Contents

1 Introduction

Customer churn refers to the phenomenon in which customers discontinue a service. In the highly competitive telecom industry, predicting churn in advance enables service providers to take proactive retention actions. Machine learning provides an effective framework for learning churn patterns from historical customer behavior data.

This project focuses on the academic study, experimentation, and analysis of an existing telecom customer churn prediction system. The project was explored by modifying models, preprocessing pipelines, and evaluation strategies to observe performance changes and draw conclusions aligned with machine learning concepts covered in the course. The dataset and problem domain are credited to **Telecom Egypt (WE)**.

2 Learning Framework (Task–Experience–Performance)

The supervised learning formulation of the problem is defined as follows:

- **Task (T):** Binary classification to predict whether a customer will churn.
- **Experience (E):** Historical labeled telecom customer data.
- **Performance (P):** Accuracy, Precision, Recall, F1-score, and ROC-AUC on a held-out test set.

This formulation anchors the project in formal supervised learning theory.

3 Implementation Environment and Libraries

The project is implemented in Python using standard data science and machine learning libraries, primarily from the `scikit-learn` ecosystem.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    ConfusionMatrixDisplay,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans

import joblib

plt.style.use("seaborn-v0_8")
pd.set_option("display.max_columns", None)
RANDOM_STATE = 42
```

Figure 1: Imported libraries and project configuration.

4 Data Loading and Inspection

The dataset is loaded from a CSV file, with date fields parsed explicitly to support time-aware features. Initial inspection verifies the dataset structure and feature types.

Load and Inspect Data

```python
# Load data
df = pd.read_csv("telecom_churn_full.csv", parse_dates=["signup_date","last_activity_ts"])
```

Python

```python
# 1) Basic info
print("Shape:", df.shape)
print("Data Types:")
print(df.dtypes)
```

Python

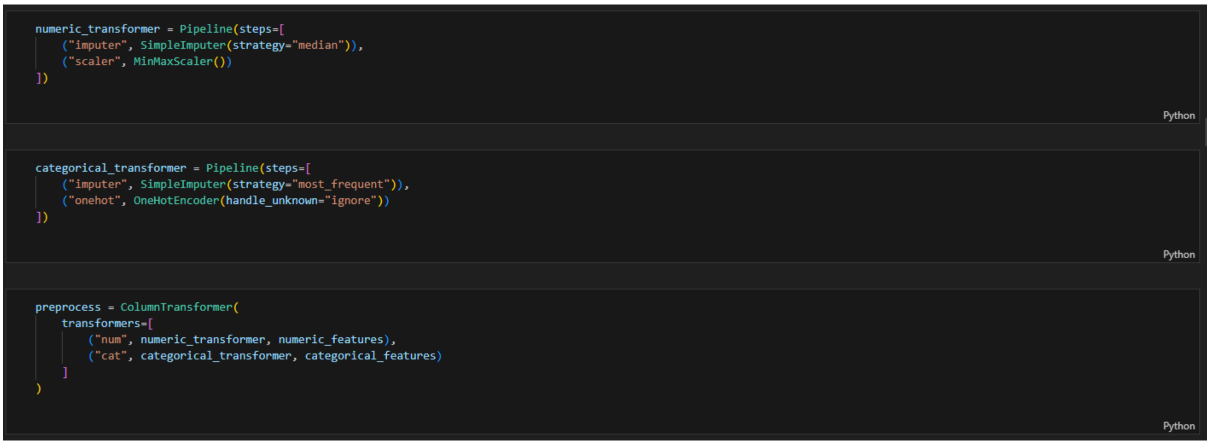
Figure 2: Dataset loading and basic inspection.

5 Preprocessing Pipeline

Separate preprocessing pipelines are defined for numerical and categorical features using a `ColumnTransformer`.

5.1 Numerical Features

Numerical features are imputed using the median and normalized using Min–Max scaling.



```python
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", MinMaxScaler())
])
```

```python
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])
```

```python
preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
)
```


Figure 3: Numerical preprocessing pipeline.

5.2 Categorical Features

Categorical features are imputed using the most frequent value and encoded using one-hot encoding.

