final_project

April 13, 2025

1 Customer Churn Data Set Analysis (Final Project)

Developer : Htet Aung Lynn
Type : Classification
Date : 10 - April - 2025

1.1 1. Objective

The analysis will provide better prediction over customer churn rate based on correlated features of the data set. This project will be focused on prediction and interpretation for stakeholders to understand the behaviour of the customers. Moreover, they can create strategic decision-making for customer retention based on the predicted results.

[1]:
```
# Import basic libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Import sklearn libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

# Import imblearn libraries
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

[2]:
```
def evaluation_metrics(test, pred):
    accuracy = accuracy_score(test, pred)
    precision, recall, fbeta, _ = precision_recall_fscore_support(test,
                                                                   pred,
                                                                   beta = 5,
                                                                   pos_label =␣
 ↪1,      # default
```

```
                                                                    average =␣
↪'binary') # default

    return [precision, recall, fbeta, accuracy]
```

[3]:
```
def evaluation_plot(test, pred):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    # Pie chart for test data distribution
    test_unique, test_counts = np.unique(test, return_counts=True)
    axes[0].pie(test_counts,
                autopct = '%1.2f%%',
                labels = ['matched', 'unmatched'],
                colors = ['orange', 'lightgreen'],
                pctdistance = 0.6,
                labeldistance = 1.1,
                shadow = False,
                startangle = 90,
                textprops = {'size': 10})
    axes[0].set_title("Tested and Predicted Data Distribution")

    # Confusion matrix
    cf = confusion_matrix(test, pred, labels=[1,0])
    cfm = ConfusionMatrixDisplay(confusion_matrix=cf, display_labels=['True',␣
↪'False'])
    cfm.plot(ax=axes[1], cmap='Blues', colorbar=True)  # Plot into specific␣
↪subplot
    axes[1].set_title("Confusion Matrix")

    plt.suptitle("Metrics Evaluation", fontsize=14)
    plt.tight_layout()
    plt.show()
```

[4]:
```
def resample(X_train, y_train):

    # SMOTE sampler (Oversampling)
    smote_sampler = SMOTE(random_state = 123)
    # Undersampling
    under_sampler = RandomUnderSampler(random_state = 123)
    # Resampled datasets
    X_smo, y_smo = smote_sampler.fit_resample(X_train, y_train)
    X_under, y_under = under_sampler.fit_resample(X_train, y_train)

    print("Original \t:", np.unique(y_train, return_counts=True))
    print("SMOTE \t\t:", np.unique(y_smo, return_counts=True))
    print("UnderSampler \t:", np.unique(y_under, return_counts=True))
```

```
    return X_smo, y_smo, X_under, y_under
```

[5]:
```
def plot_data_balancing(data, title="Data balance (Original)"):
    ax = data.value_counts().plot.bar(color=['green','blue'])
    for p in ax.patches:
        width = p.get_width()
        height = p.get_height()

        xy = (p.get_x() + width/2, height)
        ax.annotate(f"{height} times",xy, va='bottom', ha='center',␣
 ↪color='blue')
        ax.set_title(
            title)
        ax.set_xticklabels(['No','Yes'])
        ax.set_ylabel("Counts")
```

[6]:
```
rs = 123

# LINEAR REGRESSION
def find_best_params_LR(X_train, y_train):
    parameters = {'class_weight': [{0:0.05, 1:0.95}, {0:0.1, 1:0.9}, {0:0.2, 1:
 ↪0.8}]}
    lr = LogisticRegression(random_state=rs, max_iter=1000)
    gs = GridSearchCV(estimator = lr,
                      param_grid = parameters,
                      scoring = 'f1',
                      cv = 5,
                      verbose = 1)
    gs.fit(X_train, y_train.values.ravel())
    best_params = gs.best_params_
    return gs, best_params

# DECISION TREE
def find_best_params_DT(X_train, y_train):
    parameters = {'max_depth': [5,10,15,20],
                  'class_weight': [{0:0.1, 1:0.9}, {0:0.2, 1:0.8}, {0:0.3, 1:0.
 ↪7}],
                  'min_samples_split': [2,5]}
    dtc = DecisionTreeClassifier(random_state=rs)
    gs  = GridSearchCV(estimator = dtc,
                       param_grid = parameters,
                       scoring='f1',
                       cv = 5,
                       verbose = 1)
    gs.fit(X_train, y_train.values.ravel())
    best_params = gs.best_params_
    return gs, best_params
```

```
# RANDOMFOREST
def find_best_params_RF(X_train, y_train):
    parameters = {'max_depth': [5, 10, 15, 20],
                  'n_estimators': [25, 50, 100],
                  'min_samples_split': [2, 5],
                  'class_weight': [{0:0.1, 1:0.9}, {0:0.2, 1:0.8}, {0:0.3, 1:0.
 ↪7}]}
    rf = RandomForestClassifier(random_state=rs)
    gs = GridSearchCV(estimator = rf,
                      param_grid = parameters,
                      scoring = 'f1',
                      cv = 5,
                      verbose = 1)
    gs.fit(X_train, y_train.values.ravel())
    best_params = gs.best_params_
    return gs, best_params

