02_predictive_modelling_2

October 12, 2021

1 Wells Fargo Challenge

- https://www.mindsumo.com/contests/campus-analytics-challenge-2021

1.0.1 To Complete a Submission:

Build a classification model for predicting elder fraud in the digital payments space as described in Rule 4, which:

- Handles missing variables
- Maximizes the F1 score
- Uses the given data set
- Includes suitable encoding schemes
- Has the least set of feature variables

1.0.2 Resources

- https://github.com/pdglenn/WellsFargoAnalyticsChallenge

[1]:
```
import pandas as pd
import numpy as np
import pylab as plt
import seaborn as sns

data_dir = "./dataset/"

# following few lines are to supress the pandas warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

pd.options.mode.chained_assignment = None
pd.options.display.max_columns = 20
np.set_printoptions(suppress=True)

data_dir = "./dataset/"
image_dir = "./images/"
```

1.1 Loading the data

[2]:
```
# load the file
df_orig = pd.read_excel(data_dir+"trainset.xlsx", engine='openpyxl')
df_orig.head(2)
```

[2]:
```
   TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
0      5.38                 23619.91        47             4        2777
1     65.19                     0.00        45             5        2721

          PWD_UPDT_TS                CARR_NAME   RGN_NAME STATE_PRVNC_TXT  \
0  1/16/2018 11:3:58  cox communications inc.  southwest          nevada
1                NaN   charter communications  southwest      california

  ALERT_TRGR_CD  … CUST_STATE      PH_NUM_UPDT_TS CUST_SINCE_DT  \
0          MOBL  …          NV  2/24/2021 15:55:10    1993-01-06
1          MOBL  …          CA                 NaN    1971-01-07

               TRAN_TS    TRAN_DT ACTN_CD ACTN_INTNL_TXT TRAN_TYPE_CD  \
0    5/3/2021 18:3:58   5/3/2021   SCHPMT     P2P_COMMIT          P2P
1  1/13/2021 19:19:37  1/13/2021   SCHPMT     P2P_COMMIT          P2P

    ACTVY_DT FRAUD_NONFRAUD
0   5/3/2021      Non-Fraud
1  1/13/2021      Non-Fraud

[2 rows x 24 columns]
```

1.2 Train test split

Before doing any data visualization let's set some test data aside and use them to score the model later on.

[3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# stratify the target column so that the distribution look similar in the train␣
 ↪and test data
df_train0, df_test0 = train_test_split(df_orig,
                                       test_size = .2,
                                       random_state = 8848,
                                       shuffle = True,
                                       stratify = df_orig["FRAUD_NONFRAUD"])
```

1.3 Feature Engineering, Transformation and Data Imputation

- Generate New features from the given date and time features

[4]:
```
def convert_date_format(x):
    if len(str(x).split("/"))>1:
        m,d,y=str(x).strip().split()[0].split("/")
        if d=='0':
            d='1'
        elif d=='31':
            d='30'
        return "-".join([y,m,d])
    else:
        return str(x).split()[0]

def feature_engineering(df):
    # conver the _DT columns to pandas datetime
    cols_DT = [c for c in df.columns if "_DT" in c]
    df[cols_DT] = df[cols_DT].apply(pd.to_datetime)
    # convert the TRAN_Timestamp to only hour
    df["TRAN_HOUR"]=pd.to_datetime(df['TRAN_TS']).dt.strftime("%H")
    # Fill the Nulls for Phone update by the cust_since_date and keep only date
    df["PH_NUM_UPDT_DT"]=pd.to_datetime(df["PH_NUM_UPDT_TS"].
 ↪fillna(df["CUST_SINCE_DT"]).apply(convert_date_format))
    # Fill the Nulls for pwd update by the cust_since_date and keep only date
    df["PWD_UPDT_DT"]=pd.to_datetime(df["PWD_UPDT_TS"].
 ↪fillna(df["CUST_SINCE_DT"]).apply(convert_date_format))
    # Num of days between TRAN_DATE and PWD_UPDATE_DAYS
    df["PWD_UPDT_DAYS"] = (df["TRAN_DT"]-df["PWD_UPDT_DT"]).dt.days

    # Num of days between TRAN_DATE and PHONE_NUM_UPDATE_DAYS
    df["PH_NUM_UPDT_DAYS"] = (df["TRAN_DT"]-df["PH_NUM_UPDT_DT"]).dt.days

    # Num of days between TRAN_DATE and CUST_SINCE_DATE
    df["TRAN_DAYS"] = (df["TRAN_DT"] - df["CUST_SINCE_DT"]).dt.days
    # Num of days between PWD update and phone number update
    df["PH_NUM_PWD_DAYS"]=df["PH_NUM_UPDT_DAYS"] - df["PWD_UPDT_DAYS"]
    return df
```

[5]:
```
def get_imputation_values(df):

    # find numerical and categorical columns
    nume_cols = list(df.select_dtypes(include="number").columns)
    cate_cols = list(df.select_dtypes(exclude="number").columns)
    nume_cols.remove('CUST_ZIP')
    cate_cols.append('CUST_ZIP')
    nume_cols.remove('FRAUD_NONFRAUD')

    impute_vals={}

    for col in df.columns:
```

```
        if col in nume_cols:
            impute_vals[col] = df[col].median()
        elif col in cate_cols:
            impute_vals[col] = df[col].mode()[0]
    return nume_cols, cate_cols, impute_vals
```

[6]:
```
def impute_data(df, impute_dict):
    """
    this function takes in a dataframe and list of columns which have missing␣
 ↪values
    then imputes those columns using the precomputed values.
    """
    for col in list(impute_dict.keys()):
        df[col] = df[col].fillna(impute_dict[col])
    return df
```

[7]:
```
def transform_cate_data(df):
    #CUST_STATE
    # keep only CO, TX, MN, AZ and convert rest into OTHER
    df["CUST_STATE"] = df["CUST_STATE"].apply(lambda x: x if x in ["CO", "TX",␣
 ↪"MN", "AZ"] else "OTHER")
    #OPEN_ACCT_CT
    #keep only the [2,9] and convert others to 10
    df["OPEN_ACCT_CT"] = df["OPEN_ACCT_CT"].apply(lambda x: x if x in␣
 ↪range(1,13) else 13)
    return df
```

1.4 Modelling

[8]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score
from sklearn.metrics import classification_report, roc_auc_score, plot_roc_curve
```

[9]:
```
class Model_training:
    def __init__(self, model, X_train, y_train, X_test, y_test, savename="Fig"):
        self.model = model
        self.X_train = X_train
        self.y_train = y_train
        self.X_test  = X_test
        self.y_test  = y_test
        self.savename = savename
        self.model.fit(self.X_train, self.y_train)
```

```
    def print_metrics(self):
        round_to_pct = lambda x: np.round(100*x, 2)
        y_pred = self.model.predict(self.X_test)
        ac = round_to_pct(accuracy_score(self.y_test, y_pred))
        f1 = round_to_pct(f1_score(self.y_test, y_pred))
        pr = round_to_pct(precision_score(self.y_test, y_pred))
        re = round_to_pct(recall_score(self.y_test, y_pred))
        print (f"Accuracy = {ac}% F1 Score= {f1}% \nPrecision={pr}% Recall=␣
↪{re}%")
        print (classification_report(self.y_test, y_pred))
        return (self.model, (ac, f1, pr, re))

    def displot(self):
        pr=self.model.predict_proba(self.X_test)
        roc_auc = np.round(roc_auc_score(self.y_test,
                                         self.model.predict_proba(self.X_test)[:
↪, 1]), 2)

        pr_df = pd.DataFrame({'pred_0':pr[:,0],
                              'pred_1':pr[:,1],
                              'y': self.y_test})

        ax=sns.displot(data=pr_df,
                       x='pred_1',
                       hue='y',
                       alpha=0.8,
                       kind="kde",
                       height = 3.5,
                       aspect=1.8);

        plt.xlabel("Prob. Positive Predictions", fontsize=16)
        plt.text(0.2, 2, "ROC_AUC="+str(roc_auc), fontsize=16)
        plt.ylabel("Density", fontsize=16)

        plt.yticks(fontsize=16);
        plt.xticks(fontsize=16);
        figname = "images/displot_"+self.savename+"_nb3.png"
        plt.savefig(figname, dpi=300, bbox_inches='tight')

    def feature_importance(self):

        try:
            mod = self.model.base_estimator
            #mod.feature_importances_
        except:
            mod = self.model.best_estimator_
```

```
        df_imp = pd.DataFrame({"Feature":self.X_train.columns,
                               "Feature Importance":mod.feature_importances_})

        #df_imp = pd.DataFrame({"Feature":self.X_train.columns,
        #                       "Feature Importance":self.model.
↪feature_importances_})

        df_imp = df_imp.sort_values(by=['Feature Importance'],
                                    axis=0,
                                    ascending=True)

        df_imp.plot(kind='barh',
                    x='Feature',
                    y='Feature Importance',
                    color="C2", figsize=(8,5));

        plt.grid(axis='x')
        plt.yticks(fontsize=16);
        plt.ylabel('');
        plt.xticks(fontsize=16);
        plt.legend(loc='best',fontsize=16);

        figname = "images/feat_imp_"+self.savename+"_nb3.png"
        plt.savefig(figname, dpi=300, bbox_inches='tight')

    def plot_roc_curve(self):
        roc_auc = np.round(roc_auc_score(self.y_test,
                                         self.model.predict_proba(self.X_test)[:
↪, 1]), 2)
        label_name = self.savename +"\nAUC = "+str(roc_auc)

        plot_roc_curve(self.model, self.X_test, self.y_test,
                       lw=3., color='C2', label=label_name)
        plt.title("ROC Curve", fontsize=18)
        plt.xlabel("False Positive Rate", fontsize=16)
        plt.ylabel("True Positive Rate", fontsize=16)
        plt.xticks(fontsize=16);
        plt.yticks(fontsize=16);
        plt.legend(loc="center", fontsize=14);
        plt.axvline(x=0, color='k', ls='--', lw=1)
        plt.axhline(y=0, color='k', ls='--', lw=1)
        plt.axhline(y=1, color='k', ls='--', lw=1)

        figname = "images/roc_curve_"+self.savename+"_nb3.png"
        plt.savefig(figname, dpi=300, bbox_inches='tight')
```

1.5 Modeling 1: Numerical features (Given only)

[10]:
```
df1 = df_train0.copy()
df1["FRAUD_NONFRAUD"] =  df1["FRAUD_NONFRAUD"].map({"Fraud":1, "Non-Fraud":0})
df1 = feature_engineering(df1)
nume_cols, cate_cols, impute_vals = get_imputation_values(df1)
df1 = impute_data(df1, impute_vals)
df1 = transform_cate_data(df1)
```