```python
models = {
    "log_reg_baseline": LogisticRegression(max_iter=800, class_weight="balanced", n_jobs=-1, solver="lbfgs"),
    "decision_tree": DecisionTreeClassifier(
        max_depth=8, min_samples_leaf=10, class_weight="balanced", random_state=RANDOM_STATE
    ),
    "random_forest": RandomForestClassifier(
        n_estimators=350, random_state=RANDOM_STATE, class_weight="balanced_subsample"
    ),
    "knn": KNeighborsClassifier(n_neighbors=7, weights="distance"),
    "svm_rbf": SVC(
        probability=True,
        kernel="rbf",
        class_weight="balanced",
        C=1.5,
        gamma="scale",
        random_state=RANDOM_STATE,
    ),
    "mlp": MLPClassifier(
        hidden_layer_sizes=(128, 64),
        activation="relu",
        solver="adam",
        alpha=1e-4,
        learning_rate="adaptive",
        max_iter=400,
        early_stopping=True,
        random_state=RANDOM_STATE,
    ),
}

display_name_map = {
    "log_reg_baseline": "Logistic Regression (Baseline)",
    "decision_tree": "Decision Tree Classifier",
    "random_forest": "Random Forest Classifier",
    "knn": "K-Nearest Neighbors (KNN)",
    "svm_rbf": "Support Vector Machine (RBF Kernel)",
    "mlp": "Neural Network (Multilayer Perceptron)",
}
```

Figure 4: Categorical preprocessing pipeline.

5.3 Combined Preprocessing

```python
metrics_records = []
pipelines = {}

for name, model in models.items():
    clf = Pipeline(steps=[("preprocess", preprocess), ("model", model)])
    display_name = display_name_map.get(name, name)
    # 5-fold CV Balanced Accuracy
    bal_acc = cross_val_score(clf, X_train, y_train, cv=5, scoring="balanced_accuracy")
    print(f"{display_name} 5-fold CV Balanced Accuracy: {bal_acc.mean():.3f} +/- {bal_acc.std():.3f}")
    # Fit full train
    clf.fit(X_train, y_train)
    pipelines[name] = clf

    # Held-out test predictions
    y_pred = clf.predict(X_test)
    y_proba = clf.predict_proba(X_test)[:, 1]

    metrics_records.append({
        "model_key": name,
        "model": display_name,
        "accuracy": accuracy_score(y_test, y_pred),
        "precision": precision_score(y_test, y_pred, zero_division=0),
        "recall": recall_score(y_test, y_pred, zero_division=0),
        "f1": f1_score(y_test, y_pred, zero_division=0),
        "roc_auc": roc_auc_score(y_test, y_proba),
    })

    print(classification_report(y_test, y_pred, digits=3))

    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots()
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
    disp.plot(cmap="Blues", ax=ax)
    ax.set_title(f"Confusion Matrix - {display_name}")
    plt.show()

metrics_df = pd.DataFrame(metrics_records).set_index("model").sort_values("roc_auc", ascending=False)
display(metrics_df)
```

Figure 5: ColumnTransformer combining numerical and categorical pipelines.

Min–Max normalization is required for distance-based and margin-based models, while tree-based models remain scale-invariant.

6 Supervised Learning Models

Multiple supervised classifiers were studied to reflect course coverage.

```python
# Logistic Regression churn probabilities (sample)
log_reg_probs = pipelines["log_reg_baseline"].predict_proba(X_test)[:, 1]
prob_sample = pd.DataFrame({
    "true_churn": y_test.reset_index(drop=True),
    "log_reg_probability": log_reg_probs
})
display(prob_sample.head(10))
```

Figure 6: Defined supervised learning models and display names.

The models include Logistic Regression (baseline), Decision Tree, Random Forest, K-Nearest Neighbors, Support Vector Machine (RBF kernel), and a Neural Network (MLP).

7 Model Training and Evaluation

Each model is trained within a unified pipeline that includes preprocessing. Five-fold cross-validation using balanced accuracy is performed, followed by evaluation on a held-out test set.

Model Evaluation & Selection

```python
sorted_metrics = metrics_df.sort_values("roc_auc", ascending=False)
best_model_display = sorted_metrics.index[0]
best_model_key = sorted_metrics["model_key"].iloc[0]
best_auc = sorted_metrics.iloc[0]["roc_auc"]
print(f"Best model by ROC-AUC: {best_model_display} ({best_auc:.3f})")

best_model = pipelines[best_model_key]
```

Python

```
Best model by ROC-AUC: log_reg_baseline (0.735)
```

```python
# K-Means clustering for customer segmentation
kmeans_pipeline = Pipeline(steps=[
    ("preprocess", preprocess),
    ("kmeans", KMeans(n_clusters=3, n_init=20, random_state=RANDOM_STATE)),
])

cluster_labels = kmeans_pipeline.fit_predict(X)

cluster_df = df_new.copy()
cluster_df["cluster"] = cluster_labels
cluster_churn = cluster_df.groupby("cluster")["churn"].agg(["count", "mean"])
cluster_churn = cluster_churn.rename(columns={"mean": "churn_rate"})

display(cluster_churn)
```

Python

Figure 7: Model training, cross-validation, and evaluation pipeline.

8 Model Selection

Models are ranked based on ROC-AUC, which serves as the primary selection metric due to its robustness for probabilistic classification and imbalanced datasets.

