Boxplots for tenure, MonthlyCharges and TotalCharges reveal:

- Churned customers typically have lower tenure.
- MonthlyCharges show a slight upward trend for churners.
- TotalCharges are lower for churners due to shorter tenure.

3.4. Categorical Feature Insights

Patterns observed:

- Month-to-month contracts have significantly higher churn than one-year or two-year contracts.
- Electronic check users tend to churn more.
- Fiber optic internet customers shower churn probabilities.

3.5. Correlation Analysis

- Weak correlations between numerical features

- Tenure and TotalCharges strongly correlated.

4. Solution Methodology

4.1. Preprocessing

Performed using ColumnTransformer:

- Numeric features – StandardScaler
- Categorical features – OneHotEncoder
- Train-test split: 80/20, stratified to preserve class imbalance.

4.2. Class Imbalance Handling

Used:

- `compute_class_weight(class_weight='balanced')`

This adjusts training weights so minority class receives more importance.

4.3. Models Implemented

4.3.1. Decision Tree Classifier

Used withing Scikit-Learn pipeline.

Hyperparameter tuning using GridSearchCV:

- criterion = gini/entropy
- max_depth = None, 5, 8, 12
- min_samples_split
- min_samples_leaf

4.3.2. Neural Network

- Architecture:
 - Dense(128, ReLU) + Dropout
 - Dense(64, ReLU) + Dropout
 - Dense(32, ReLU)
 - Dense (1, sigmoid)
- Optimizer: Adam
- Loss: Binary crossentropy

- Metrics: accuracy
- EarlyStopping with patience=8 to prevent overfitting.

5. Evaluation Criteria

Models evaluated using:

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC
- PR-AUC
- Confusion Matrix

Neural networks and decision trees are compared using ROC-AUC and F1 which are more meaningful under imbalance.

6. Experimental Results

6.1. Decision Tree Test Results

Classification report summary:

Metric	Score
Accuracy	0.753
Precision	0.525
Recall	0.743
F1	0.615
ROC AUC	0.829
PR AUC	0.608

Confusion matrix and ROC/PR curves generated.

6.2. Neural network

Classification report summary:

Metric	Score

Accuracy	0.713
Precision	0.477
Recall	0.834
F1	0.607
ROC AUC	0.836
PR AUC	0.629

Also includes training curves (loss and accuracy).

7. Model Comparison

Metric	Decision Tree	Neural Network
Accuracy	0.753	0.713
Precision	0.525	0.477
Recall	0.743	0.834
F1	0.615	0.607
ROC-AUC	0.829	0.836
PR-AUC	0.608	0.629

Overall Analysis

- The Neural Network generally performs better on recall and ROC-AUC due to ability to model complex relationships.
- Decision tree is more interpretable but prone to overfitting.

8. Ethical Considerations

8.1. Bias and Fairness

- Ensure no sensitive attributes like race or gender were included.
- Used class weighting to avoid disadvantaging the minority class.
- Avoided models that make opaque decisions without proper documentation.

8.2. Transparency

- Decision Trees are chosen partly for interpretability.
- Clear documentation of preprocessing and modelling steps.

8.3. Privacy considerations

- No personally identifiable information was used.
- customerID removed.

8.4. Responsible Model Use

- The model should not be used to make punitive decisions against customers.
- Predictions should be used to offer retention support, not deny service.

9. Post-Deployment Strategy

9.1. Monitoring

- Track model drift in:
 - Input data distribution
 - Model performance (F1, ROC-AUC)
- Retrain model periodically as customer behavior changes.

9.2. Explainability

- Use SHAP or LIME for interpreting predictions in real-world deployment.

9.3. Updating and Recalibration

Refit models every 3 – 6 months.

Re-tune hyperparameters regularly.

9.4. Ethical Audit

Check for unintended bias forming over time.

9.5. Human Oversight

- Combine model predictions with human review for high-impact cases

10. Limitations

- Class imbalance still present despite weighting.
- Decision tree is prone to overfitting.
- Neural network requires more computational resources.
- Some important customer behavioral patterns may not be captured without time-series data.

11. Future Enhancements

- Try more advanced models: Random Forest, XGBoost, LightGBM
- Use SMOTE or ADASYN for oversampling.
- Hyperparameter tuning of neural network using Optuna/KerasTuner.
- Deployment-ready API with FastAPI.
- Improve explainability with SHAP summary plots.