# XGBoost
def find_best_params_XGB(X_train, y_train):
    parameters = {'n_estimators': [25,50,100],
                  'max_depth': [5,10,15,20],
                  'learning_rate': [0.01, 0.1, 0.2]}

    xgb = XGBClassifier(random_state=rs)
    gs = GridSearchCV(estimator = rf,
                      param_grid = parameters,
                      scoring = 'f1',
                      cv = 5,
                      verbose = 1)
    gs.fit(X_train, y_train.values.ravel())
    best_params = gs.best_params_
    return gs, best_params
```

[7]:
```
def train_models(X_train, X_test, y_train, y_test):
    LR, lr_params = find_best_params_LR(X_train, y_train)
    DTC, dtc_params = find_best_params_DT(X_train, y_train)
    RFC, rf_params = find_best_params_RF(X_train, y_train)

    pred_lr = LR.predict(X_test)
    pred_dtc = DTC.predict(X_test)
    pred_rfc = RFC.predict(X_test)

    lr = evaluation_metrics(y_test, pred_lr)
    dtc = evaluation_metrics(y_test, pred_dtc)
    rfc = evaluation_metrics(y_test, pred_rfc)
```

```
    # Linear Regression evaluation scores
    # FORMAT --> [precision, recall, fbeta, accuracy]
    print('Logistic Regression')
    print('-' * 25)
    print(f'Best parameters\t: {lr_params}')
    print(f"Precision\t: {lr[0]:.2f}")
    print(f"Recall\t\t: {lr[1]:.2f}")
    print(f"F-score\t\t: {lr[2]:.2f}")
    print(f"Accuracy\t: {lr[3]:.2f}")
    print('-' * 25)

    # Decision Tree evaluation scores
    print('Decision Tree')
    print('-' * 25)
    print(f'Best parameters\t: {dtc_params}')
    print(f"Precision\t: {dtc[0]:.2f}")
    print(f"Recall\t\t: {dtc[1]:.2f}")
    print(f"F-score\t\t: {dtc[2]:.2f}")
    print(f"Accuracy\t: {dtc[3]:.2f}")
    print('-' * 25)

    # Random Forest evaluation scores
    print('Random Forest')
    print('-' * 25)
    print(f'Best parameters\t: {rf_params}')
    print(f"Precision\t: {rfc[0]:.2f}")
    print(f"Recall\t\t: {rfc[1]:.2f}")
    print(f"F-score\t\t: {rfc[2]:.2f}")
    print(f"Accuracy\t: {rfc[3]:.2f}")

    return pred_lr, pred_dtc, pred_rfc
```

[8]:
```
def resample(train1, train2):
    train1_smt, train2_smt = SMOTE(random_state = 123).fit_resample(train1,␣
 ↪train2)
    train1_rus, train2_rus = RandomUnderSampler(random_state = 123).
 ↪fit_resample(train1, train2)
    print(f'Original\t : {np.unique(train2, return_counts=True)}')
    print(f'Up-sample \t : {np.unique(train2_smt, return_counts=True)}')
    print(f'Down-sample\t : {np.unique(train2_rus, return_counts=True)}')

    return train1_smt, train2_smt, train1_rus, train2_rus
```

[9]:
```
df = pd.read_csv('im_churn.csv', index_col=False)
```

```
df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-ML201
```

1.2 2. Data Description

There are total 265482 information (6174 rows | 43 columns) inside this data set. One row represent one customer.

Source : IBM cloud | LINK
Target column8 : Class (No: 5174 | Yes: 1000)
Total Customers : 6174 customers

[10]:
```
df.head()
```

[10]:

	tenure	MonthlyCharges	TotalCharges	Partner_0	Partner_1	Dependents_0	\
0	27	70.55	1943.90	1.0	0.0	1.0	
1	69	93.30	6398.05	1.0	0.0	0.0	
2	55	59.20	3175.85	0.0	1.0	1.0	
3	49	59.60	2970.30	1.0	0.0	0.0	
4	72	109.55	7887.25	1.0	0.0	0.0	

	Dependents_1	PhoneService_0	PhoneService_1	MultipleLines_0	…	\
0	0.0	0.0	1.0	0.0	…	
1	1.0	0.0	1.0	0.0	…	
2	0.0	0.0	1.0	0.0	…	
3	1.0	0.0	1.0	0.0	…	
4	1.0	0.0	1.0	0.0	…	

	Contract_0	Contract_1	Contract_2	PaperlessBilling_0	PaperlessBilling_1	\
0	1.0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	1.0	0.0	
2	0.0	0.0	1.0	1.0	0.0	
3	1.0	0.0	0.0	0.0	1.0	
4	0.0	0.0	1.0	0.0	1.0	

	PaymentMethod_0	PaymentMethod_1	PaymentMethod_2	PaymentMethod_3	Class
0	1.0	0.0	0.0	0.0	0
1	0.0	0.0	1.0	0.0	0
2	1.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	1.0	0
4	0.0	0.0	0.0	1.0	0

[5 rows x 43 columns]

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6174 entries, 0 to 6173
Data columns (total 43 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   tenure              6174 non-null   int64
```

```
 1   MonthlyCharges      6174 non-null   float64
 2   TotalCharges        6174 non-null   float64
 3   Partner_0           6174 non-null   float64
 4   Partner_1           6174 non-null   float64
 5   Dependents_0        6174 non-null   float64
 6   Dependents_1        6174 non-null   float64
 7   PhoneService_0      6174 non-null   float64
 8   PhoneService_1      6174 non-null   float64
 9   MultipleLines_0     6174 non-null   float64
 10  MultipleLines_1     6174 non-null   float64
 11  MultipleLines_2     6174 non-null   float64
 12  InternetService_0   6174 non-null   float64
 13  InternetService_1   6174 non-null   float64
 14  InternetService_2   6174 non-null   float64
 15  OnlineSecurity_0    6174 non-null   float64
 16  OnlineSecurity_1    6174 non-null   float64
 17  OnlineSecurity_2    6174 non-null   float64
 18  OnlineBackup_0      6174 non-null   float64
 19  OnlineBackup_1      6174 non-null   float64
 20  OnlineBackup_2      6174 non-null   float64
 21  DeviceProtection_0  6174 non-null   float64
 22  DeviceProtection_1  6174 non-null   float64
 23  DeviceProtection_2  6174 non-null   float64
 24  TechSupport_0       6174 non-null   float64
 25  TechSupport_1       6174 non-null   float64
 26  TechSupport_2       6174 non-null   float64
 27  StreamingTV_0       6174 non-null   float64
 28  StreamingTV_1       6174 non-null   float64
 29  StreamingTV_2       6174 non-null   float64
 30  StreamingMovies_0   6174 non-null   float64
 31  StreamingMovies_1   6174 non-null   float64
 32  StreamingMovies_2   6174 non-null   float64
 33  Contract_0          6174 non-null   float64
 34  Contract_1          6174 non-null   float64
 35  Contract_2          6174 non-null   float64
 36  PaperlessBilling_0  6174 non-null   float64
 37  PaperlessBilling_1  6174 non-null   float64
 38  PaymentMethod_0     6174 non-null   float64
 39  PaymentMethod_1     6174 non-null   float64
 40  PaymentMethod_2     6174 non-null   float64
 41  PaymentMethod_3     6174 non-null   float64
 42  Class               6174 non-null   int64
dtypes: float64(41), int64(2)
memory usage: 2.0 MB
```