```python
# Persist pipelines and evaluation splits for deployment
joblib.dump(pipelines, "all_churn_pipelines.pkl")
X_test.to_csv("X_test_churn.csv", index=False)
y_test.to_csv("y_test_churn.csv", index=False)
```

Figure 8: Model ranking and best model selection based on ROC-AUC.

9 Sample Predictions and Model Output Interpretation

Sample predictions from the best-performing model are displayed to illustrate how customer attributes are translated into churn probabilities and final predictions.

Sample predictions (best model)

Showing first 50 customers with true churn label, predicted churn, and churn probability.

plan_type	monthly_data_gb	avg_call_minutes	num_support_tickets_last_3m	num_dropped_calls_last_30d	payment_method	auto_pay	last_bill_amount	is_promo	city	region	device_type	contract_type	data_overage_gb_last_3m	intl_call_minutes_last_3m	com
Basic	20.301735496548933	271.22689322841234	1	1	credit_card	1	132.8454082427372	1	Aswan	UpperEgypt	iOS	month-to-month	0.0	4.374269515887875	0
Premium	12.526367612990326	213.1123379362303	0	2	wallet	1	106.24422549330164	0	Giza	UpperEgypt	Android	month-to-month	0.0	0.0	1
Basic	nan	230.2901565001576	0	1	credit_card	1	110.8853361446646	0	Aswan	Delta	iOS	month-to-month	0.0	34.15470991241391	0
Standard	37.09701783708509	117.35586265318427	1	1	credit_card	1	161.5504549350666	0	Mansoura	Cairo	Android	month-to-month	0.0	28.392056982631694	0
Premium	29.945809623488813	235.15948814482235	1	1	wallet	0	136.61507970183314	0	Tanta	Delta	Android	month-to-month	0.0	35.849156667656665	0
Basic	15.634648497438576	342.4113172955875 4	0	1	wallet	1	127.3944659205061	1	Luxor	UpperEgypt	Android	two-year	5.2910935084662745	69.6184548519424	1
Basic	31.6683979499183	273.9881875339263	1	1	credit_card	0	147.2975932021021	0	Luxor	Delta	Android	two-year	0.0	42.03355667807076	0
Premium	4.439044283060185	130.0822123161352	1	1	credit_card	1	80.48978614172081	0	Giza	Delta	Android	one-year	0.0	13.470407499034645	0
Standard	nan	259.378346533498	0	0	credit_card	1	108.1188758809924	0	Mansoura	Alex	iOS	month-to-month	2.778612596185964	41.36258116290029	1
Standard	13.333772273749103	205.1886683849699	1	1	credit_card	1	100.33022024770762	0	Asyut	Delta	iOS	month-to-month	0.0	44.1692063389076	0
Standard	15.744374481553582	360.4433752618274	0	1	credit_card	1	109.78000599031488	0	Luxor	Giza	iOS	one-year	0.0	61.77569229500296	1
Standard	12.33538990737969	137.78426970962315	0	2	nan	0	87.93234183117633	0	Giza	Delta	Android	month-to-month	6.018321787541528	32.00907552261091	0
Standard	nan	316.20762188507393	0	1	cash	0	121.6982492929063	1	Giza	Alex	Android	one-year	0.0	8.49510107094908	0
Standard	11.586119630253178	299.66379513179186	1	1	credit_card	1	107.88050788669004	0	Mansoura	Giza	Android	month-to-month	5.703555454637012	82.39095665144957	2
Standard	7.915372142159457	77.44840905540397	0	0	credit_card	0	66.7854994470536	1	Aswan	UpperEgypt	Android	one-year	0.0	35.217831822074594	1
Standard	14.992920620732097	238.1012327386234	0	3	credit_card	0	117.9538455998438	0	Luxor	Alex	Android	month-to-month	0.0	0.0	0
Standard	14.265559568353169	436.0729373372251	1	0	credit_card	1	117.6754082594922	0	Alexandria	Delta	iOS	one-year	8.659837370050514	30.0629987645726	1
Standard	28.430911118719333	230.71894049716315	1	2	debit_card	0	135.55244453444857	0	Asyut	Cairo	iOS	one-year	4.984678442624524	22.99957134323449	0
Basic	nan	nan	0	0	wallet	1	122.47226488703588	0	Tanta	UpperEgypt	Android	month-to-month	0.0	0.0	0
Standard	nan	194.0017852448618	1	2	nan	0	80.71037418586582	0	Aswan	Cairo	Android	two-year	7.54276624338297	49.05833918949909	0
Premium	nan	321.0440748650616	0	0	cash	0	125.85111638894274	0	Giza	UpperEgypt	Android	month-to-month	4.048484325648612	22.85554768594265	0
Standard	nan	137.57828422197116	1	0	cash	0	118.19053961689691	0	Mansoura	Delta	iOS	month-to-month	0.0	38.52180955564511	0
Basic	13.294816358313312	332.21838201176865	1	0	wallet	0	118.67212062468386	0	Luxor	Cairo	iOS	month-to-month	0.0	11.812070628827165	0
Basic	35.42843551562986	255.8306783769058	0	2	wallet	0	173.1423108724528	1	Asyut	Giza	iOS	one-year	0.0	25.57138763461409	1
Basic	14.611941575125758	243.1528154711356	2	1	credit_card	1	94.42344333141726	1	Alexandria	Alex	Android	two-year	0.0	18.630874528667647	0