12. Tools, Libraries and Frameworks Used

- Python
- Pandas, NumPy
- Scikit-Learn
- TensorFlow/Keras
- Matplotlib, Seaborn
- GridSearchCV, StratifiedKFold
- Git for version control

13. GitHub Repository

Github Repository Link: https://github.com/GangaminiH/Telco_Churn_Project.git

14. Appendix

Source code:

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
confusion_matrix, classification_report, roc_curve, precision_recall_curve, average_precision_score)
from sklearn.utils.class_weight import compute_class_weight

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

#Reproducibility
random_seed = 42
np.random.seed(random_seed)
random.seed(random_seed)
tf.random.set_seed(random_seed)

#Load data
df = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn.csv")

print("Dataset shape:", df.shape)
display(df.head())

df.info()

# Check for missing values
df.isnull().sum()

print("\nColumn types:")
display(df.dtypes.value_counts())

print("\nChurn distribution (counts):")
display(df['Churn'].value_counts())

print("\nChurn distribution (proportions):")
display(df['Churn'].value_counts(normalize=True))

#Data cleaning
# Convert TotalCharges to numeric
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Inspect rows with missing TotalCharges
```

```

missing_tc = df[df['TotalCharges'].isna()]
print("Rows with missing TotalCharges:", missing_tc.shape[0])
if not missing_tc.empty:
    display(missing_tc.head())

#Drop rows with missing TotalCharges
df = df[df['TotalCharges'].notna()].copy()
df.reset_index(drop=True, inplace=True)

# Drop identifier
if 'customerID' in df.columns:
    df.drop('customerID', axis=1, inplace=True)

# Map target to numeric
df['Churn'] = df['Churn'].map({'No': 0, 'Yes': 1})

print("After cleaning shape:", df.shape)

#Visualizations
#Churn countplot
plt.figure(figsize=(6,4))
sns.countplot(x='Churn', data=df)
plt.title('Churn distribution (0 = No, 1 = Yes)')
plt.show()

#Boxplots for numeric features vs churn
num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
fig, axes = plt.subplots(1, len(num_cols), figsize=(5*len(num_cols), 4))
for ax, col in zip(axes, num_cols):
    sns.boxplot(x='Churn', y=col, data=df, ax=ax)
    ax.set_title(f'{col} vs Churn')
plt.tight_layout()
plt.show()

#Correlation heatmap for numeric features
plt.figure(figsize=(6,5))
sns.heatmap(df[num_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Numeric feature correlations')
plt.show()

#Data Preprocessing

# Identify numeric and categorical columns
numeric_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
categorical_features = [c for c in df.columns if c not in numeric_features + ['Churn']]

print("Numeric features:", numeric_features)
print("Categorical features (sample):", categorical_features[:10])

# ColumnTransformer: scale numeric, one-hot encode categorical (ignore unknown categories)
preprocessor = ColumnTransformer(
    transformers=[]
)

```

```

        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)
    ],
    remainder='drop' # drop any other columns
)

# Prepare X and y
X = df.drop('Churn', axis=1)
y = df['Churn']

#Train-test split and preprocessing fit

X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=random_seed, stratify=y
)

# Fit preprocessor on training data and transform both train and test
X_train = preprocessor.fit_transform(X_train_raw)
X_test = preprocessor.transform(X_test_raw)

print("Preprocessed shapes -> X_train:", X_train.shape, "X_test:", X_test.shape)

#Class weights for imbalance handling
classes = np.unique(y_train)
class_weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train)
class_weight_dict = {int(c): w for c, w in zip(classes, class_weights)}
print("Class weights:", class_weight_dict)

#Decision Tree

#Building a pipeline
dt_pipeline = Pipeline([
    ('preproc', preprocessor),
    ('clf', DecisionTreeClassifier(random_state=random_seed, class_weight='balanced'))
])
param_grid = {
    'clf_criterion': ['gini', 'entropy'],
    'clf_max_depth': [None, 5, 8, 12],
    'clf_min_samples_split': [2, 5, 10],
    'clf_min_samples_leaf': [1, 2, 4]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
grid = GridSearchCV(dt_pipeline, param_grid, scoring='f1', cv=cv, n_jobs=-1, verbose=1)
grid.fit(X_train_raw, y_train) #pipeline will fit preprocessor internally

best_dt_pipeline = grid.best_estimator_
print("Best Decision Tree parameters:", grid.best_params_)