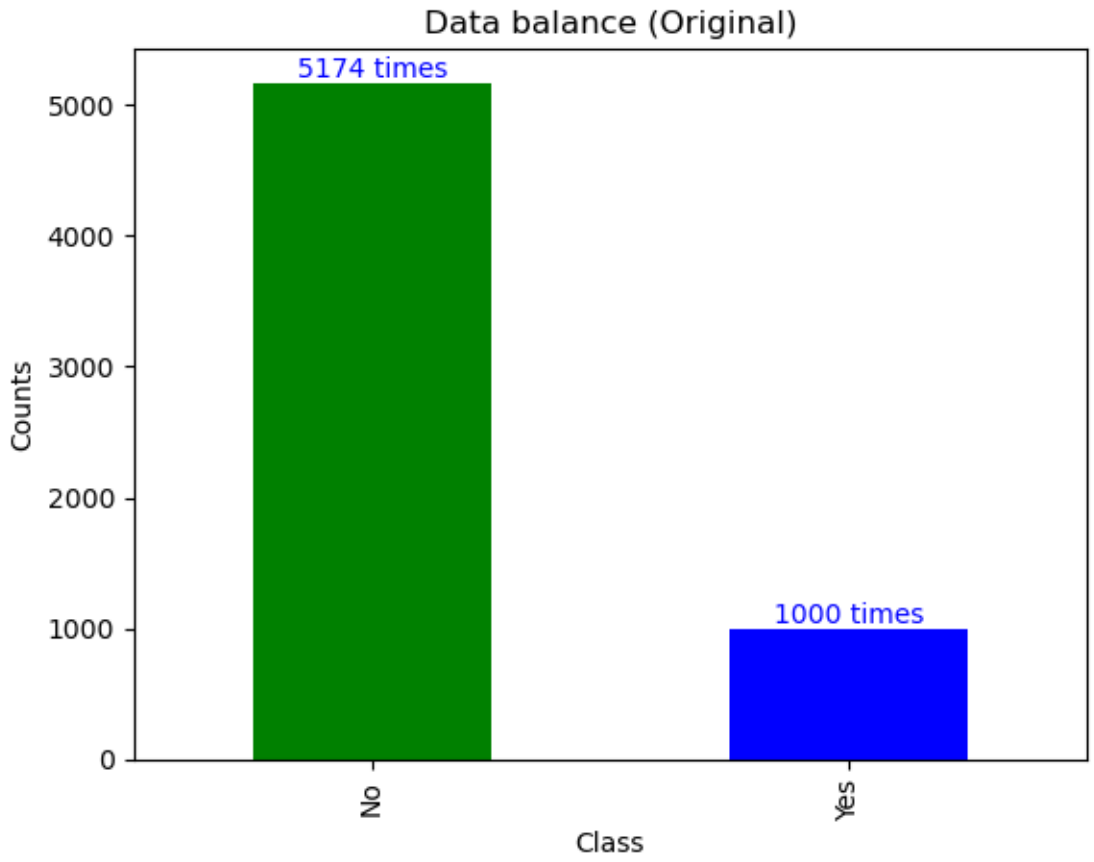
[12]:
```
df.shape
```

[12]:
```
(6174, 43)
```

[13]:
```
df.shape[0] * df.shape[1]
```

[13]:
```
265482
```

[14]:
```
plot_data_balancing(df['Class'])
```



The data in target column (`class` column) are not balanced, which can impact while splitting data.