[11]:
```
df1.CUST_STATE.nunique()
```

[11]:
```
5
```

[12]:
```
df1_te = df_test0.copy()
df1_te["FRAUD_NONFRAUD"] =  df1_te["FRAUD_NONFRAUD"].map({"Fraud":1,␣
↪"Non-Fraud":0})
df1_te = feature_engineering(df1_te)
df1_te = impute_data(df1_te, impute_dict=impute_vals)
df1_te = transform_cate_data(df1_te)
```

[13]:
```
nume_cols1 = list(df_train0.select_dtypes(include="number").columns)
nume_cols1.remove('CUST_ZIP')
nume_cols1
```

[13]:
```
['TRAN_AMT',
 'ACCT_PRE_TRAN_AVAIL_BAL',
 'CUST_AGE',
 'OPEN_ACCT_CT',
 'WF_dvc_age']
```

[14]:
```
X_train1, y_train1 = df1[nume_cols1], df1["FRAUD_NONFRAUD"]
X_test1,  y_test1  = df1_te[nume_cols1], df1_te["FRAUD_NONFRAUD"]
X_train1.shape, y_train1.shape, X_test1.shape, y_test1.shape
```

[14]:
```
((11200, 5), (11200,), (2800, 5), (2800,))
```

[15]:
```
X_train1.head(2)
```

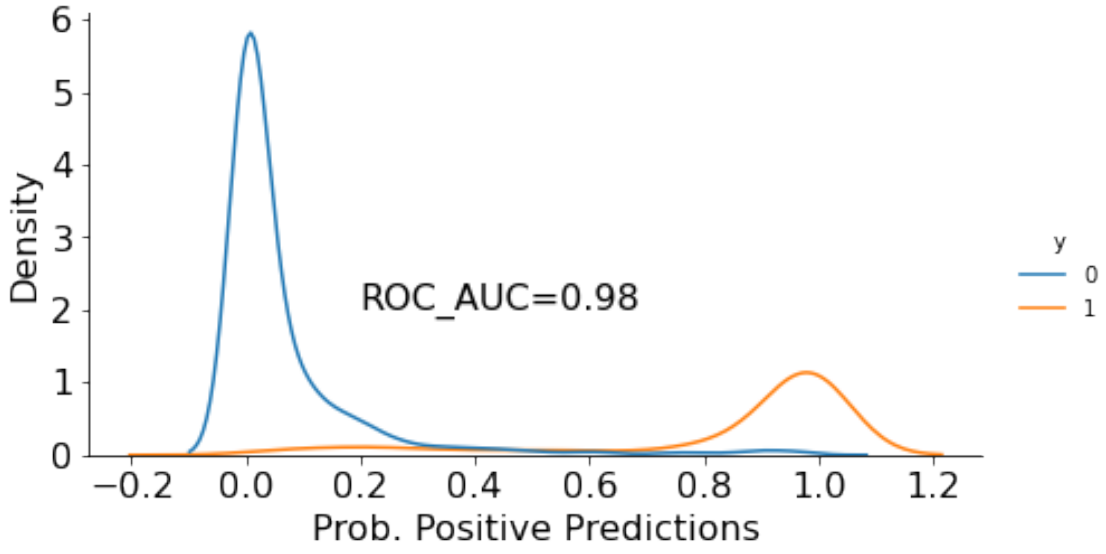
[15]:

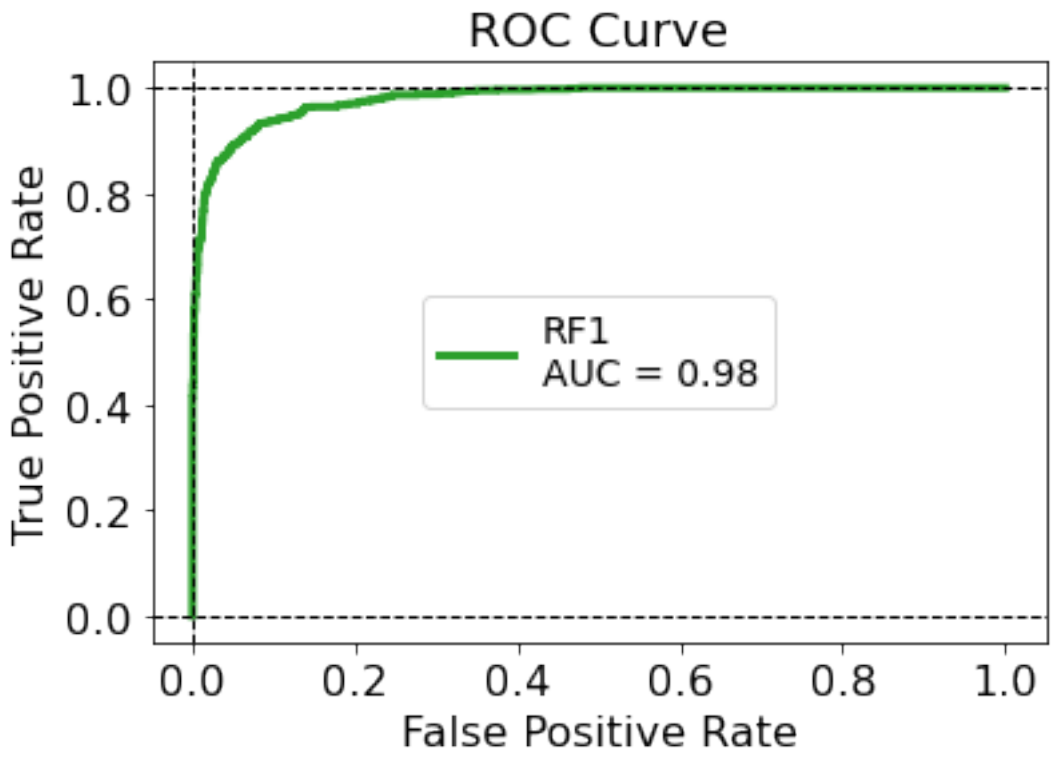
	TRAN_AMT	ACCT_PRE_TRAN_AVAIL_BAL	CUST_AGE	OPEN_ACCT_CT	WF_dvc_age
2413	487.93	3714.91	43	5	1037
1003	4.84	0.00	53	5	305

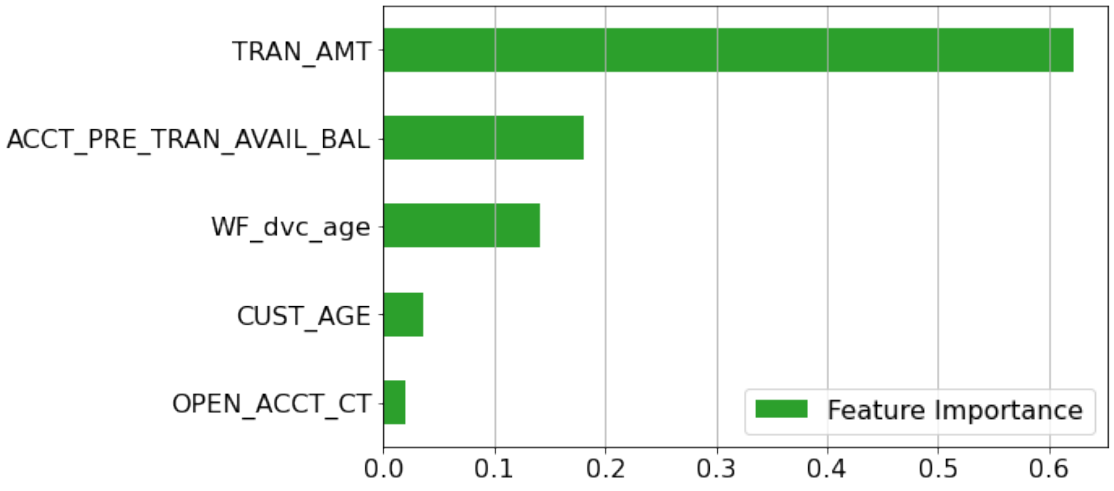
[16]:
```
model_rf_gs = GridSearchCV(RandomForestClassifier(),
                           param_grid={'max_depth':[12, 13, 14, 15, 16, 17]},
                           scoring ='f1',
                           verbose=1)
```

```
mod = Model_training(model_rf_gs,
                     X_train1, y_train1, X_test1, y_test1,
                     "RF1")
mod_tr, _= mod.print_metrics()
print ("Best parameters:", mod_tr.best_params_)
mod.displot()
mod.plot_roc_curve()
mod.feature_importance()
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:   25.9s finished

Accuracy = 93.46% F1 Score= 88.51%
Precision=92.76% Recall= 84.63%
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      1967
           1       0.93      0.85      0.89       833

    accuracy                           0.93      2800
   macro avg       0.93      0.91      0.92      2800
weighted avg       0.93      0.93      0.93      2800

Best parameters: {'max_depth': 13}
```





