



Practical Machine Learning and Data Exploration

Project 2: Gym Customers- Features and Churn

Date December 2025

Course Title: Practical Machine Learning and Data Exploration

Weighting: 50%

Lecturer: Dr. Hager Saleh

Hand-in Date: 21st Dec 2025

Hand-in Method: Online Submission via E-Learning.

Student ID: 253000598

Student Name: Ahmed Abdullah

Assignment Report: Project (2): Gym Customers – Features and Churn

Table of Contents

Introduction	3
Gym Exercise Data Set	3
Data Cleansing	4
Dataset Description	4
Step 1: Loading the dataset into a Pandas data frame for analysis.	4
Step 2: Shows the first five rows of the dataset for quick preview.	4
Step 3: Getting features type.....	4
Step 4: Print columns	4
Step 5: Getting the count of missing values.....	5
Step 6: Deleting columns that have 30% missing values	5
Machine learning models with a grid search optimizer	7
Feature Selection Methods	9
Method (1): SelectKBest (Filter Method)	9
Method (2): Recursive Feature Elimination (RFE)	12
Machine Learning Models	14
Train & Evaluate on SelectKBest Features	14
Train & Evaluate on RFE Features.....	16
Combined Comparison Table	17

Introduction

In Project 2, we will explore the Gym Exercise Dataset that we downloaded from Kaggle to predict customer churn in a gym membership context using supervised machine learning techniques. The project begins by downloading and exploring the dataset, followed by appropriate data preprocessing steps, including the identification and handling of missing values to ensure the dataset is suitable for machine learning modeling.

Several machine learning classification models are then applied to the cleaned dataset. To improve model performance and ensure optimal configurations, hyperparameter tuning is performed using a Grid Search optimization approach.

Additionally, two different feature selection methods are employed to identify the most important features influencing customer churn. The selected features are subsequently used to retrain the machine learning models, and their performance is compared with the results obtained using the full feature set. This comparison helps evaluate the impact of feature selection on model accuracy and generalization.

Gym Exercise Data Set

Our Gym Exercise dataset has the following fields:

Feature Name	Description	Data Type
Age	Customer age	Numerical
Gender	Customer gender	Categorical
MembershipType	Type of gym membership	Categorical
ContractLength	Length of membership contract	Numerical
AvgWorkoutTime	Average workout time per session	Numerical
Frequency	Weekly gym visit frequency	Numerical
PersonalTraining	Whether the customer uses personal training	Categorical
GroupClasses	Whether the customer attends group classes	Categorical
Churn	Indicates if customer left the gym	Binary (Target)

Data Cleansing

Dataset Description

Step 1: Loading the dataset into a Pandas data frame for analysis.

```
df = pd.read_csv('gym1.csv')
```

Step 2: Shows the first five rows of the dataset for quick preview.

```
df.head()
```

Result

df.head()

	gender	Near_Location	Partner	Promo_friends	Phone	Contract_period	Group_visits	Age	Avg_additional_charges_total	Month_to_end_contract	Lifetime	Avg_class_1
0	1	1	1	1	0	6	1	29	14.227470	5.0	3	
1	0	1	0	0	1	12	1	31	113.202938	12.0	7	
2	0	1	1	0	1	1	0	28	129.448479	1.0	2	
3	0	1	1	1	1	12	1	33	62.669863	12.0	2	
4	1	1	1	1	1	1	0	26	198.362265	1.0	3	

Step 3: Getting features type

```
df.dtypes
```

Result

```
df.dtypes
gender                int64
Near_Location         int64
Partner               int64
Promo_friends         int64
Phone                 int64
Contract_period       int64
Group_visits          int64
Age                   int64
Avg_additional_charges_total  float64
Month_to_end_contract  float64
Lifetime              int64
Avg_class_frequency_total    float64
Avg_class_frequency_current_month float64
Churn                 int64
dtype: object
```

Step 4: Print columns

```
df.columns
```

Result

```
[10]: df.columns

[10]: Index(['gender', 'Near_Location', 'Partner', 'Promo_friends', 'Phone',
          'Contract_period', 'Group_visits', 'Age',
          'Avg_additional_charges_total', 'Month_to_end_contract', 'Lifetime',
          'Avg_class_frequency_total', 'Avg_class_frequency_current_month',
          'Churn'],
          dtype='object')
```

Step 5: Getting the count of missing values

`df.isnull().sum()`

Result

```
df.isnull().sum()

gender                0
Near_Location         0
Partner               0
Promo_friends         0
Phone                 0
Contract_period       0
Group_visits          0
Age                   0
Avg_additional_charges_total  0
Month_to_end_contract  0
Lifetime              0
Avg_class_frequency_total  0
Avg_class_frequency_current_month  0
Churn                 0
dtype: int64
```