[15]:
```
X = df.loc[:, df.columns != 'Class']
y = df[['Class']]
```

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪stratify=y, random_state=123)
```

[17]:
```
ss = StandardScaler()
X_train_s = ss.fit_transform(X_train)
X_test_s  = ss.transform(X_test)
```

[18]:
```
np.unique(y_train, return_counts=True)
```

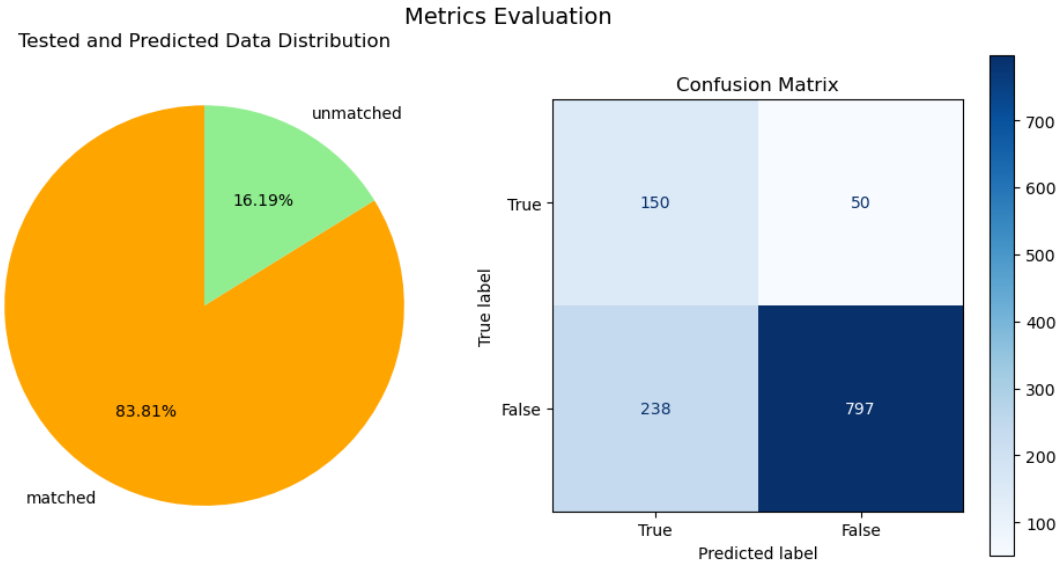
[18]:
```
(array([0, 1]), array([4139,  800]))
```

1.3 BEFORE RESAMPLING

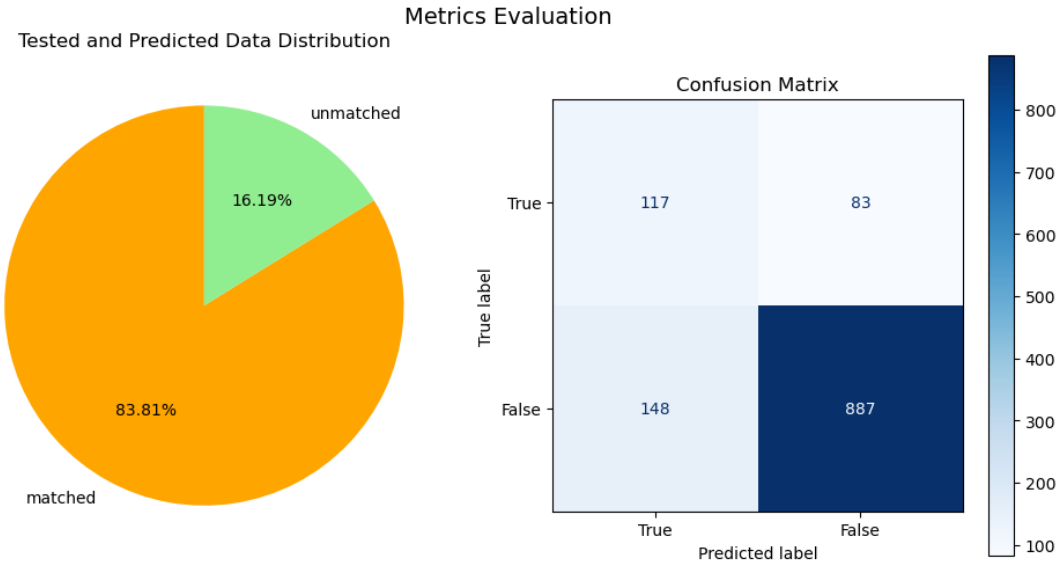
[19]:
```
pred_lr, pred_dtc, pred_rfc = train_models(X_train_s, X_test_s, y_train, y_test)
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Logistic Regression
-------------------------
Best parameters : {'class_weight': {0: 0.2, 1: 0.8}}
Precision       : 0.39
Recall          : 0.75
F-score         : 0.72
Accuracy        : 0.77
-------------------------
Decision Tree
-------------------------