#Evaluate on test set
y_pred_dt = best_dt_pipeline.predict(X_test_raw)
y_proba_dt = best_dt_pipeline.predict_proba(X_test_raw)[:, 1]

```

```

print("\nDecision Tree (tuned) classification report:")
print(classification_report(y_test, y_pred_dt, digits=4))
print("ROC AUC:", roc_auc_score(y_test, y_proba_dt))

#Neural Network

#Prepare arrays for Keras
X_train_nn = X_train # from earlier preprocessor.fit_transform
X_test_nn = X_test

input_dim = X_train_nn.shape[1]

def build_nn(input_dim, lr=1e-3, dropout_rates=(0.3, 0.2)):
    model = Sequential([
        tf.keras.Input(shape=(input_dim,)),
        Dense(128, activation='relu'),
        Dropout(dropout_rates[0]),
        Dense(64, activation='relu'),
        Dropout(dropout_rates[1]),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model

nn_model = build_nn(input_dim, lr=1e-3)
nn_model.summary()

#Callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True, verbose=1)
]

history = nn_model.fit(
    X_train_nn, y_train,
    validation_split=0.15,
    epochs=100,
    batch_size=64,
    class_weight=class_weight_dict,
    callbacks=callbacks,
    verbose=2
)

#Predictions
y_proba_nn = nn_model.predict(X_test_nn).ravel()
y_pred_nn = (y_proba_nn >= 0.5).astype(int)

#Evaluation: metrics, confusion matrices, ROC & PR curves

```

```

def print_metrics(y_true, y_pred, y_proba=None, label='Model'):
    print(label)
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1:", f1_score(y_true, y_pred))
    if y_proba is not None:
        print("ROC AUC:", roc_auc_score(y_true, y_proba))
        print("PR AUC:", average_precision_score(y_true, y_proba))
    print(classification_report(y_true, y_pred, digits=4))

# Decision Tree metrics
print_metrics(y_test, y_pred_dt, y_proba_dt, label='Decision Tree (tuned)')

# Neural Network metrics
print_metrics(y_test, y_pred_nn, y_proba_nn, label='Neural Network')

# Confusion matrices
def plot_confusion(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(title)
    plt.show()

plot_confusion(y_test, y_pred_dt, 'Decision Tree Confusion Matrix')
plot_confusion(y_test, y_pred_nn, 'Neural Network Confusion Matrix')

# ROC curves
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_proba_dt)
fpr_nn, tpr_nn, _ = roc_curve(y_test, y_proba_nn)

plt.figure(figsize=(8,6))
plt.plot(fpr_dt, tpr_dt, label=f'DT (AUC={roc_auc_score(y_test, y_proba_dt):.3f})')
plt.plot(fpr_nn, tpr_nn, label=f'NN (AUC={roc_auc_score(y_test, y_proba_nn):.3f})')
plt.plot([0,1],[0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Decision Tree vs Neural Network)')
plt.legend()
plt.show()

# Precision-Recall curves
prec_dt, rec_dt, _ = precision_recall_curve(y_test, y_proba_dt)
prec_nn, rec_nn, _ = precision_recall_curve(y_test, y_proba_nn)
ap_dt = average_precision_score(y_test, y_proba_dt)
ap_nn = average_precision_score(y_test, y_proba_nn)

plt.figure(figsize=(8,6))
plt.plot(rec_dt, prec_dt, label=f'DT (AP={ap_dt:.3f})')
plt.plot(rec_nn, prec_nn, label=f'NN (AP={ap_nn:.3f})')

```

```

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Decision Tree vs Neural Network)')
plt.legend()
plt.show()

#Plot NN training history

hist = history.history

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(hist['accuracy'], label='Train Accuracy')
plt.plot(hist['val_accuracy'], label='Validation Accuracy')
plt.title('Neural Network Accuracy (Train vs Validation)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(hist['loss'], label='Train Loss')
plt.plot(hist['val_loss'], label='Validation Loss')
plt.title('Neural Network Loss (Train vs Validation)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.tight_layout()
plt.show()

```