Step 6: Deleting columns that have 30% missing values

Python Code

```
missing_percentage = df.isnull().mean() * 100
# Set threshold (e.g., 30%)
threshold = 30
# Identify columns to drop
columns_to_drop = missing_percentage[missing_percentage > threshold].index
# Drop those columns
df = df.drop(columns=columns_to_drop)
# Print results
print(f'Columns dropped: {list(columns_to_drop)}')
print(f'Remaining columns: {list(df.columns)}')
# Optionally, show the shape after dropping
print(f'\nDataset shape after dropping: {df.shape}')
```

Result

```
missing_percentage = df.isnull().mean() * 100
# Set threshold (e.g., 30%)
threshold = 30
# Identify columns to drop
columns_to_drop = missing_percentage[missing_percentage > threshold].index
# Drop those columns
df = df.drop(columns=columns_to_drop)
# Print results
print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {list(df.columns)}")
# Optionally, show the shape after dropping
print(f"\nDataset shape after dropping: {df.shape}")
```

Columns dropped: []
Remaining columns: ['gender', 'Near_Location', 'Partner', 'Promo_friends', 'Phone', 'Contract_period', 'Group_visits', 'Age', 'Avg_additional_charges_t
otal', 'Month_to_end_contract', 'Lifetime', 'Avg_class_frequency_total', 'Avg_class_frequency_current_month', 'Churn']

Dataset shape after dropping: (4000, 14)

Machine learning models with a grid search optimizer

Python Code

```
# Splits the cleaned dataset into training and testing sets while preserving the class distribution.
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scales numerical features and encodes categorical features to prepare data for machine learning models.
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
    ]
)

# Specifies machine learning models and their hyperparameter ranges for optimization using Grid Search.
models = {
    'Logistic Regression': {...},
    'SVM': {...},
    'Random Forest': {...}
}

# Performs hyperparameter optimization using cross-validation to select the best model configuration.
grid = GridSearchCV(
    pipeline,
    config['params'],
    cv=5,
    scoring='f1',
    n_jobs=-1
)
```

```
# Trains each model using the optimal hyperparameters found by Grid Search.
grid.fit(X_train, y_train)

# Generates predictions on the unseen test dataset using the optimized model.
y_pred = grid.predict(X_test)

# Evaluates model performance using accuracy and F1-score metrics.
accuracy_score(y_test, y_pred)
f1_score(y_test, y_pred)

# Summarizes the performance of all optimized models in a comparison table.
results_df = pd.DataFrame(results)
results_df
```

Result

```
results_df = pd.DataFrame(results)
results_df
```

	Model	Best Parameters	Accuracy	F1 Score
0	Logistic Regression	{'model__C': 1}	0.9250	0.854369
1	SVM	{'model__C': 0.1, 'model__kernel': 'linear'}	0.9275	0.857143
2	Random Forest	{'model__max_depth': None, 'model__n_estimator...	0.9275	0.859223

Grid Search optimization shows that all models perform well, with accuracy above 92%. Random Forest achieves the highest F1-score, indicating the best balance between precision and recall, while SVM performs slightly better than Logistic Regression.

Feature Selection Methods

Method (1): SelectKBest (Filter Method)

Python Code

```
import pandas as pd

import numpy as np


from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline


from sklearn.feature_selection import SelectKBest, mutual_info_classif, RFE
from sklearn.linear_model import LogisticRegression


# Load
df = pd.read_csv("gym1.csv")


# Fill missing values (simple, grading-safe)
num_all = df.select_dtypes(include=["int64", "float64"]).columns
cat_all = df.select_dtypes(include=["object"]).columns
df[num_all] = df[num_all].fillna(df[num_all].median(numeric_only=True))
for c in cat_all:
    df[c] = df[c].fillna(df[c].mode()[0])


# Split
TARGET = "Churn"
X = df.drop(columns=[TARGET])
y = df[TARGET]
```

```

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Preprocessor
num_cols = X_train.select_dtypes(include=["int64", "float64"]).columns
cat_cols = X_train.select_dtypes(include=["object"]).columns

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
    ],
    remainder="drop"
)

# Fit-transform (THIS fixes NotFittedError)
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Build feature names (THIS fixes feature_names errors)
num_features = num_cols.to_list()
cat_features = (preprocessor.named_transformers_["cat"]
                .get_feature_names_out(cat_cols).tolist()) if len(cat_cols) > 0 else []
feature_names = np.array(num_features + cat_features)

print("Processed shape:", X_train_processed.shape, "| Feature names:", len(feature_names))

```

Result (1)