Best parameters : {'class_weight': {0: 0.3, 1: 0.7}, 'max_depth': 5,
'min_samples_split': 2}
Precision       : 0.44
Recall          : 0.58
F-score         : 0.58
Accuracy        : 0.81
-------------------------
Random Forest
-------------------------
Best parameters : {'class_weight': {0: 0.2, 1: 0.8}, 'max_depth': 5,
'min_samples_split': 5, 'n_estimators': 100}
Precision       : 0.39
Recall          : 0.76
F-score         : 0.73
Accuracy        : 0.77
```

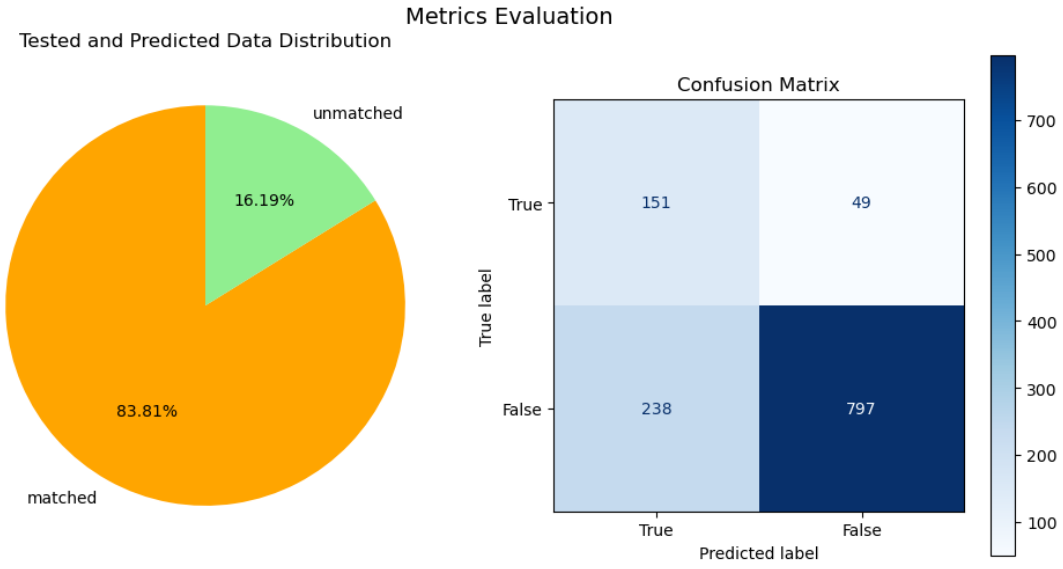
[20]:
```
evaluation_plot( y_test, pred_lr)
```



[21]:
```
evaluation_plot(y_test, pred_dtc)
```



[22]:
```
evaluation_plot( y_test, pred_rfc)
```

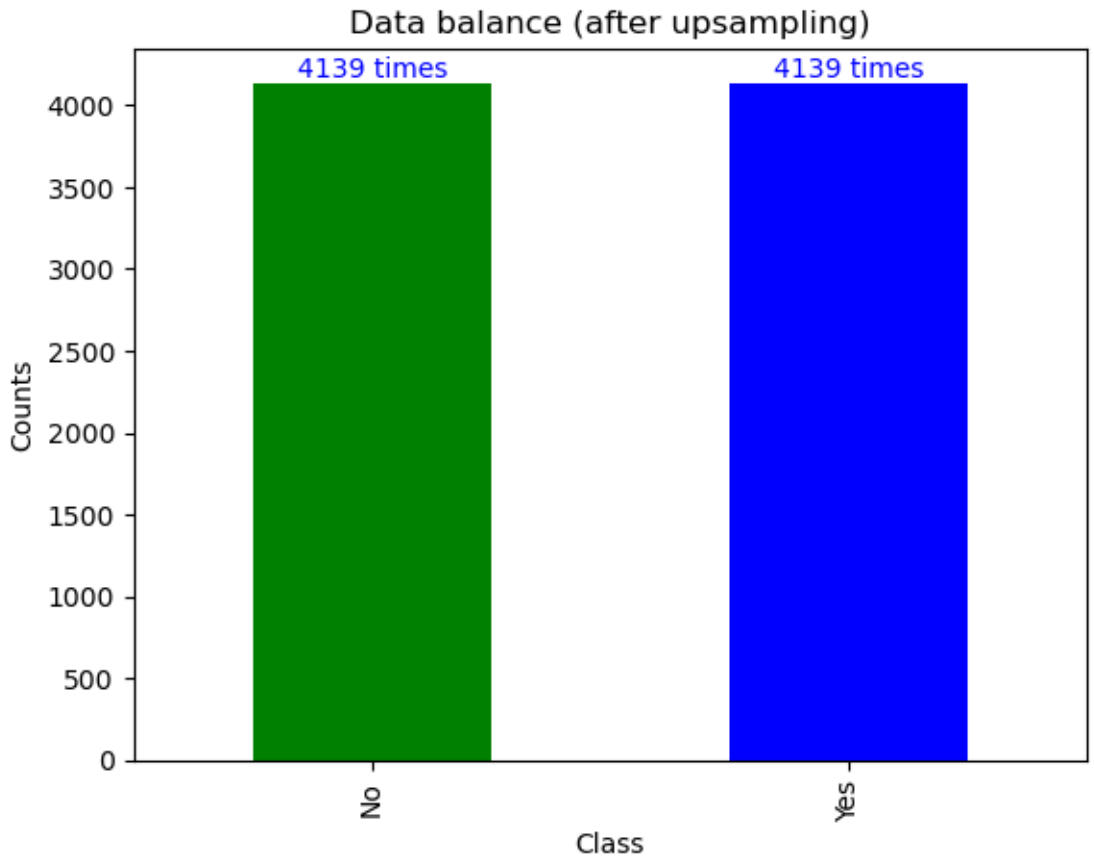


1.4 AFTER RESAMPLING

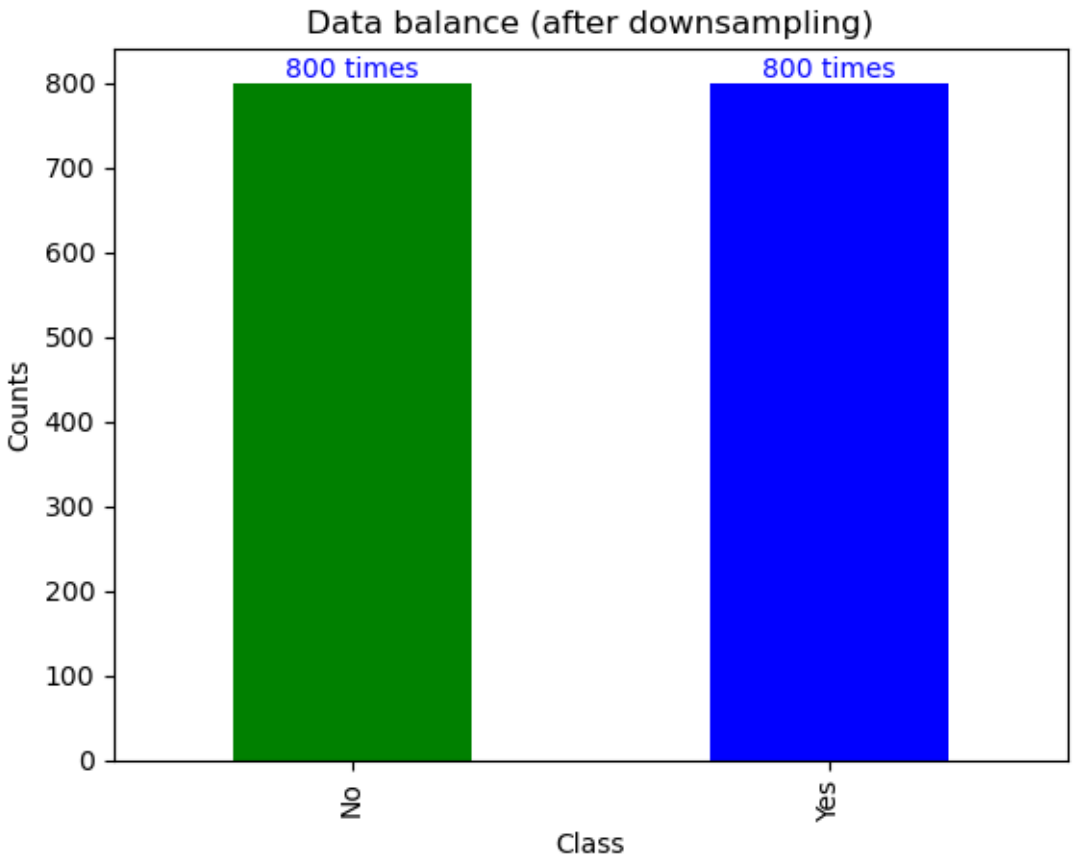
[23]:
```
X_train_sm, y_train_sm, X_train_under, y_train_under = resample(X_train_s,␣
↪y_train)
```