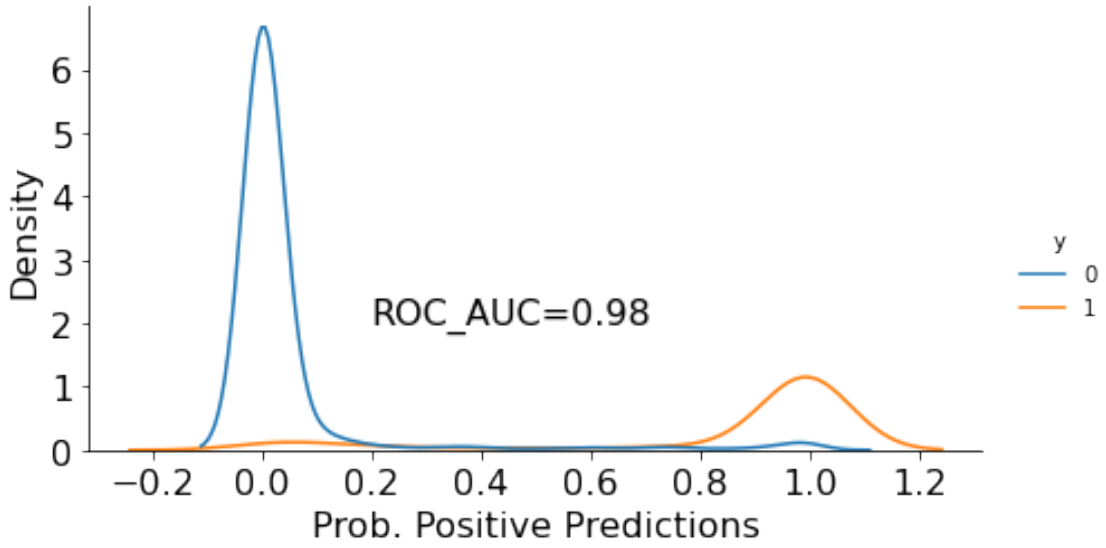


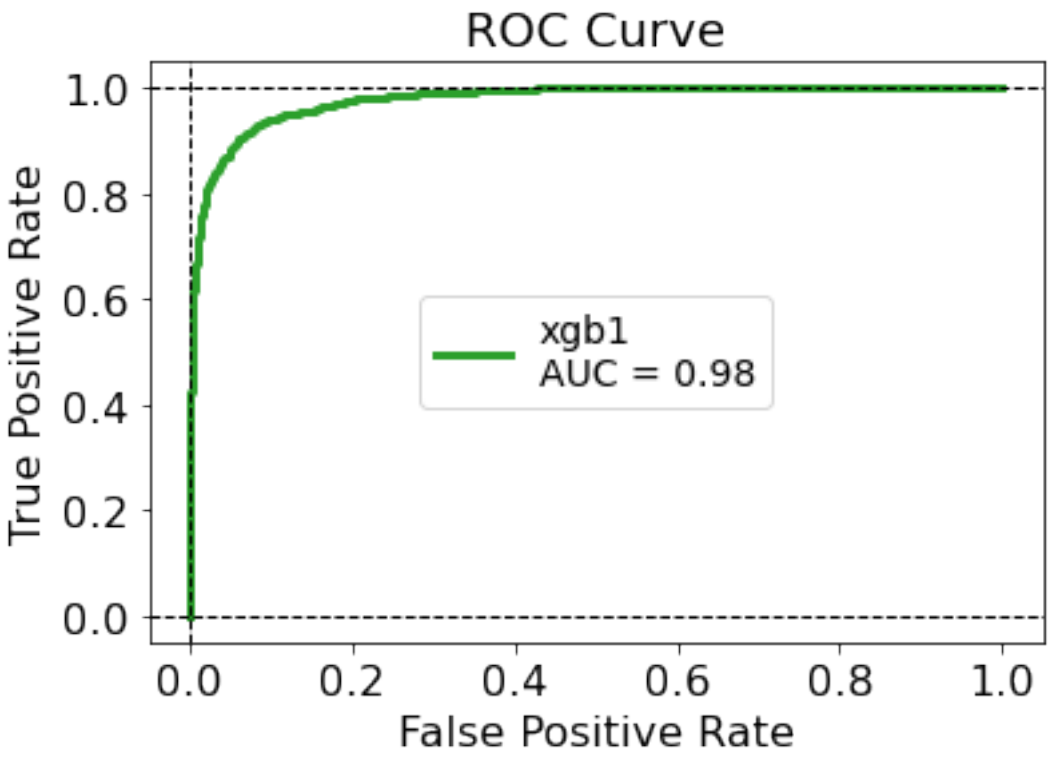
[17]:
```
xgb_gs = GridSearchCV(XGBClassifier(),
                      param_grid={'max_depth':[10, 11, 12, 13, 14],
                                  'eval_metric' :["logloss"],
                                  'reg_alpha':[0.1, 0.5]},
                      scoring = 'f1',
```

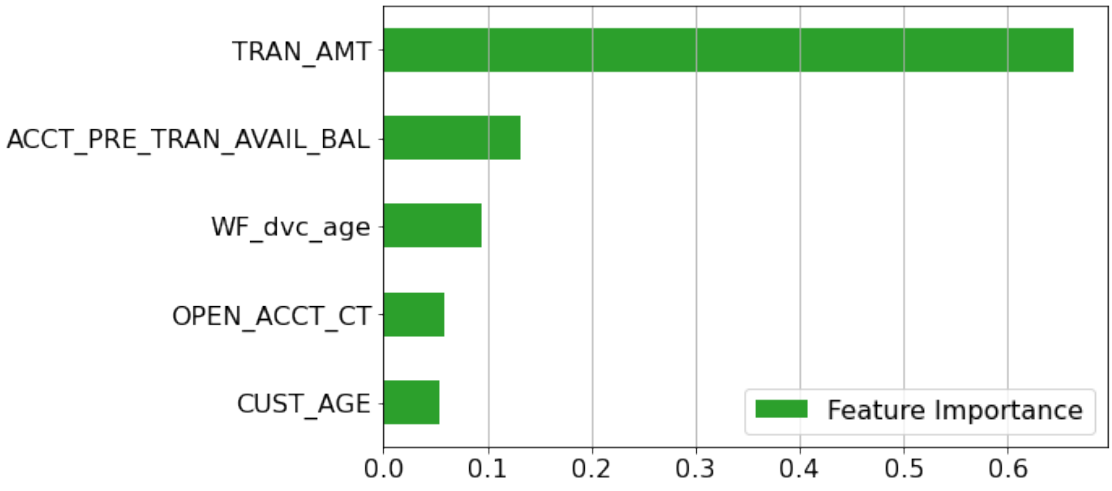
```
                      verbose = 0)

mod = Model_training(xgb_gs, X_train1, y_train1, X_test1, y_test1, "xgb1")
mod_tr, _= mod.print_metrics()
print ("Best parameters:", mod_tr.best_params_)
mod.displot()
mod.plot_roc_curve()
mod.feature_importance()
```

```
Accuracy = 92.93% F1 Score= 87.78%
Precision=90.34% Recall= 85.35%
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1967
           1       0.90      0.85      0.88       833

    accuracy                           0.93      2800
   macro avg       0.92      0.91      0.91      2800
weighted avg       0.93      0.93      0.93      2800
```

```
Best parameters: {'eval_metric': 'logloss', 'max_depth': 11, 'reg_alpha': 0.1}
```




ROC Curve
True Positive Rate
False Positive Rate
xgb1
AUC = 0.98
0.0
0.2
0.4
0.6
0.8
1.0


TRAN_AMT
ACCT_PRE_TRAN_AVAIL_BAL
WF_dvc_age
OPEN_ACCT_CT
CUST_AGE
Feature Importance
0.0
0.1
0.2
0.3
0.4
0.5
0.6

1.6 Modeling 2: Numerical features (Given + Engineered)

[18]:
```
X_train2, y_train2 = df1[nume_cols], df1["FRAUD_NONFRAUD"]
X_test2,  y_test2  = df1_te[nume_cols], df1_te["FRAUD_NONFRAUD"]
X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape
```

[18]:
```
((11200, 9), (11200,), (2800, 9), (2800,))
```

[19]:
```
X_train2.head(2)
```

[19]:

	TRAN_AMT	ACCT_PRE_TRAN_AVAIL_BAL	CUST_AGE	OPEN_ACCT_CT	WF_dvc_age	\
2413	487.93	3714.91	43	5	1037	
1003	4.84	0.00	53	5	305	

	PWD_UPDT_DAYS	PH_NUM_UPDT_DAYS	TRAN_DAYS	PH_NUM_PWD_DAYS
2413	12146	347	12146	-11799
1003	1478	12443	12443	10965

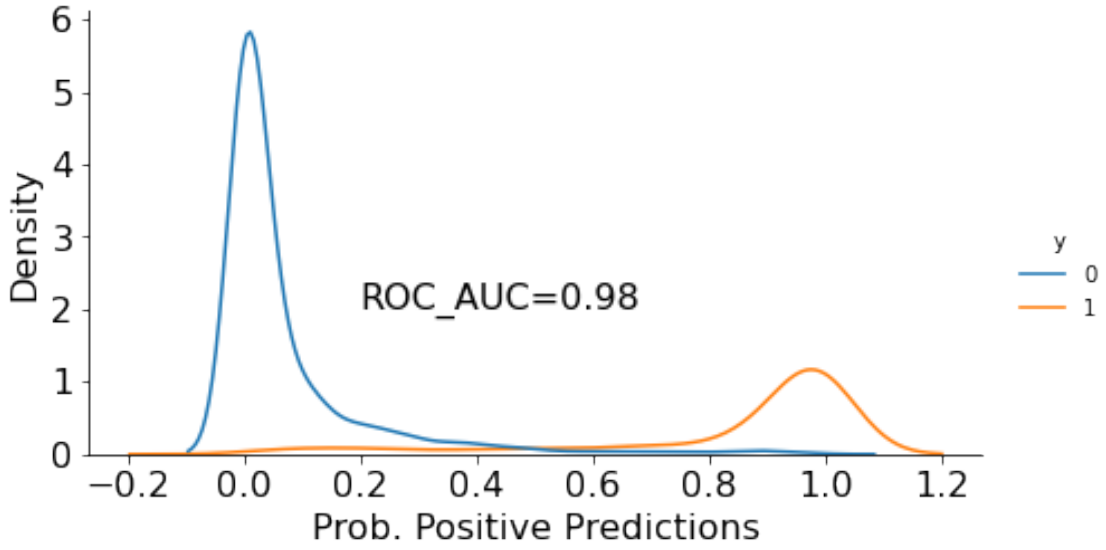
[20]:
```
(X_train2["PWD_UPDT_DAYS"]<0).sum()
```