Figure 9: Sample predictions from the best-performing model (part 1).

Sample predictions (best model)

Showing first 50 customers with true churn label, predicted churn, and churn probability.

plan_type	monthly_data_gb	avg_call_minutes	num_support_tickets_last_3m	num_dropped_calls_last_30d	payment_method	auto_pay	last_bill_amount	is_promo	city	region	device_type	contract_type	data_overage_gb_last_3m	intl_call_minutes_last_3m	com
Basic	20.301735496548933	271.22689322841234	1	1	credit_card	1	132.8454082427372	1	Aswan	UpperEgypt	iOS	month-to-month	0.0	4.374269515887875	0
Premium	12.526367612990326	213.1123379362303	0	2	wallet	1	106.24422549330164	0	Giza	UpperEgypt	Android	month-to-month	0.0	0.0	1
Basic	nan	230.2901565001576	0	1	credit_card	1	110.8853361446646	0	Aswan	Delta	iOS	month-to-month	0.0	34.15470991241391	0
Standard	37.09701783708509	117.35586265318427	1	1	credit_card	1	161.5504549350666	0	Mansoura	Cairo	Android	month-to-month	0.0	28.392056982631694	0
Premium	29.945809623488813	235.15948814482235	1	1	wallet	0	136.61507970183314	0	Tanta	Delta	Android	month-to-month	0.0	35.849156667656665	0
Basic	15.634648497438576	342.4113172955875 4	0	1	wallet	1	127.3944659205061	1	Luxor	UpperEgypt	Android	two-year	5.2910935084662745	69.6184548519424	1
Basic	31.6683979499183	273.9881875339263	1	1	credit_card	0	147.2975932021021	0	Luxor	Delta	Android	two-year	0.0	42.03355667807076	0
Premium	4.439044283060185	130.0822123161352	1	1	credit_card	1	80.48978614172081	0	Giza	Delta	Android	one-year	0.0	13.470407499034645	0
Standard	nan	259.378346533498	0	0	credit_card	1	108.1188758809924	0	Mansoura	Alex	iOS	month-to-month	2.778612596185964	41.36258116290029	1
Standard	13.333772273749103	205.1886683849699	1	1	credit_card	1	100.33022024770762	0	Asyut	Delta	iOS	month-to-month	0.0	44.1692063389076	0
Standard	15.744374481553582	360.4433752618274	0	1	credit_card	1	109.78000599031488	0	Luxor	Giza	iOS	one-year	0.0	61.77569229500296	1
Standard	12.33538990737969	137.78426970962315	0	2	nan	0	87.93234183117633	0	Giza	Delta	Android	month-to-month	6.018321787541528	32.00907552261091	0
Standard	nan	316.20762188507393	0	1	cash	0	121.6982492929063	1	Giza	Alex	Android	one-year	0.0	8.49510107094908	0
Standard	11.586119630253178	299.66379513179186	1	1	credit_card	1	107.88050788669004	0	Mansoura	Giza	Android	month-to-month	5.703555454637012	82.39095665144957	2
Standard	7.915372142159457	77.44840905540397	0	0	credit_card	0	66.7854994470536	1	Aswan	UpperEgypt	Android	one-year	0.0	35.217831822074594	1
Standard	14.992920620732097	238.1012327386234	0	3	credit_card	0	117.9538455998438	0	Luxor	Alex	Android	month-to-month	0.0	0.0	0
Standard	14.265559568353169	436.0729373372251	1	0	credit_card	1	117.6754082594922	0	Alexandria	Delta	iOS	one-year	8.659837370050514	30.0629987645726	1
Standard	28.430911118719333	230.71894049716315	1	2	debit_card	0	135.55244453444857	0	Asyut	Cairo	iOS	one-year	4.984678442624524	22.99957134323449	0
Basic	nan	nan	0	0	wallet	1	122.47226488703588	0	Tanta	UpperEgypt	Android	month-to-month	0.0	0.0	0
Standard	nan	194.0017852448618	1	2	nan	0	80.71037418586582	0	Aswan	Cairo	Android	two-year	7.54276624338297	49.05833918949909	0
Premium	nan	321.0440748650616	0	0	cash	0	125.85111638894274	0	Giza	UpperEgypt	Android	month-to-month	4.048484325648612	22.85554768594265	0
Standard	nan	137.57828422197116	1	0	cash	0	118.19053961689691	0	Mansoura	Delta	iOS	month-to-month	0.0	38.52180955564511	0
Basic	13.294816358313312	332.21838201176865	1	0	wallet	0	118.67212062468386	0	Luxor	Cairo	iOS	month-to-month	0.0	11.812070628827165	0
Basic	35.42843551562986	255.8306783769058	0	2	wallet	0	173.1423108724528	1	Asyut	Giza	iOS	one-year	0.0	25.57138763461409	1
Basic	14.611941575125758	243.1528154711356	2	1	credit_card	1	94.42344333141726	1	Alexandria	Alex	Android	two-year	0.0	18.630874528667647	0

Figure 10: Sample predictions from the best-performing model (part 2).

10 Unsupervised Learning: K-Means Clustering

K-Means clustering is applied for customer segmentation using the same preprocessing pipeline. Customers are grouped into three clusters, and churn rates are analyzed per cluster.