```
Original          : (array([0, 1]), array([4139,  800]))
Up-sample         : (array([0, 1]), array([4139, 4139]))
Down-sample       : (array([0, 1]), array([800, 800]))
```

[24]:
```
plot_data_balancing(y_train_sm, "Data balance (after upsampling)")
```



[25]:
```
plot_data_balancing(y_train_under, "Data balance (after downsampling)")
```



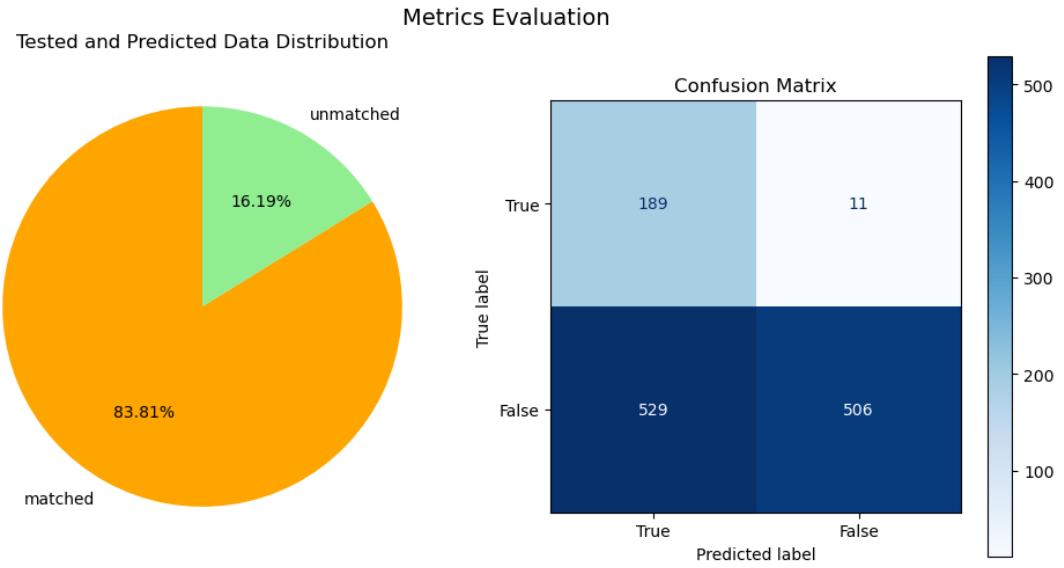
1.4.1 Upsampling model

[26]:
```
pred_lr, pred_dtc, pred_rfc = train_models(X_train_sm, X_test_s, y_train_sm,␣
↪y_test)
```

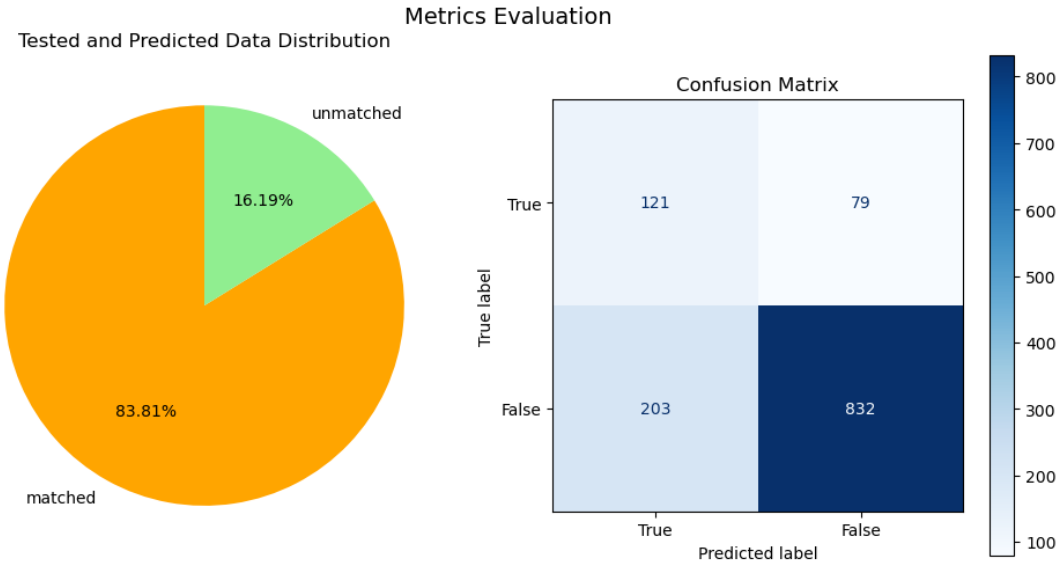
```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Logistic Regression
-------------------------
Best parameters : {'class_weight': {0: 0.2, 1: 0.8}}
Precision       : 0.26
Recall          : 0.94
F-score         : 0.86
Accuracy        : 0.56
-------------------------
Decision Tree
-------------------------
Best parameters : {'class_weight': {0: 0.3, 1: 0.7}, 'max_depth': 15,
'min_samples_split': 2}
```

```
Precision       : 0.37
Recall          : 0.60
F-score         : 0.59
Accuracy        : 0.77
-------------------------
Random Forest
-------------------------
Best parameters : {'class_weight': {0: 0.3, 1: 0.7}, 'max_depth': 20,
'min_samples_split': 2, 'n_estimators': 100}
Precision       : 0.47
Recall          : 0.52
F-score         : 0.52
Accuracy        : 0.83
```

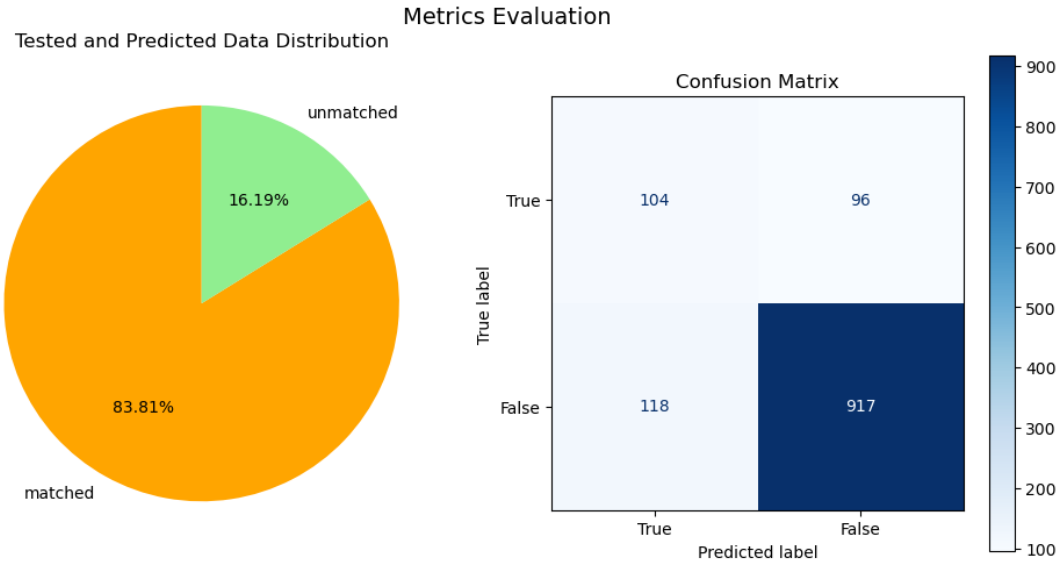
[27]:
```
evaluation_plot( y_test, pred_lr)
```



[28]:
```
evaluation_plot(y_test, pred_dtc)
```



[29]:
```
evaluation_plot(y_test, pred_rfc)
```



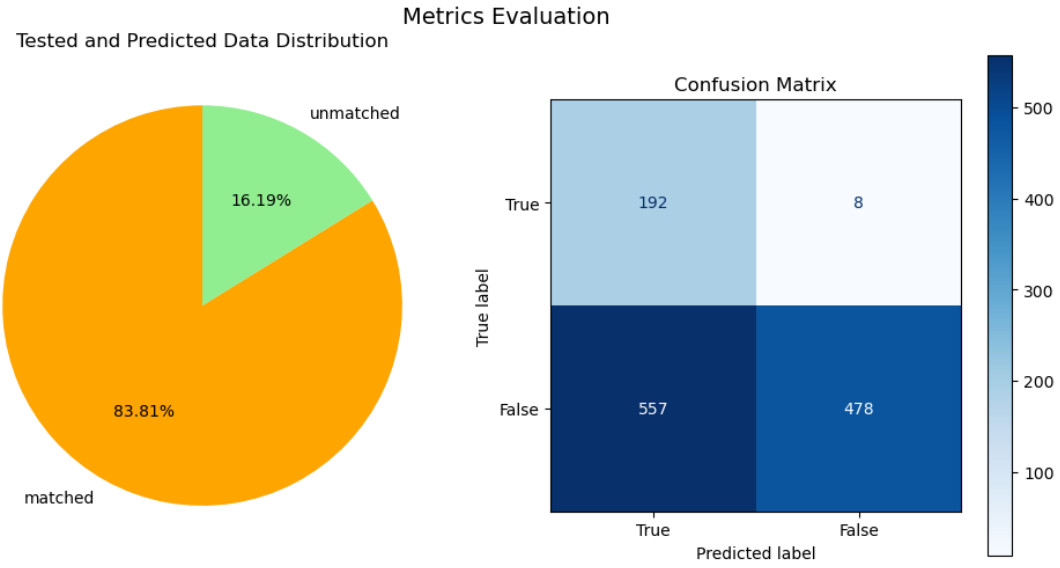
1.4.2 Downsampling model

[30]:
```
pred_lr, pred_dtc, pred_rfc = train_models(X_train_under, X_test_s,␣
↪y_train_under, y_test)
```

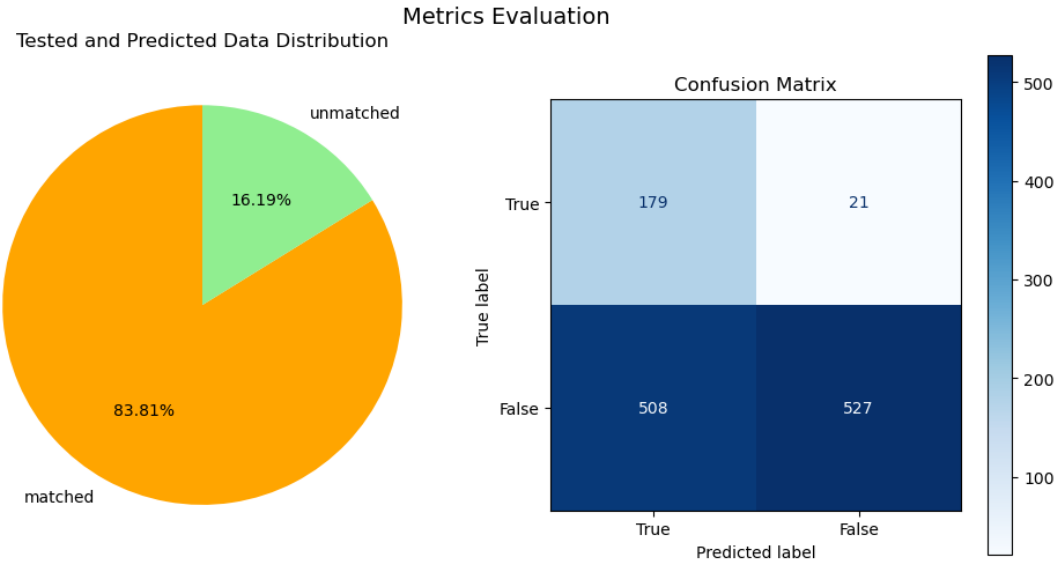
```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Logistic Regression
-------------------------