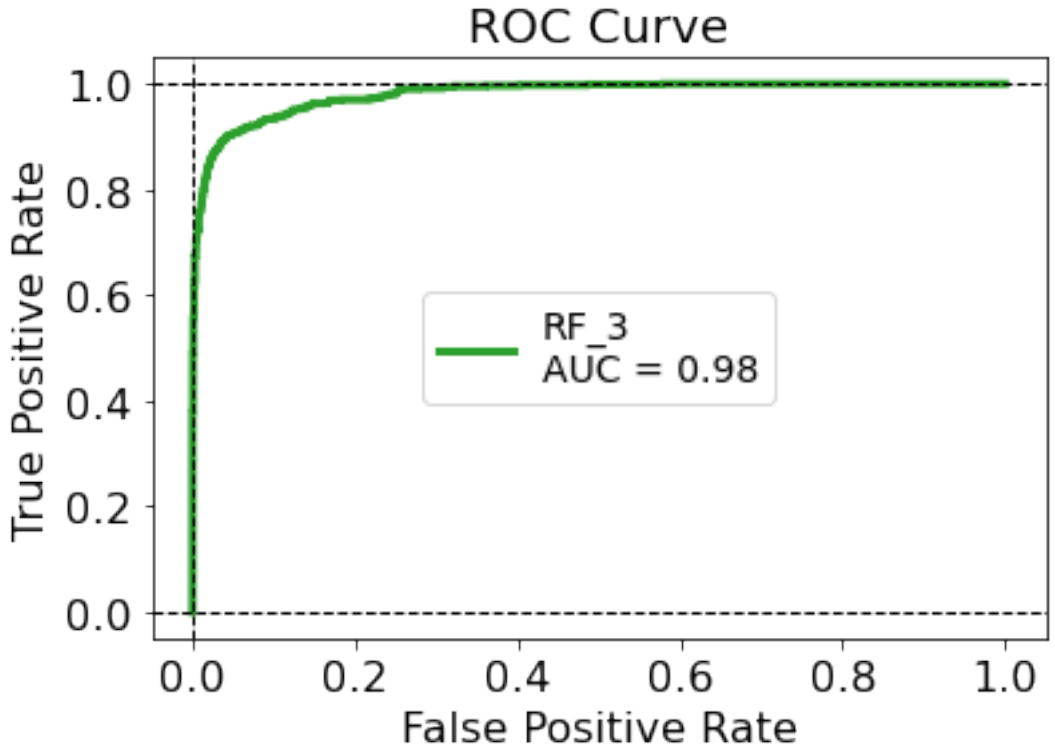
[20]:
```
1509
```

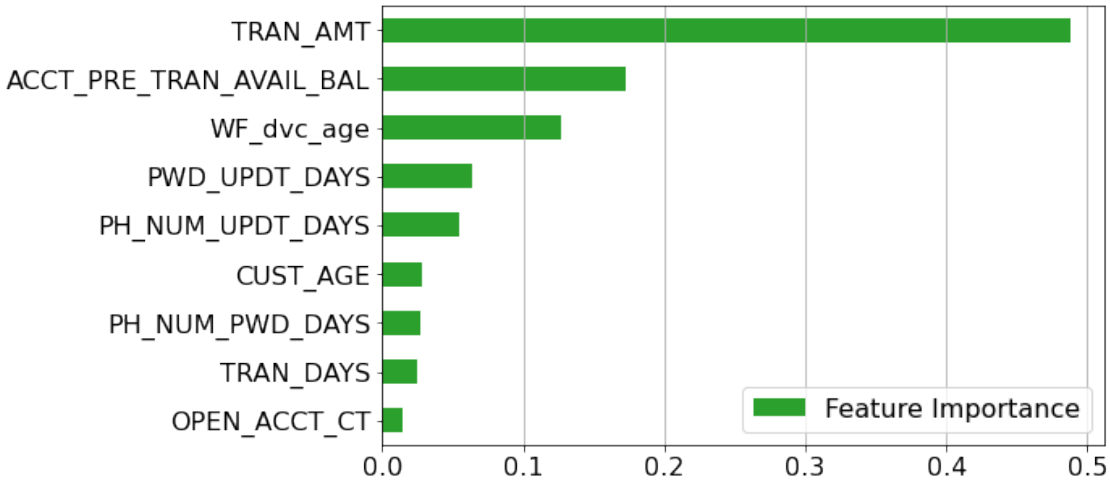
[21]:
```
model_rf_gs = GridSearchCV(RandomForestClassifier(),
                           param_grid={'max_depth':[12, 13, 14, 15, 16, 17]},
                           scoring ='f1',
                           verbose=1)

mod = Model_training(model_rf_gs,
                     X_train2, y_train2, X_test2, y_test2,
                     "RF_3")
mod_tr, _= mod.print_metrics()
print ("Grid Search Best Parameters", mod_tr.best_params_)
mod.displot()
mod.plot_roc_curve()
mod.feature_importance()
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:   43.5s finished

Accuracy = 94.18% F1 Score= 89.87%
Precision=93.17% Recall= 86.79%
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1967
           1       0.93      0.87      0.90       833

    accuracy                           0.94      2800
```

```
   macro avg       0.94      0.92      0.93      2800
weighted avg       0.94      0.94      0.94      2800

Grid Search Best Parameters {'max_depth': 17}
```



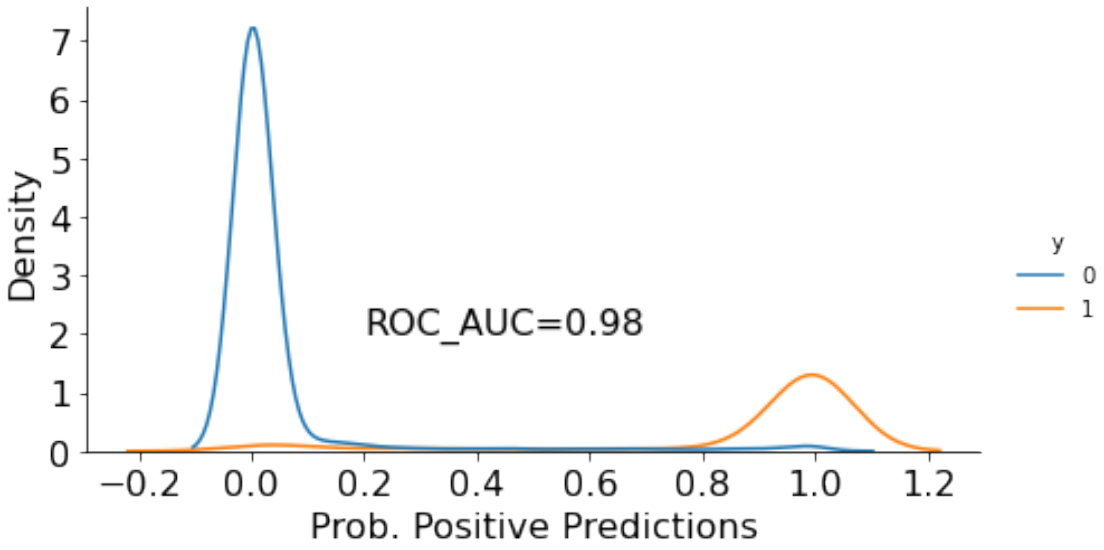


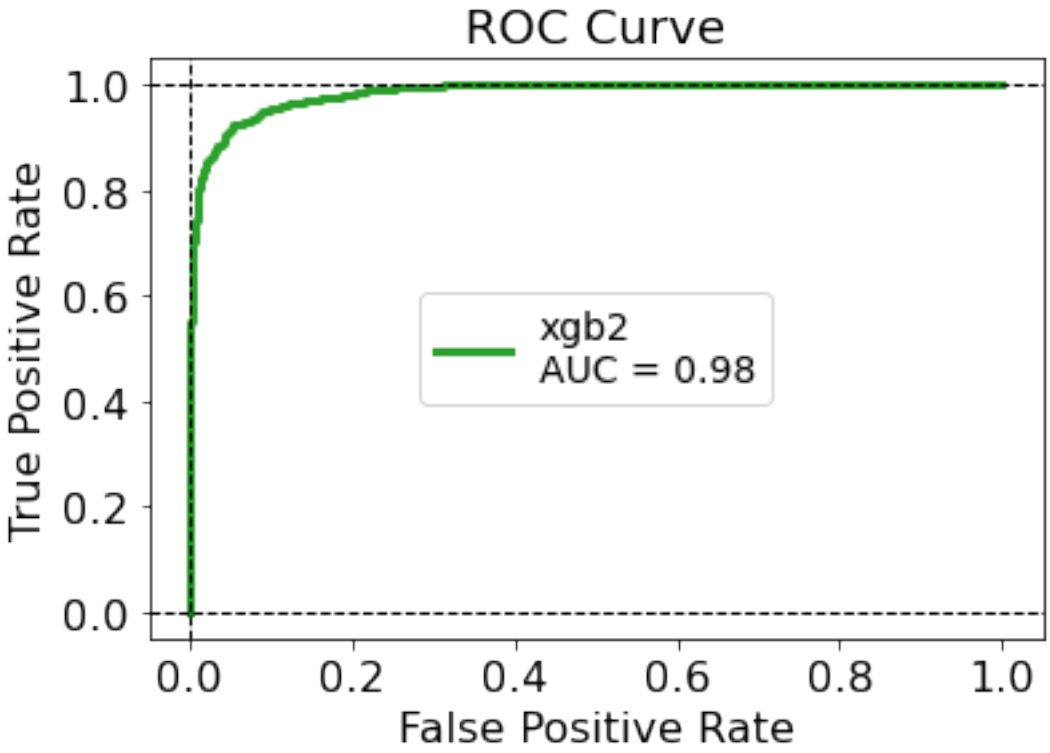


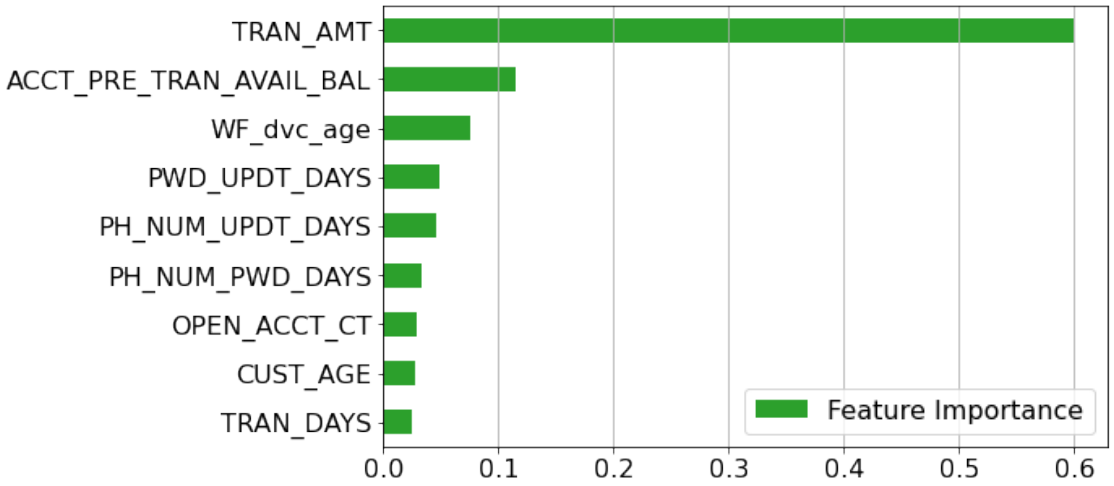
[ ]:

[23]:
```
xgb_gs = GridSearchCV(XGBClassifier(),
                      param_grid={'max_depth':[8, 9, 10, 11, 12, 13, 14],
                                  'eval_metric' :["logloss"],
                                  'reg_alpha':[0.1, 0.5]},
                      scoring = 'f1',
                      verbose = 0)

mod = Model_training(xgb_gs, X_train2, y_train2, X_test2, y_test2, "xgb2")
mod_tr, _= mod.print_metrics()
print ("Best parameters:", mod_tr.best_params_)
mod.displot()
mod.plot_roc_curve()
mod.feature_importance()
```

```
Accuracy = 94.07% F1 Score= 89.83%
Precision=91.74% Recall= 88.0%
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1967
           1       0.92      0.88      0.90       833

    accuracy                           0.94      2800
   macro avg       0.93      0.92      0.93      2800
weighted avg       0.94      0.94      0.94      2800

Best parameters: {'eval_metric': 'logloss', 'max_depth': 10, 'reg_alpha': 0.5}
```