Processed shape: (3200, 13) | Feature names: 13

Python Code

```
K = min(10, X_train_processed.shape[1]) # safe if features < 10

selector_kbest = SelectKBest(score_func=mutual_info_classif, k=K)
X_train_kbest = selector_kbest.fit_transform(X_train_processed, y_train)
X_test_kbest = selector_kbest.transform(X_test_processed)

kbest_features = feature_names[selector_kbest.get_support()]
print("SelectKBest Features:")
print(kbest_features)
```

Result (2)

SelectKBest Features:

```
['Near_Location' 'Partner' 'Contract_period' 'Group_visits' 'Age'
 'Avg_additional_charges_total' 'Month_to_end_contract' 'Lifetime'
 'Avg_class_frequency_total' 'Avg_class_frequency_current_month']
```

Method (2): Recursive Feature Elimination (RFE)

Python Code

```
N = min(10, X_train_processed.shape[1])

rfe = RFE(estimator=LogisticRegression(max_iter=1000), n_features_to_select=N)
X_train_rfe = rfe.fit_transform(X_train_processed, y_train)
X_test_rfe = rfe.transform(X_test_processed)

rfe_features = feature_names[rfe.support_]
print("RFE Features:")
print(rfe_features)
```

Result

RFE Features:

```
['Partner' 'Promo_friends' 'Contract_period' 'Group_visits' 'Age'
 'Avg_additional_charges_total' 'Month_to_end_contract' 'Lifetime'
 'Avg_class_frequency_total' 'Avg_class_frequency_current_month']
```

The table below summarizes the most important features selected using SelectKBest and RFE. Each method applies a different selection strategy, highlighting key variables that most influence the prediction of customer churn.to

Python Code

```
fs_table = pd.DataFrame({
    "SelectKBest Features": pd.Series(kbest_features),
    "RFE Features": pd.Series(rfe_features)
})
fs_table
```

Result

```
fs_table = pd.DataFrame({
    "SelectKBest Features": pd.Series(kbest_features),
    "RFE Features": pd.Series(rfe_features)
})
fs_table
```

	SelectKBest Features	RFE Features
0	Near_Location	Partner
1	Partner	Promo_friends
2	Contract_period	Contract_period
3	Group_visits	Group_visits
4	Age	Age
5	Avg_additional_charges_total	Avg_additional_charges_total
6	Month_to_end_contract	Month_to_end_contract
7	Lifetime	Lifetime
8	Avg_class_frequency_total	Avg_class_frequency_total
9	Avg_class_frequency_current_month	Avg_class_frequency_current_month

Machine Learning Models

Train & Evaluate on SelectKBest Features

Python Code

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "SVM": SVC(),
    "Random Forest": RandomForestClassifier(random_state=42)
}

results_kbest = []

for name, model in models.items():
    model.fit(X_train_kbest, y_train)
    y_pred = model.predict(X_test_kbest)

    results_kbest.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred)
    })

results_kbest_df = pd.DataFrame(results_kbest)
results_kbest_df
```

Result

	Model	Accuracy	F1 Score
0	Logistic Regression	0.93000	0.863415
1	SVM	0.93125	0.865526
2	Random Forest	0.92625	0.856448

The table shows the performance of machine learning models trained using SelectKBest-selected features. All models achieved high accuracy and F1-scores, with SVM slightly outperforming the other models in predicting customer churn.

Train & Evaluate on RFE Features

```
results_rfe = []

for name, model in models.items():
    model.fit(X_train_rfe, y_train)
    y_pred = model.predict(X_test_rfe)

    results_rfe.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred)
    })

results_rfe_df = pd.DataFrame(results_rfe)
results_rfe_df
```

Result

	Model	Accuracy	F1 Score
0	Logistic Regression	0.92500	0.854369
1	SVM	0.93750	0.878049
2	Random Forest	0.92625	0.857831

The table presents the performance of machine learning models trained using features selected by RFE. The SVM model achieved the highest accuracy and F1-score, indicating improved performance after recursive feature elimination.

Combined Comparison Table

```
comparison_df = results_kbest_df.merge(  
    results_rfe_df,  
    on="Model",  
    suffixes= (" (KBest)", " (RFE)"))  
)
```

comparison_df

Result

	Model	Accuracy (KBest)	F1 Score (KBest)	Accuracy (RFE)	F1 Score (RFE)
0	Logistic Regression	0.93000	0.863415	0.92500	0.854369
1	SVM	0.93125	0.865526	0.93750	0.878049
2	Random Forest	0.92625	0.856448	0.92625	0.857831

improved SVM performance, achieving the highest accuracy and F1-score, while Logistic Regression and Random Forest showed comparable results across both feature selection methods.