Best parameters : {'class_weight': {0: 0.2, 1: 0.8}}
Precision       : 0.26
Recall          : 0.96
F-score         : 0.87
Accuracy        : 0.54
-------------------------
Decision Tree
-------------------------
Best parameters : {'class_weight': {0: 0.3, 1: 0.7}, 'max_depth': 5,
'min_samples_split': 2}
Precision       : 0.26
Recall          : 0.90
F-score         : 0.82
Accuracy        : 0.57
-------------------------
Random Forest
-------------------------
Best parameters : {'class_weight': {0: 0.1, 1: 0.9}, 'max_depth': 20,
'min_samples_split': 5, 'n_estimators': 50}
Precision       : 0.32
Recall          : 0.89
F-score         : 0.83
Accuracy        : 0.67
```

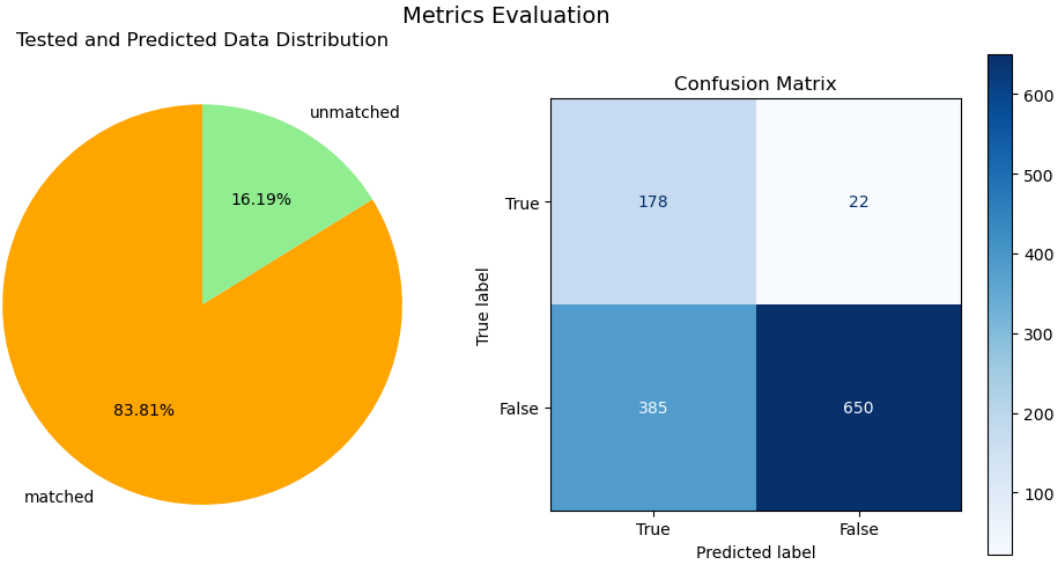
[31]:
```
evaluation_plot( y_test, pred_lr)
```



[32]:
```
evaluation_plot( y_test, pred_dtc)
```



[33]:
```
evaluation_plot( y_test, pred_rfc)
```



HTET AUNG LYNN