Density
Prob. Positive Predictions
ROC_AUC=0.98
y
0
1


ROC Curve
True Positive Rate
False Positive Rate
xgb2
AUC = 0.98



1.7 Modeling 3: Numerical + Categorical features

[25]:
```
import category_encoders as ce
```

[26]:
```
df1.head(2)
```

[26]:
```
       TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413     487.93                  3714.91        43             5        1037
1003       4.84                     0.00        53             5         305

              PWD_UPDT_TS                CARR_NAME   RGN_NAME STATE_PRVNC_TXT  \
2413    5/18/2020 4:7:20  cox communications inc.  southwest      california
1003  4/12/2017 15:54:53  cox communications inc.  southwest      california

     ALERT_TRGR_CD  … TRAN_TYPE_CD   ACTVY_DT FRAUD_NONFRAUD  TRAN_HOUR  \
2413          MOBL  …          P2P 2021-04-13              1         05
1003          MOBL  …          P2P 2021-04-29              0         22

     PH_NUM_UPDT_DT PWD_UPDT_DT PWD_UPDT_DAYS PH_NUM_UPDT_DAYS TRAN_DAYS  \
2413     2020-05-01  1988-01-11         12146              347     12146
1003     1987-04-05  2017-04-12          1478            12443     12443

     PH_NUM_PWD_DAYS
2413          -11799
1003           10965

[2 rows x 31 columns]
```

[27]:
```
df2 = df_train0.copy()
df2["FRAUD_NONFRAUD"] =  df2["FRAUD_NONFRAUD"].map({"Fraud":1, "Non-Fraud":0})
```

```
df2 = feature_engineering(df2)
nume_cols, cate_cols, impute_vals = get_imputation_values(df2)
df2 = impute_data(df2, impute_vals)
df2 = transform_cate_data(df2)
```

[28]:
```
df2.head(2)
```

[28]:
```
      TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413    487.93                  3714.91        43             5        1037
1003      4.84                     0.00        53             5         305

             PWD_UPDT_TS                CARR_NAME   RGN_NAME STATE_PRVNC_TXT  \
2413    5/18/2020 4:7:20  cox communications inc.  southwest      california
1003  4/12/2017 15:54:53  cox communications inc.  southwest      california

     ALERT_TRGR_CD  … TRAN_TYPE_CD   ACTVY_DT FRAUD_NONFRAUD  TRAN_HOUR  \
2413          MOBL  …          P2P 2021-04-13             1         05
1003          MOBL  …          P2P 2021-04-29             0         22

     PH_NUM_UPDT_DT PWD_UPDT_DT PWD_UPDT_DAYS PH_NUM_UPDT_DAYS TRAN_DAYS  \
2413     2020-05-01  1988-01-11         12146              347     12146
1003     1987-04-05  2017-04-12          1478            12443     12443

     PH_NUM_PWD_DAYS
2413          -11799
1003           10965

[2 rows x 31 columns]
```

[29]:
```
cate_cols_to_keep = ['ALERT_TRGR_CD', "CUST_STATE"]
```

[30]:
```
encoder = ce.OneHotEncoder()
df2_tr_cat = encoder.fit_transform(df2[cate_cols_to_keep])
df2_tr_join = pd.concat( [df2[nume_cols], df2_tr_cat], axis=1)
X_train2 = df2_tr_join
y_train2 = df2["FRAUD_NONFRAUD"]
```

[31]:
```
df2_te = df_test0.copy()
df2_te["FRAUD_NONFRAUD"] =  df2_te["FRAUD_NONFRAUD"].map({"Fraud":1,␣
↪"Non-Fraud":0})
df2_te = feature_engineering(df2_te)
df2_te = impute_data(df2_te, impute_dict=impute_vals)
df2_te = transform_cate_data(df2_te)
```

[32]:
```
df2_te_cat = encoder.transform(df2_te[cate_cols_to_keep])
df2_te_join = pd.concat( [df2_te[nume_cols], df2_te_cat], axis=1)
X_test2 = df2_te_join
```

```
y_test2 = df2_te["FRAUD_NONFRAUD"]
```

[33]:
```
X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape
```

[33]:
```
((11200, 16), (11200,), (2800, 16), (2800,))
```

[34]:
```
"FRAUD_NONFRAUD" in list(X_test2.columns)
```

[34]:
```
False
```

[35]:
```
X_train2.head(2)
```

[35]:

	TRAN_AMT	ACCT_PRE_TRAN_AVAIL_BAL	CUST_AGE	OPEN_ACCT_CT	WF_dvc_age	\
2413	487.93	3714.91	43	5	1037	
1003	4.84	0.00	53	5	305	

	PWD_UPDT_DAYS	PH_NUM_UPDT_DAYS	TRAN_DAYS	PH_NUM_PWD_DAYS	\
2413	12146	347	12146	-11799	
1003	1478	12443	12443	10965	

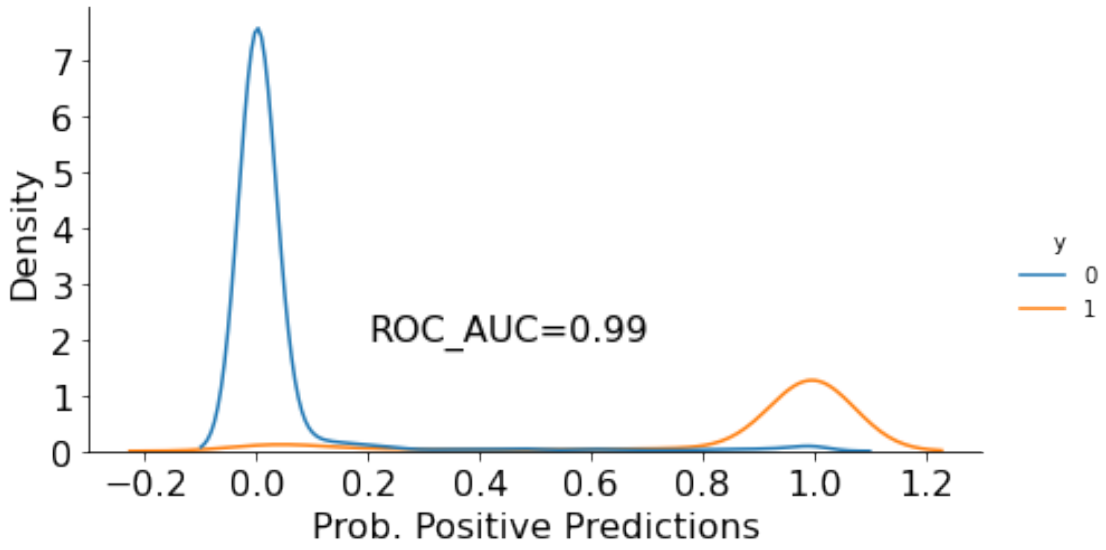
	ALERT_TRGR_CD_1	ALERT_TRGR_CD_2	CUST_STATE_1	CUST_STATE_2	\
2413	1	0	1	0	
1003	1	0	0	1	

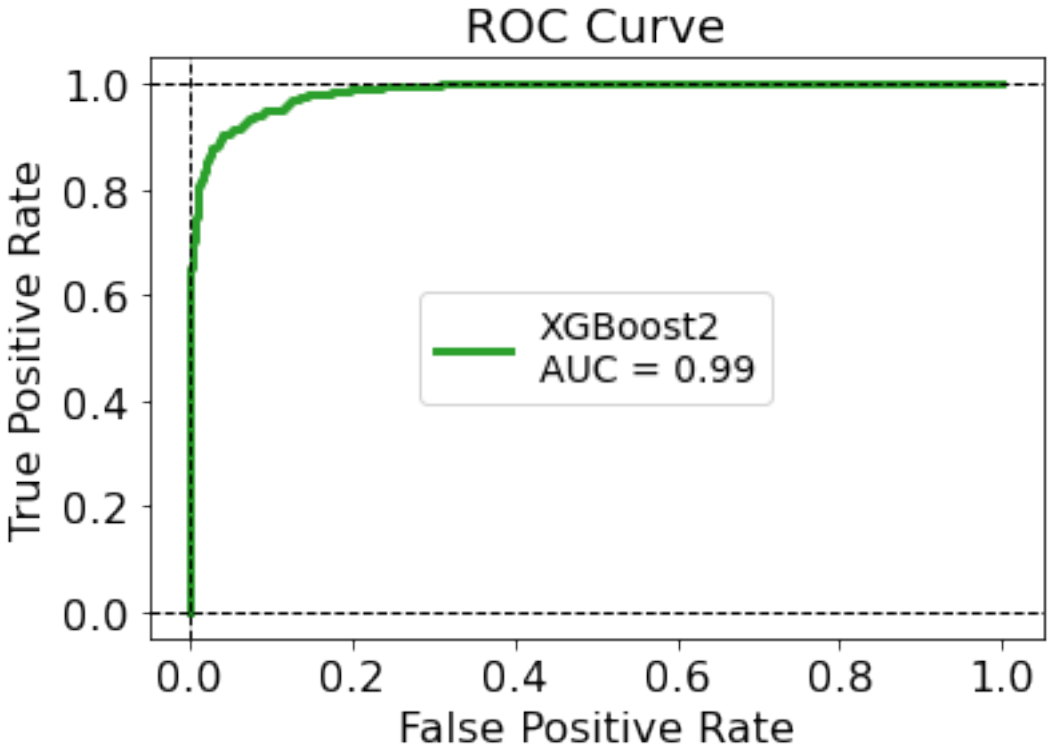
	CUST_STATE_3	CUST_STATE_4	CUST_STATE_5
2413	0	0	0
1003	0	0	0

[36]:
```
xgb = XGBClassifier(verbosity=1,
                    max_depth=10,
                    eval_metric = "logloss")

mod6 = Model_training(xgb, X_train2, y_train2, X_test2, y_test2, "XGBoost2")
mod_trained, _ = mod6.print_metrics()
mod6.displot()
mod6.plot_roc_curve()
#mod6.feature_importance()
```

```
Accuracy = 94.36% F1 Score= 90.26%
Precision=92.78% Recall= 87.88%
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1967
           1       0.93      0.88      0.90       833

    accuracy                           0.94      2800
   macro avg       0.94      0.92      0.93      2800
```