```python
models = {
    "log_reg_baseline": LogisticRegression(max_iter=800, class_weight="balanced", n_jobs=-1, solver="lbfgs"),
    "decision_tree": DecisionTreeClassifier(
        max_depth=8, min_samples_leaf=10, class_weight="balanced", random_state=RANDOM_STATE
    ),
    "random_forest": RandomForestClassifier(
        n_estimators=350, random_state=RANDOM_STATE, class_weight="balanced_subsample"
    ),
    "knn": KNeighborsClassifier(n_neighbors=7, weights="distance"),
    "svm_rbf": SVC(
        probability=True,
        kernel="rbf",
        class_weight="balanced",
        C=1.5,
        gamma="scale",
        random_state=RANDOM_STATE,
    ),
    "mlp": MLPClassifier(
        hidden_layer_sizes=(128, 64),
        activation="relu",
        solver="adam",
        alpha=1e-4,
        learning_rate="adaptive",
        max_iter=400,
        early_stopping=True,
        random_state=RANDOM_STATE,
    ),
}

display_name_map = {
    "log_reg_baseline": "Logistic Regression (Baseline)",
    "decision_tree": "Decision Tree Classifier",
    "random_forest": "Random Forest Classifier",
    "knn": "K-Nearest Neighbors (KNN)",
    "svm_rbf": "Support Vector Machine (RBF Kernel)",
    "mlp": "Neural Network (Multilayer Perceptron)",
}
```

Figure 11: K-Means clustering and churn rate analysis.

11 Deployment Preparation

All trained pipelines and evaluation splits are persisted using `joblib` to support deployment in a Flask web application.

```python
# Persist pipelines and evaluation splits for deployment
joblib.dump(pipelines, "all_churn_pipelines.pkl")
X_test.to_csv("X_test_churn.csv", index=False)
y_test.to_csv("y_test_churn.csv", index=False)
```

Figure 12: Persisting trained pipelines and evaluation data for deployment.

12 Academic Integrity and Project Scope

This project does not claim original authorship of the initial system. It represents an academic study of an existing machine learning project, where experimentation and analysis were performed to deepen understanding of course concepts.

13 Limitations and Future Improvements

Potential improvements aligned with course content include:

- Introducing a majority-class baseline.
- Using cross-validation for model selection.
- Performing hyperparameter sensitivity analysis.
- Tuning decision thresholds using ROC or precision–recall trade-offs.
- Justifying the number of clusters using silhouette or elbow methods.

14 Conclusion

This project demonstrates the application of supervised and unsupervised machine learning techniques to a real-world telecom churn problem. By studying and modifying an existing system, the project reinforces theoretical concepts while delivering a deployable machine learning solution.