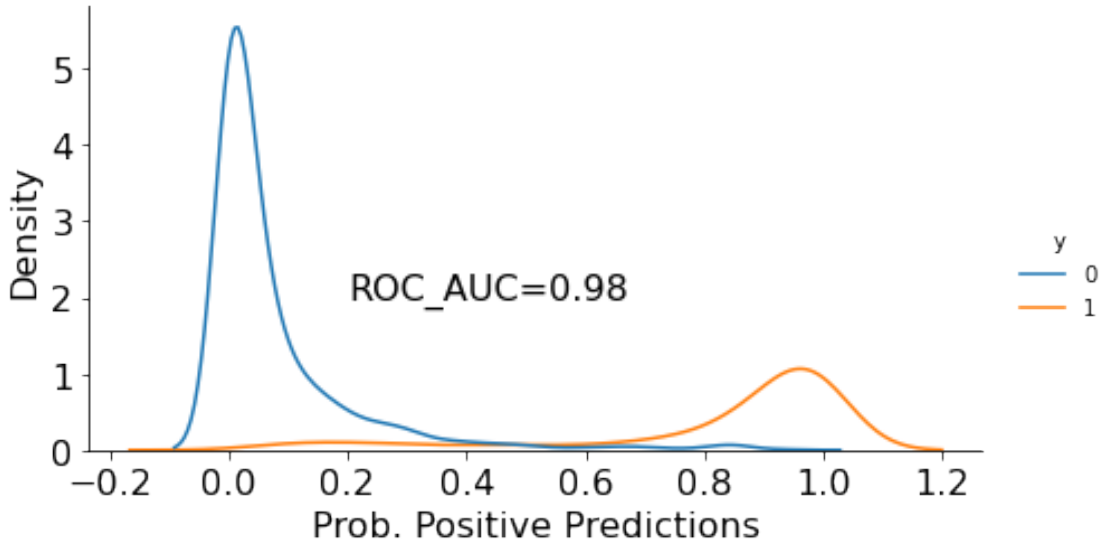
```
weighted avg       0.94      0.94      0.94      2800
```

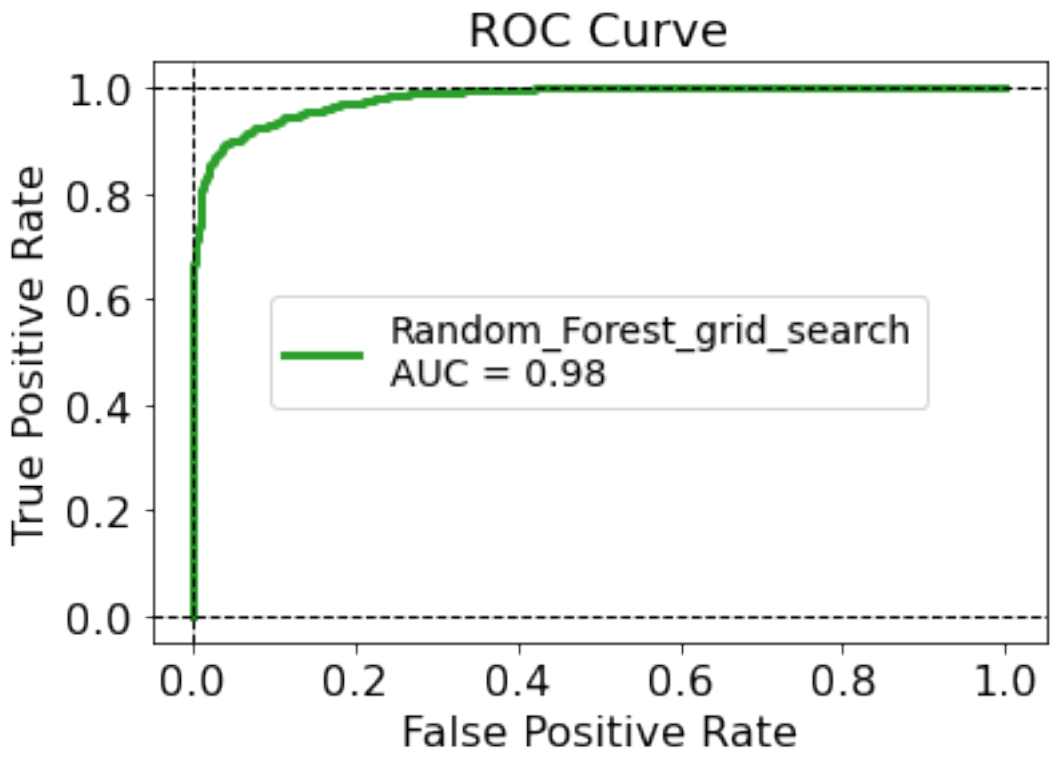




[37]:
```
model_rf_gs = GridSearchCV(RandomForestClassifier(),
                           param_grid={'max_depth':[10, 11, 12]},
                           scoring ='f1',
                           verbose=1)

mod7 = Model_training(model_rf_gs,
                      X_train2, y_train2, X_test2, y_test2,
                      "Random_Forest_grid_search")

mod_tr, _= mod7.print_metrics()
mod7.displot()
mod7.plot_roc_curve()
```

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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   16.4s finished

Accuracy = 94.11% F1 Score= 89.69%
Precision=93.49% Recall= 86.19%
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1967
           1       0.93      0.86      0.90       833

    accuracy                           0.94      2800
   macro avg       0.94      0.92      0.93      2800
weighted avg       0.94      0.94      0.94      2800
```





[38]:
```
mod_tr.best_params_
```

[38]:
```
{'max_depth': 12}
```

remove ALERT_TRGR_CD from features

[39]:
```
X_train2.head(1)
```

[39]:
```
      TRAN_AMT  ACCT_PRE_TRAN_AVAIL_BAL  CUST_AGE  OPEN_ACCT_CT  WF_dvc_age  \
2413    487.93                  3714.91        43             5        1037

      PWD_UPDT_DAYS  PH_NUM_UPDT_DAYS  TRAN_DAYS  PH_NUM_PWD_DAYS  \
2413          12146               347      12146           -11799

      ALERT_TRGR_CD_1  ALERT_TRGR_CD_2  CUST_STATE_1  CUST_STATE_2  \
2413                1                0             1             0

      CUST_STATE_3  CUST_STATE_4  CUST_STATE_5
2413             0             0             0
```

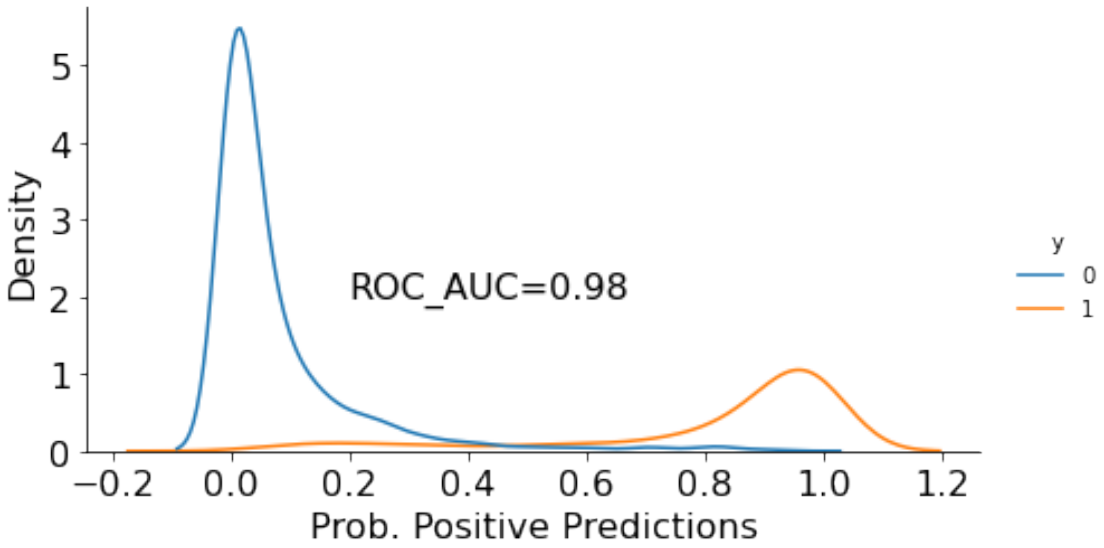
[40]:
```
cols = [c for c in list(X_train2.columns) if c not in␣
↪["ALERT_TRGR_CD_1","ALERT_TRGR_CD_2"]]
cols
```

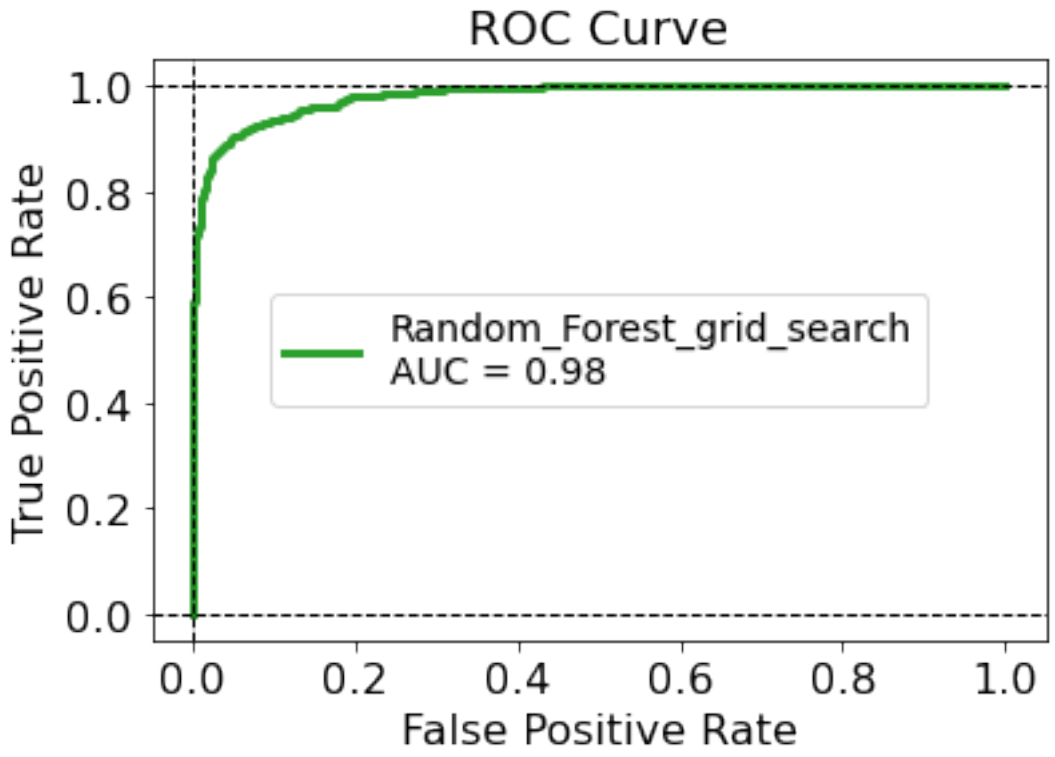
[40]:
```
['TRAN_AMT',
 'ACCT_PRE_TRAN_AVAIL_BAL',
 'CUST_AGE',
 'OPEN_ACCT_CT',
 'WF_dvc_age',
 'PWD_UPDT_DAYS',
 'PH_NUM_UPDT_DAYS',
 'TRAN_DAYS',
 'PH_NUM_PWD_DAYS',
 'CUST_STATE_1',
 'CUST_STATE_2',
 'CUST_STATE_3',
 'CUST_STATE_4',
 'CUST_STATE_5']
```

[41]:
```
X_train3 = X_train2[cols]
X_test3  = X_test2[cols]
y_train3 = y_train2
y_test3  = y_test2
```

[42]:
```
model_rf_gs = GridSearchCV(RandomForestClassifier(),
                           param_grid={'max_depth':[11, 12, 13, 14, 15]},
                           scoring ='f1',
                           verbose=1)

mod8 = Model_training(model_rf_gs,
                      X_train3, y_train3, X_test3, y_test3,
                      "Random_Forest_grid_search")

mod_tr, _= mod8.print_metrics()
mod8.displot()
mod8.plot_roc_curve()
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:   26.7s finished

Accuracy = 94.07% F1 Score= 89.68%
Precision=93.03% Recall= 86.55%
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1967
           1       0.93      0.87      0.90       833

    accuracy                           0.94      2800
   macro avg       0.94      0.92      0.93      2800
weighted avg       0.94      0.94      0.94      2800
```

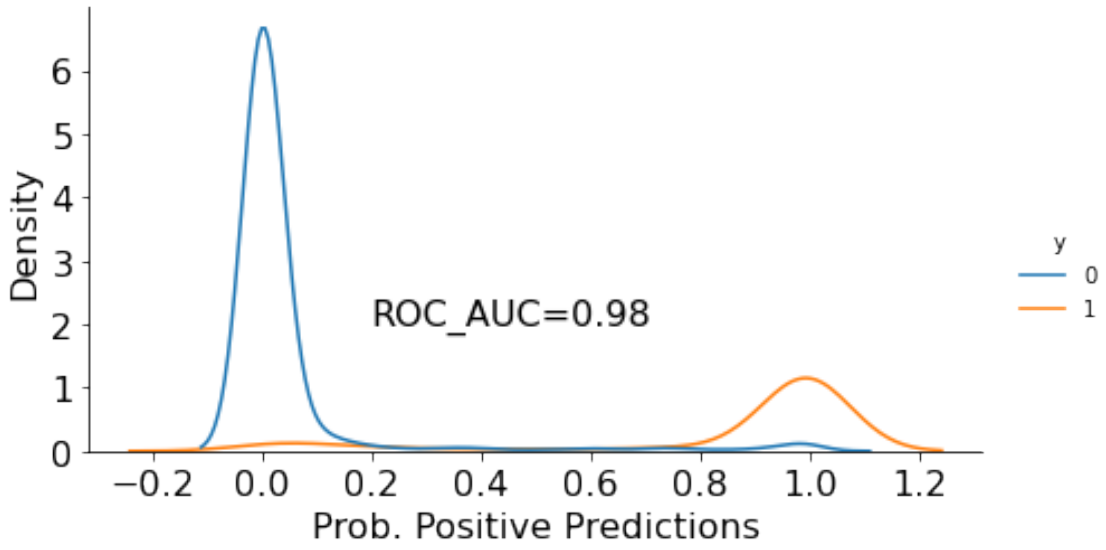




[43]:
```
from sklearn.svm import SVC
```

[44]:
```
svm = SVC(gamma='auto')
mod = Model_training(xgb_gs, X_train1, y_train1, X_test1, y_test1)
mod_tr, _= mod.print_metrics()
mod.displot()
```

```
Accuracy = 92.93% F1 Score= 87.78%
Precision=90.34% Recall= 85.35%
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      1967
           1       0.90      0.85      0.88       833

    accuracy                           0.93      2800
   macro avg       0.92      0.91      0.91      2800
weighted avg       0.93      0.93      0.93      2800
```



[45]:
```
def train_svm_gs():
    svm_gs = GridSearchCV(SVC(),
                          param_grid={'C':[0.1, 0.5, 1.0],
                                      'kernel':['poly', 'rbf'],
                                      'gamma': ['scale', 'auto']},
                          scoring = 'f1',
                          verbose = 1 )

    mod = Model_training(svm_gs, X_train1, y_train1, X_test1, y_test1)
    mod_tr, _= mod.print_metrics()
    mod.displot()
```

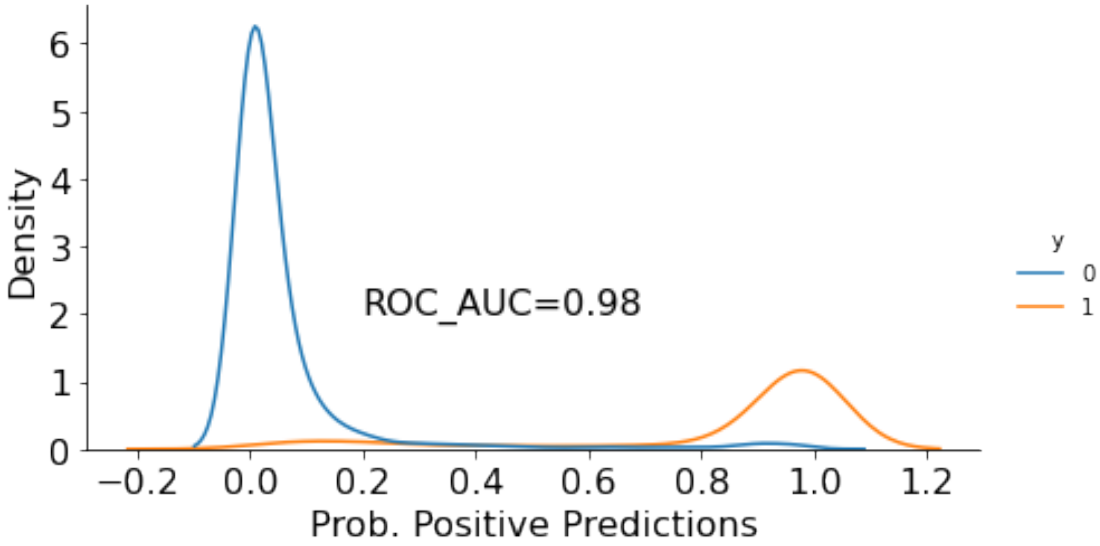
```
#this takes a little long time so think before running
#train_sum_gs()
```

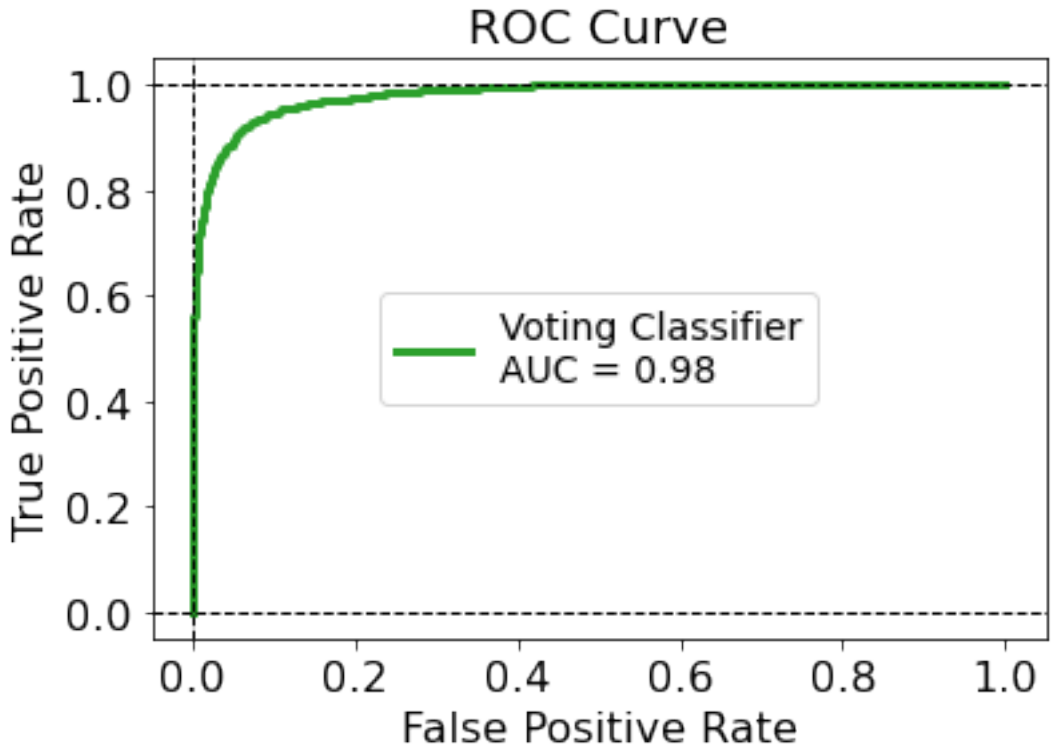
1.8 Voting Classifier

[46]:
```
#clf1 = LogisticRegression(multi_class='multinomial', random_state=1)

clf2 = RandomForestClassifier(max_depth=10,
                              random_state=8848)

clf3 = XGBClassifier(verbosity=1,
                     max_depth=13,
                     eval_metric = "logloss")

clf_voting = VotingClassifier(
    estimators=[('rf', clf2), ('xgb', clf3)],
    voting='soft')
```

[47]:
```
mod8 = Model_training(clf_voting, X_train1, y_train1, X_test1, y_test1, "Voting␣
↪Classifier")
mod_tr, _= mod8.print_metrics()
mod8.displot()
mod8.plot_roc_curve()
#mod8.feature_importance()
```

```
Accuracy = 93.39% F1 Score= 88.44%
Precision=92.19% Recall= 84.99%
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      1967
           1       0.92      0.85      0.88       833

    accuracy                           0.93      2800
   macro avg       0.93      0.91      0.92      2800
weighted avg       0.93      0.93      0.93      2800
```

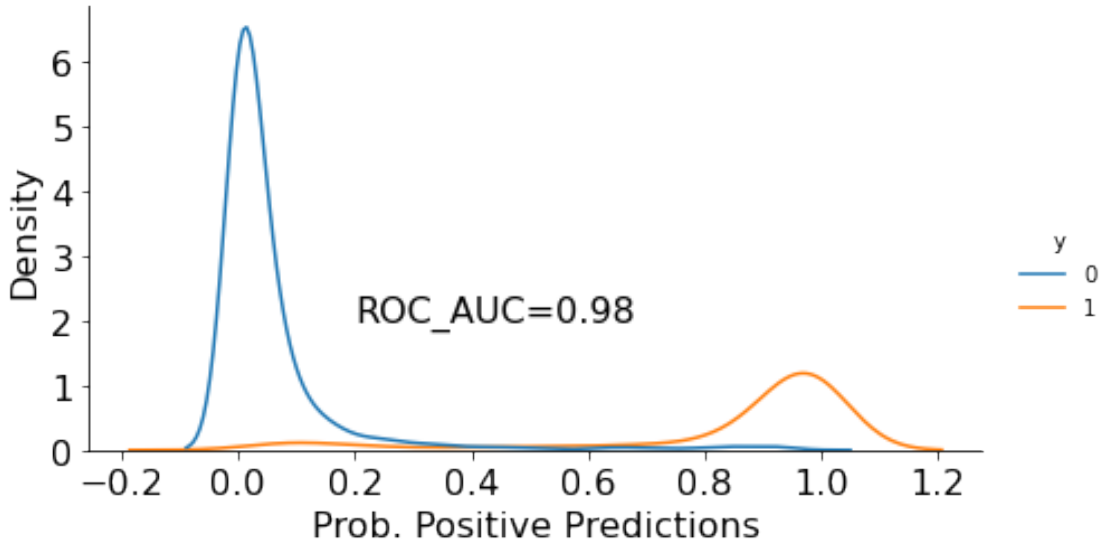


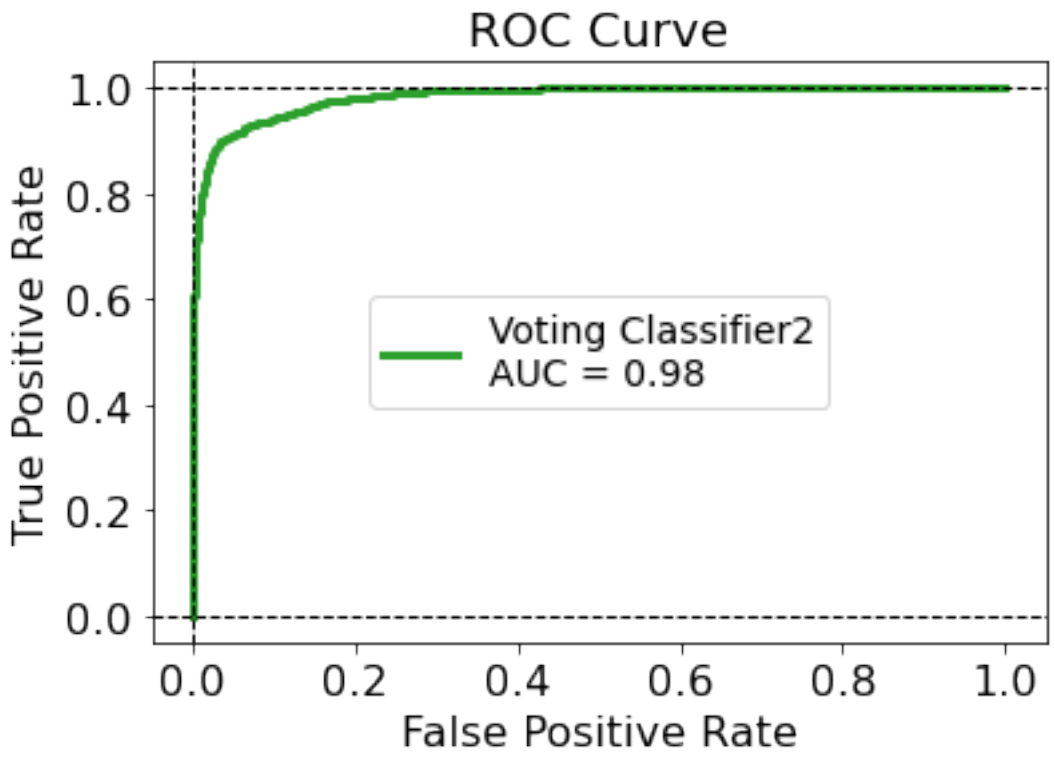


[48]:
```
mod8 = Model_training(clf_voting, X_train2, y_train2, X_test2, y_test2, "Voting␣
↪Classifier2")
mod_tr, _= mod8.print_metrics()
mod8.displot()
```

```
mod8.plot_roc_curve()
#mod8.feature_importance()
```

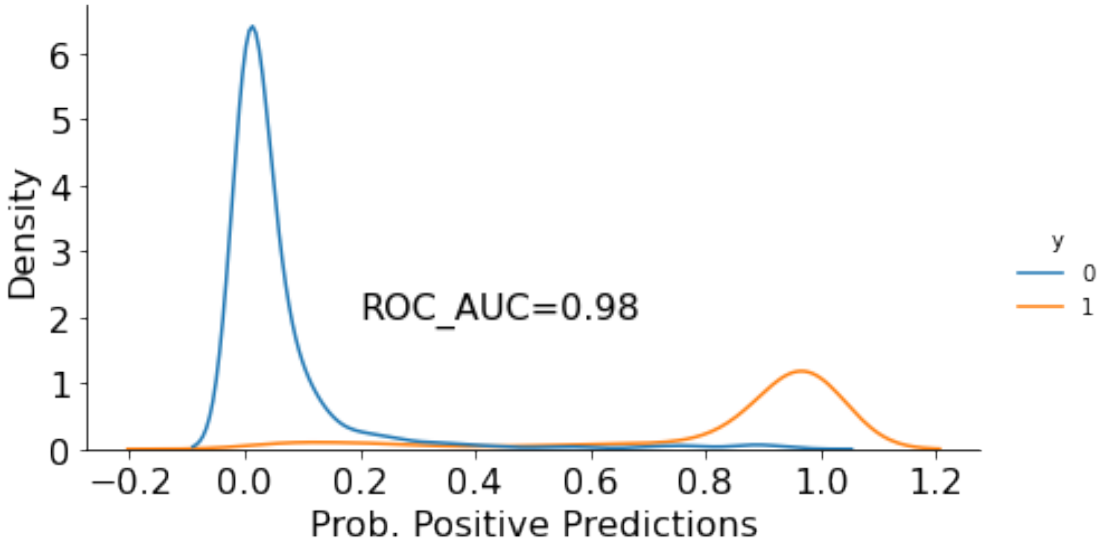
```
Accuracy = 94.61% F1 Score= 90.64%
Precision=93.72% Recall= 87.76%
              precision    recall  f1-score   support

           0       0.95      0.98      0.96      1967
           1       0.94      0.88      0.91       833

    accuracy                           0.95      2800
   macro avg       0.94      0.93      0.93      2800
weighted avg       0.95      0.95      0.95      2800
```

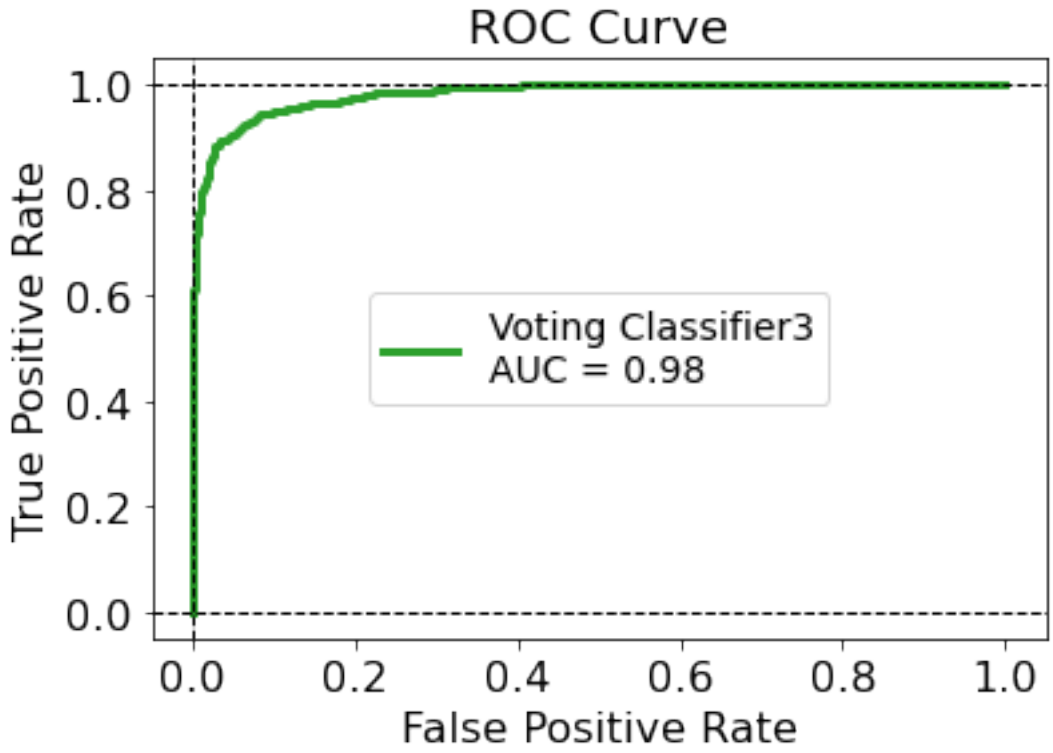




[49]:
```
mod8 = Model_training(clf_voting, X_train3, y_train3, X_test3, y_test3, "Voting␣
↪Classifier3")
mod_tr, _= mod8.print_metrics()
mod8.displot()
mod8.plot_roc_curve()
#mod8.feature_importance()
```

```
Accuracy = 94.39% F1 Score= 90.28%
Precision=93.22% Recall= 87.52%
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1967
           1       0.93      0.88      0.90       833

    accuracy                           0.94      2800
   macro avg       0.94      0.92      0.93      2800
weighted avg       0.94      0.94      0.94      2800
```





[50]:
```
X_test2.head()
```

[50]:

	TRAN_AMT	ACCT_PRE_TRAN_AVAIL_BAL	CUST_AGE	OPEN_ACCT_CT	WF_dvc_age	\
3032	494.73	2542.73	71	3	248	
4838	489.42	3324.74	46	3	0	
7117	463.06	242.04	69	3	504	
9795	493.09	4828.93	57	2	0	
5640	0.01	0.00	68	3	181	

	PWD_UPDT_DAYS	PH_NUM_UPDT_DAYS	TRAN_DAYS	PH_NUM_PWD_DAYS	\
3032	-49	1647	1647	1696	
4838	255	554	10419	299	
7117	1083	7673	7673	6590	
9795	-87	6293	6293	6380	
5640	1445	7443	7443	5998	

	ALERT_TRGR_CD_1	ALERT_TRGR_CD_2	CUST_STATE_1	CUST_STATE_2	\
3032	0	1	0	0	
4838	1	0	0	0	
7117	0	1	0	0	
9795	1	0	0	0	
5640	0	1	0	0	

	CUST_STATE_3	CUST_STATE_4	CUST_STATE_5
3032	0	0	1
4838	0	1	0
7117	0	0	1
9795	0	0	1
5640	0	0	1

[ ]:

[ ]: