2_Modeling on balanced dataset

0.1 Overview

Previous modeling on imbalanced dataset reached a highest performance on Logistic regression with Lasso penalty (C=1) and cut-off threshold 0.3.

- The testing performance are: F1: 68.43, Precision: 75.75, Recall: 62.39, AUC: 78.46, Accuracy: 87.63.

This notebook focused on training models on under-sampled and SMOTE over-sampled dataset. When do model selection, all the candidate models on under-sampled dataset performed worse than models on original dataset, all candidate models performed better on SMOTE over-sampled dataset. Thus, further parameter tuning process was performed on over-sampled dataset. To summarize, the best model was Logistic regression with Lasso penalty (C=10) and cut-off threshold 0.5.

- The new testing performance are: F1: 68.17, Precision: 85.68, Recall: 56.60, AUC: 76.57, Accuracy: 85.84.

According to feature importance, people with high last_fico_range_high and annual_inc are more like to fully pay their loans, applicants with debt_settlement_flag Y are more likely to charge off on loans which is same to feature importance on imbalanced dataset.

All in all, models on balanced dataset performed similar to models on original dataset. Therefore, it is more efficient to model on the original imbalanced dataset for the Lending Club risk loan prediction problem.

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

def warn(*args, **kwargs): pass
import warnings
warnings.warn = warn

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

from sklearn import model_selection
from sklearn.metrics import roc_auc_score
```

```
from sklearn.linear_model import LogisticRegression
#from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.metrics import confusion_matrix
```

0.1.1 model selection function

[2]:
```
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('CART', DecisionTreeClassifier(max_depth=10)))
models.append(('RF', RandomForestClassifier(max_depth=10)))#slow
models.append(('NB', GaussianNB()))
#models.append(('SVM', SVC())) slow

def models_performance(models, X_train,y_train):
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=5, random_state=seed)
        cv_results = model_selection.cross_val_score(model, X_train, y_train,
                                                     cv=kfold, scoring='f1')
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)

    fig = plt.figure()
    fig.suptitle('Algorithm Comparison')
    ax = fig.add_subplot(111)
    plt.boxplot(results,showmeans=True)
    ax.set_xticklabels(names)
    plt.show()
```

0.1.2 model evaluation function

[3]:
```
def plot_confusion_matrix(pred, act, title):
    #import seaborn as sns
    cnf_matrix = confusion_matrix(pred, act, labels=[0, 1])
    precision = cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1])
```

```
    recall = cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])
    f1 = 2*precision*recall/(precision+recall)
    aucRoc = roc_auc_score(pred, act)
    acc = (cnf_matrix[0,0] + cnf_matrix[1,1])/cnf_matrix.sum()
    print('')
    print(title)

    sns.set(font_scale=1.4) # for label size
    sns.heatmap(cnf_matrix, annot=True, annot_kws={"size": 16},fmt='g') # font␣
↪size
    #plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

    print('Mesurements:')
    method_dict = {'F1': '{:.2f}'.format(f1*100),
                'Precision': '{:.2f}'.format(precision*100),
                'Recall': '{:.2f}'.format(recall*100),
                'AUC': '{:.2f}'.format(aucRoc*100),
                'Accuracy': '{:.2f}'.format(acc*100)
               }
    return pd.DataFrame(method_dict,index=['values'])
```

0.1.3 parameter tuning: Logistic regression C value

[4]:
```
def c_scores(X_train, y_train):
    #penalty range
    c_param_range = [0.01, 0.1, 1, 10, 100]

    regularization = []
    results = []
    for c_param in c_param_range:

        lr = LogisticRegression(C = c_param, penalty='l1',solver='saga')
        kfold = model_selection.KFold(n_splits=5, random_state=seed)
        cv_results = model_selection.cross_val_score(lr, X_train, y_train,␣
↪cv=kfold, scoring='f1')

        results.append(cv_results)
        regularization.append(c_param)
        print('regularization:', c_param, 'f1: ', cv_results.mean(), 'std: ',␣
↪cv_results.std())

    fig = plt.figure()
    fig.suptitle('F1 score with different C value')
```

```
    ax = fig.add_subplot(111)
    plt.boxplot(results)
    ax.set_xticklabels(regularization)
    plt.show()
```

0.1.4 threshold selection function

[5]:
```
def best_threshod(y_pred_proba, y_train, threshold_range):
    Precision = []
    Recall = []
    F1 = []
    AUC = []
    Accuracy = []
    for i  in threshold_range:
        y_pred = y_pred_proba[:,1] > i
        cnf_matrix = confusion_matrix(y_pred, y_train)

        precision = cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])
        recall = cnf_matrix[1,1]/(cnf_matrix[0,1]+cnf_matrix[1,1])
        f1 = 2*precision*recall/(precision+recall)
        aucRoc = roc_auc_score(y_pred, y_train)
        acc = (cnf_matrix[0,0] + cnf_matrix[1,1])/cnf_matrix.sum()

        F1.append(f1)
        Precision.append(precision)
        Recall.append(recall)
        AUC.append(aucRoc)
        Accuracy.append(acc)

    method_dict = {'Threshold':threshold_range,
                   'F1': F1,
                   'Precision': Precision,
                   'Recall': Recall,
                   'AUC': AUC,
                   'Accuracy':Accuracy
                  }
    df = pd.DataFrame(method_dict)
    df2 = df.melt('Threshold', var_name='cols',  value_name='vals')

    sns.lineplot(x="Threshold", y="vals", hue='cols',data=df2)
    plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    print('Best threshold based on highest F1 score:')
    return df[df.F1 == df.F1.max()]
```

0.1.5 Data preparation

[6]:
```
X_train = pd.read_csv('x_train.csv')
y_train = pd.read_csv('y_train.csv')
print('===training set===')
print(y_train.value_counts())

X_test = pd.read_csv('x_test.csv')
y_test = pd.read_csv('y_test.csv')
print('')
print('===testing set===')
print(y_test.value_counts())

rus = RandomUnderSampler(random_state=0)
X_undersampled, y_undersampled = rus.fit_resample(X_train, y_train)
y_undersampled.value_counts()
print('')
print('===under-sampled training set===')
print(y_undersampled.value_counts())

oversample = SMOTE()
X_oversampled, y_oversampled = oversample.fit_resample(X_train, y_train)
y_oversampled.value_counts()
print('')
print('===over-sampled training set===')
print(y_oversampled.value_counts())
```

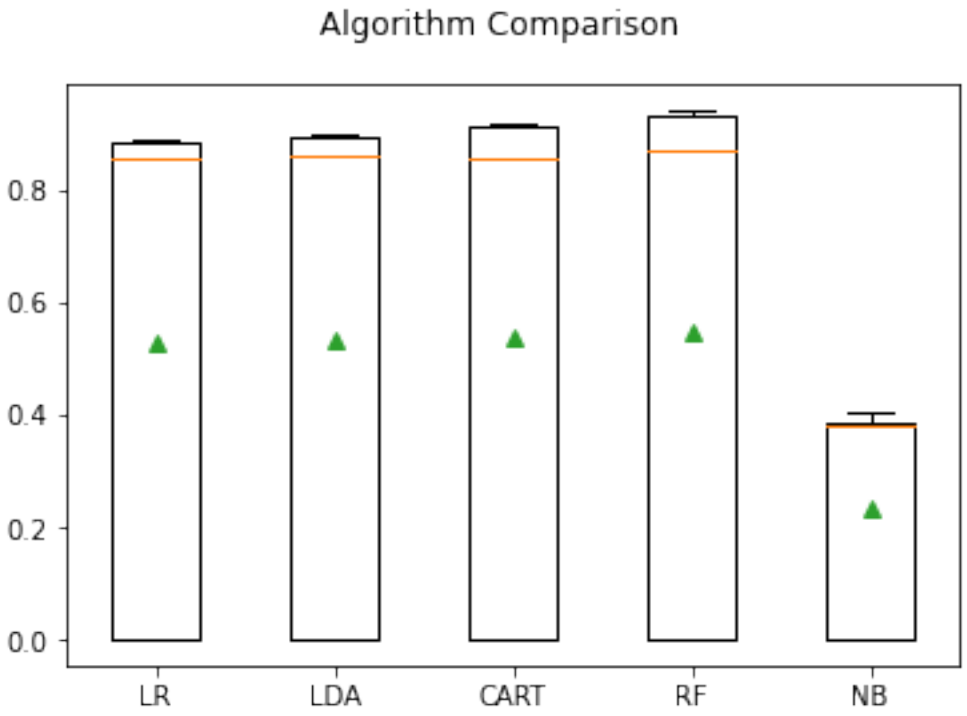
```
===training set===
loan_status
0              127625
1               27453
dtype: int64

===testing set===
loan_status
0              63813
1              13726
dtype: int64

===under-sampled training set===
loan_status
1              27453
0              27453
dtype: int64

===over-sampled training set===
loan_status
1              127625
```

```
0               127625
dtype: int64
```

0.1.6 model selection on under-sampled dataset

[7]:
```
models_performance(models, X_undersampled, y_undersampled)
```

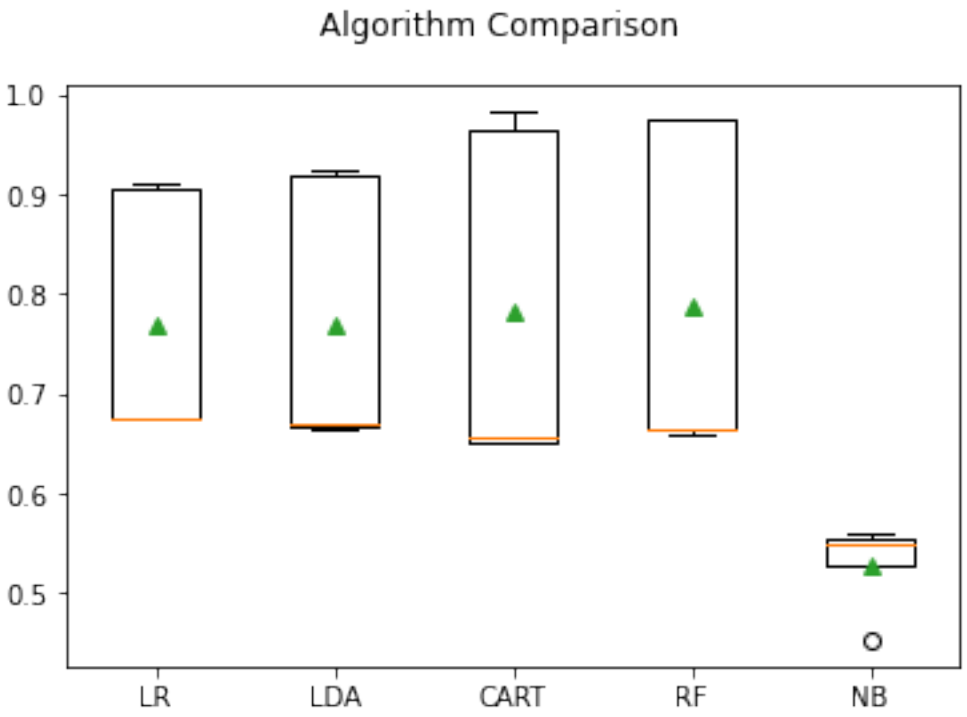
```
LR: 0.525722 (0.429383)
LDA: 0.531164 (0.433902)
CART: 0.536779 (0.438799)
RF: 0.548546 (0.448559)
NB: 0.233560 (0.190923)
```



0.1.7 model selection on over-sampled dataset

[8]:
```
models_performance(models, X_oversampled, y_oversampled)
```

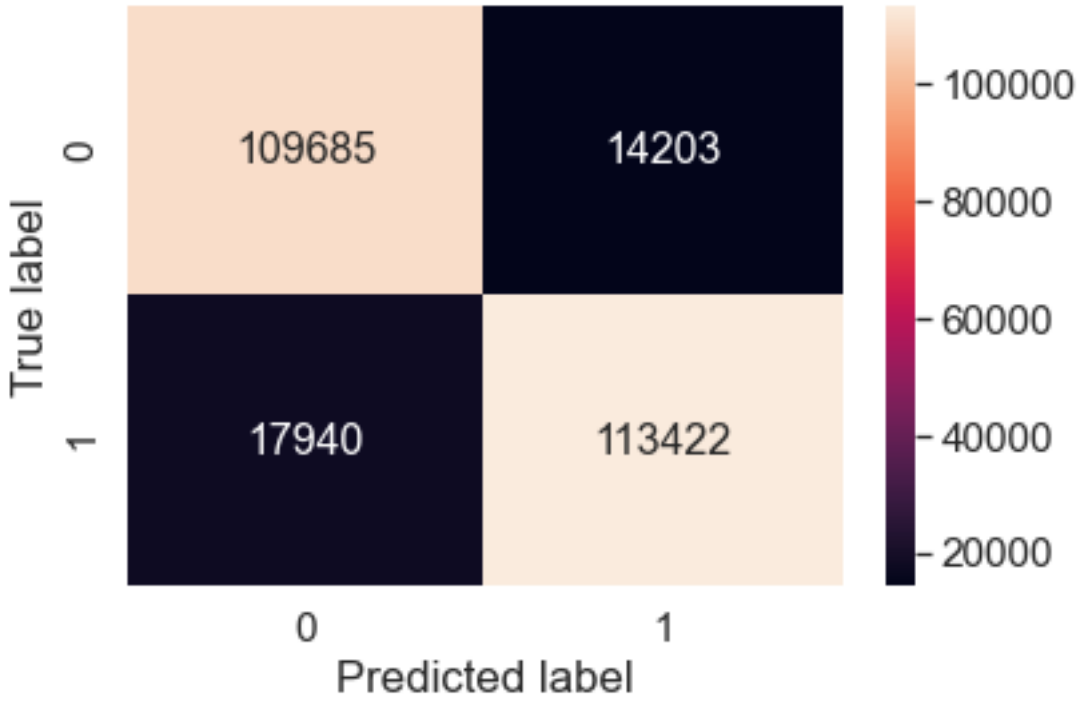
```
LR: 0.768125 (0.114060)
LDA: 0.768652 (0.124790)
CART: 0.781673 (0.156951)
RF: 0.787272 (0.153333)
NB: 0.528212 (0.039300)
```



0.1.8 ===> Logistic regression looks the best model

[9]:
```
lr = LogisticRegression().fit(X_oversampled, y_oversampled)

y_pred = lr.predict(X_oversampled)
plot_confusion_matrix(y_pred, y_oversampled, 'Logistic Regression: Train␣
 ↪Performance on balanced data')
```

```
Logistic Regression: Train Performance on balanced data
```

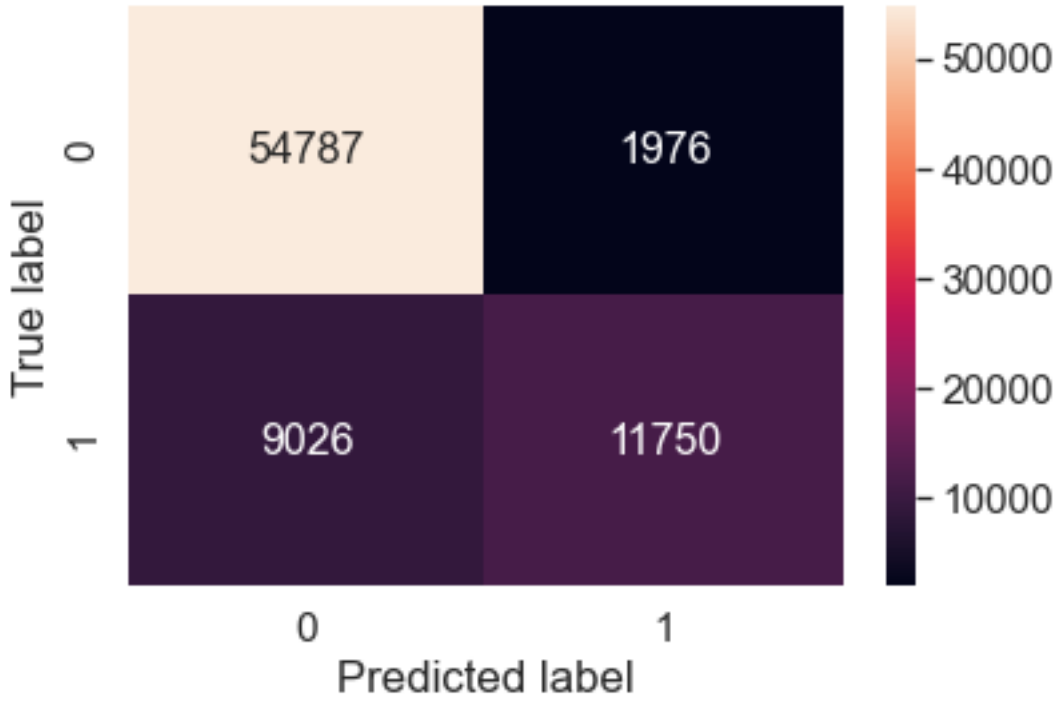


Mesurements:

[9]:
	F1	Precision	Recall	AUC	Accuracy
values	87.59	88.87	86.34	87.44	87.41

[10]:
```
y_test_pred = lr.predict(X_test)
plot_confusion_matrix(y_test_pred, y_test,
                      'Logistic Regression: Test Performance after training on␣
 ↪balanced data')
```

Logistic Regression: Test Performance after training on balanced data



Mesurements:

[10]:

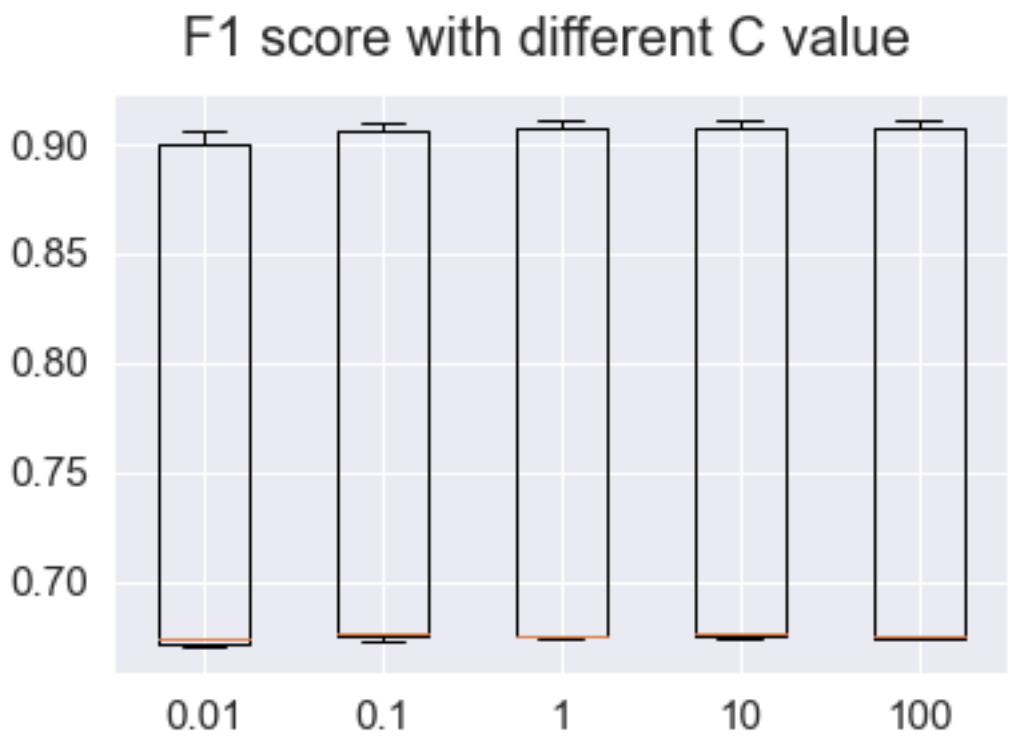
	F1	Precision	Recall	AUC	Accuracy
values	68.11	85.60	56.56	76.54	85.81

0.1.9 test on logistic regression f1 score: 68.11

0.1.10 tuning C value

[11]:
```
c_scores(X_oversampled, y_oversampled)
```

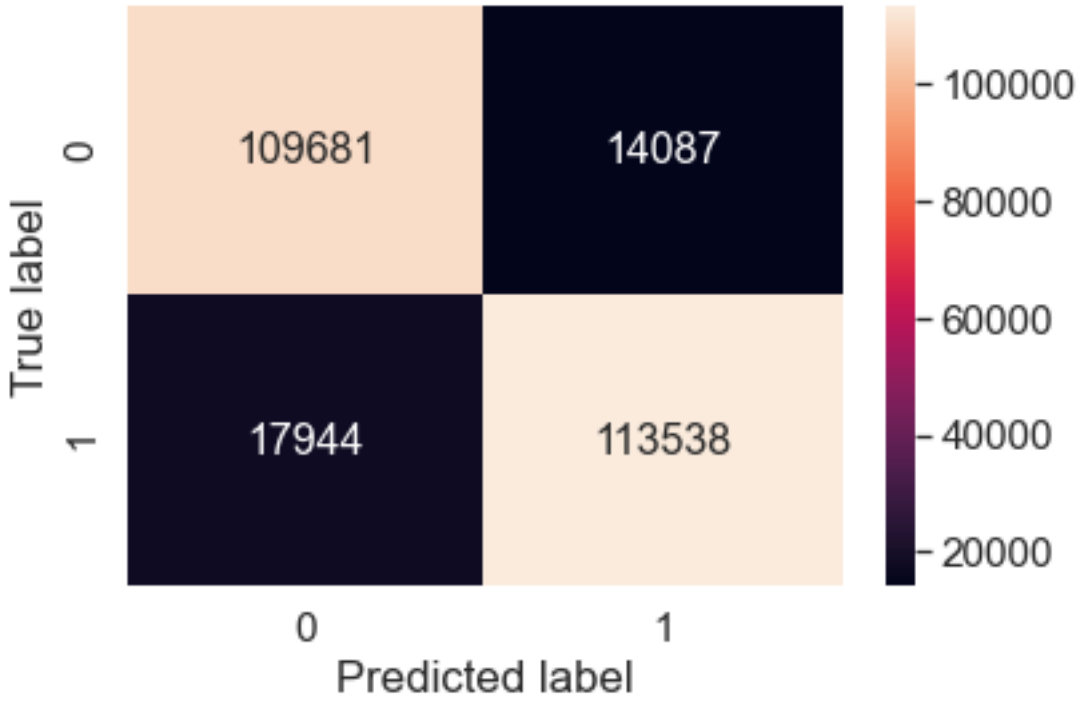
```
regularization: 0.01 f1:  0.764366590590909 std:  0.1132214772216187
regularization: 0.1 f1:  0.7681409699492686 std:  0.11424666632705474
regularization: 1 f1:  0.7685830779331706 std:  0.1147065229751978
regularization: 10 f1:  0.7686166966993528 std:  0.11466515872711594
regularization: 100 f1:  0.7685731212684284 std:  0.11471506133701083
```



0.1.11 best c value 10

[12]:
```
lr = LogisticRegression(C = 10, penalty='l1',solver='saga').fit(X_oversampled,␣
 ↪y_oversampled)

y_pred = lr.predict(X_oversampled)
plot_confusion_matrix(y_pred, y_oversampled,
                      'Logistic Regression(C=10,L1): Train Performance on␣
 ↪balanced data')
```

```
Logistic Regression(C=10,L1): Train Performance on balanced data
```



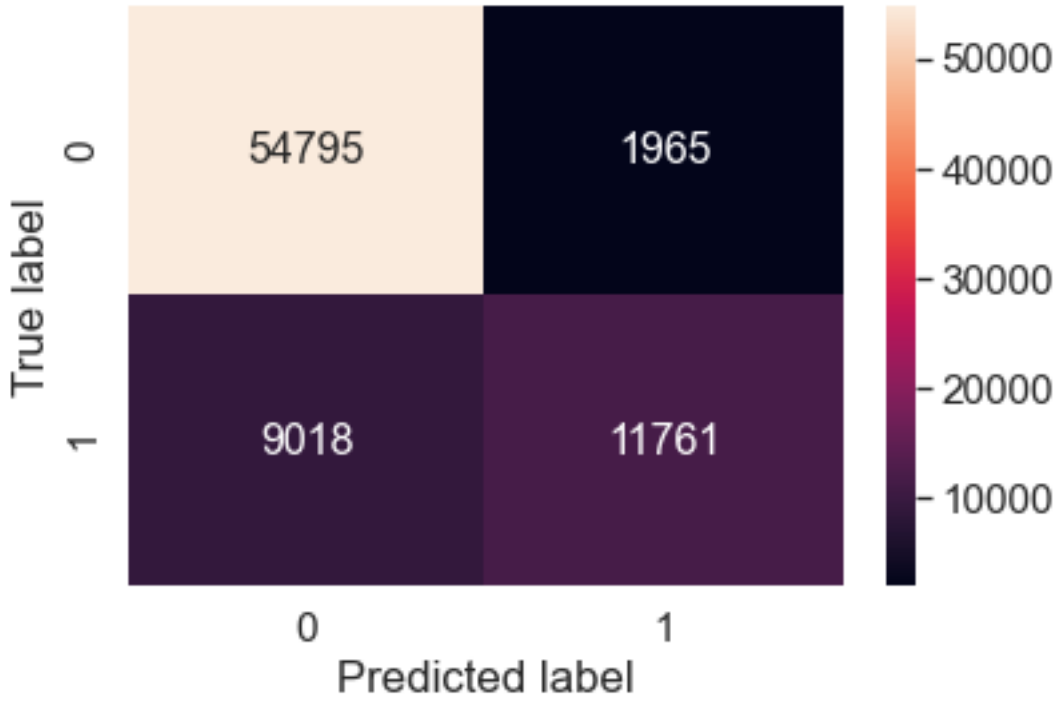
Mesurements:

[12]:

	F1	Precision	Recall	AUC	Accuracy
values	87.64	88.96	86.35	87.49	87.45

[13]:
```
y_test_pred = lr.predict(X_test)
plot_confusion_matrix(y_test_pred, y_test,
                      'Logistic Regression(C=10,L1): Test Performance after␣
 ↪training on balanced data')
```

Logistic Regression(C=10,L1): Test Performance after training on balanced data



Mesurements:

[13]:

	F1	Precision	Recall	AUC	Accuracy
values	68.17	85.68	56.60	76.57	85.84

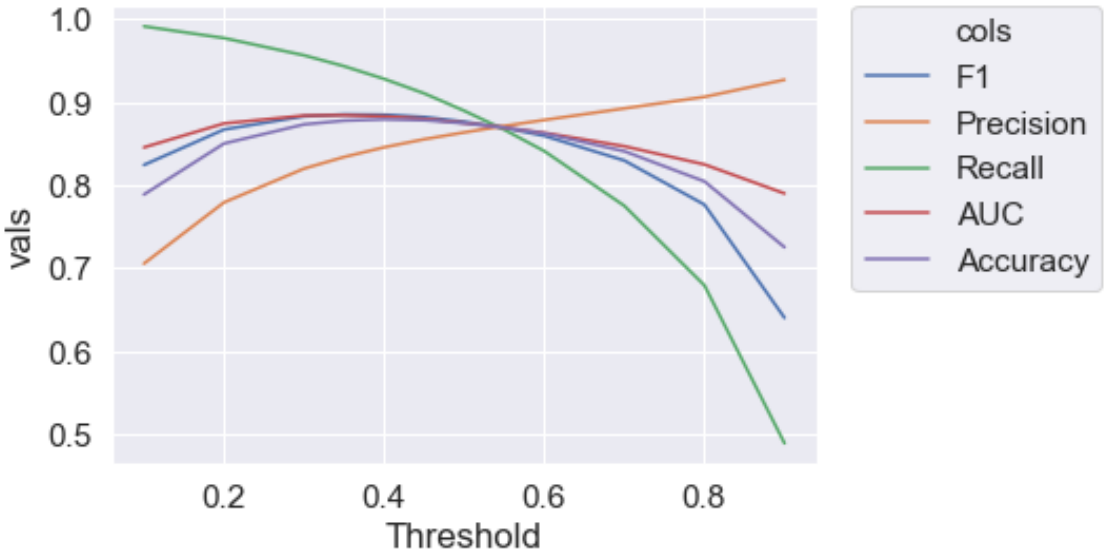
0.1.12 test on logistic regression f1 score: 68.17

[14]:
```
y_pred_proba = lr.predict_proba(X_oversampled)

threshold_range = [0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.7, 0.8, 0.9]
best_threshod(y_pred_proba, y_oversampled, threshold_range)
```

Best threshold based on highest F1 score:

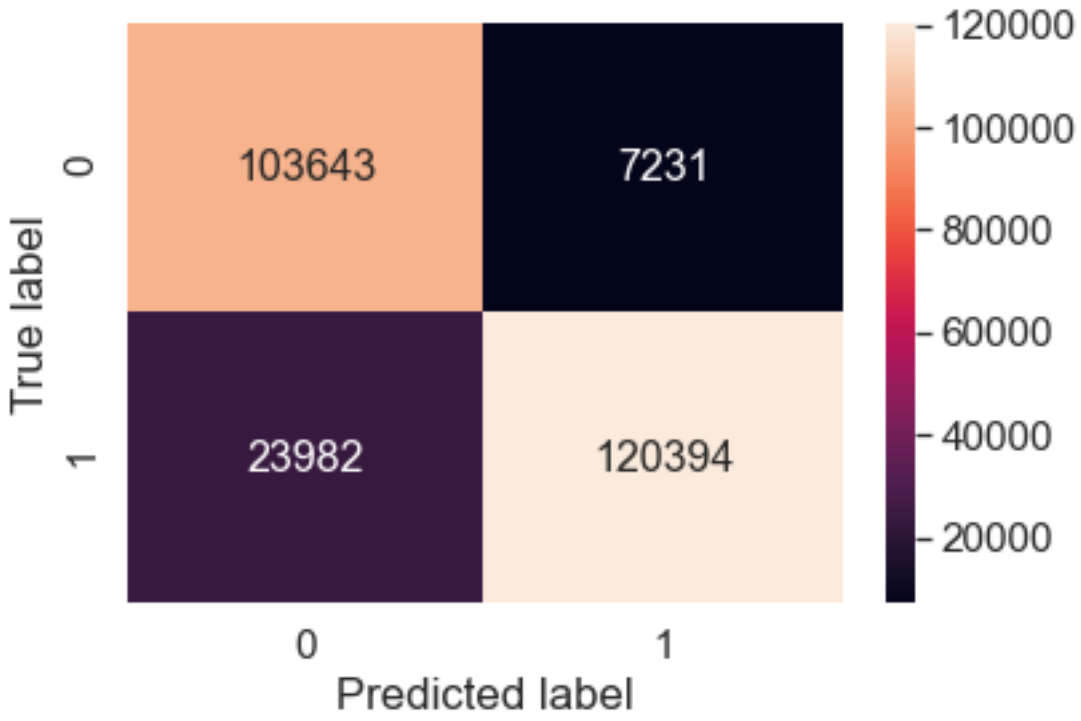
[14]:

	Threshold	F1	Precision	Recall	AUC	Accuracy
3	0.35	0.885247	0.833892	0.943342	0.884337	0.877716



[15]:
```
threshold = 0.35
y_proba = lr.predict_proba(X_oversampled)
y_pred = y_proba[:,1] > threshold

plot_confusion_matrix(y_pred, y_oversampled,
                      'Logistic Regression(C=10, L1, Threshold=0.35): Train␣
 ↪performance on balanced data')
```

```
Logistic Regression(C=10, L1, Threshold=0.35): Train performance on balanced
data
```



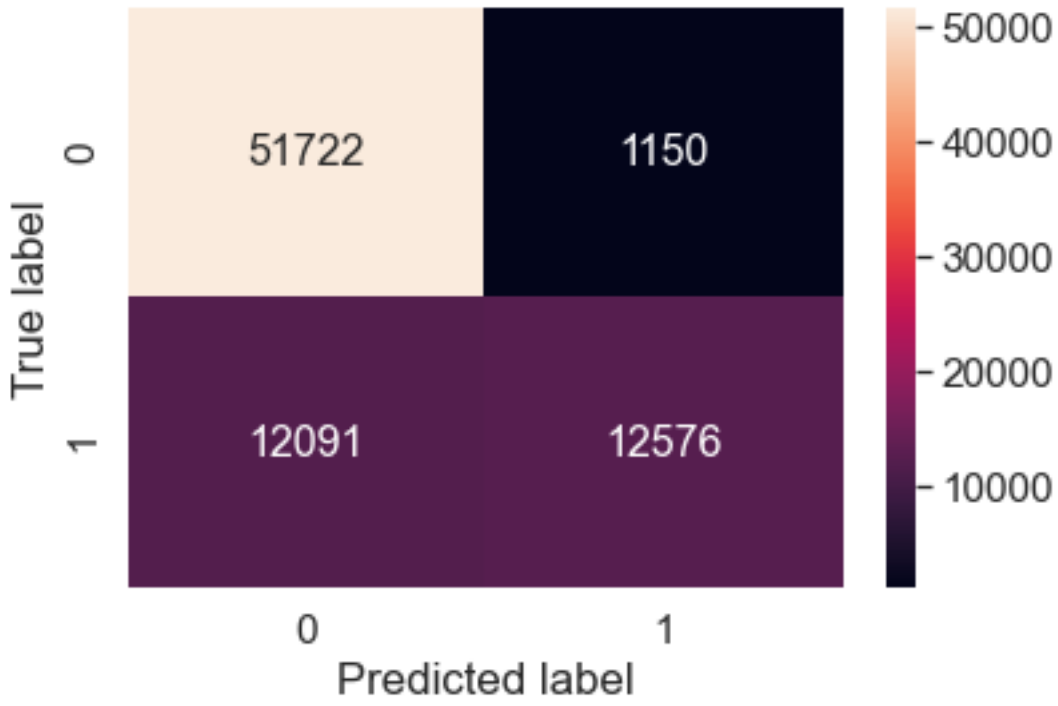
Mesurements:

[15]:

	F1	Precision	Recall	AUC	Accuracy
values	88.52	94.33	83.39	88.43	87.77

[18]:
```
y_test_proba = lr.predict_proba(X_test)
y_test_pred = y_test_proba[:,1] > 0.35

plot_confusion_matrix(y_test_pred, y_test,
                      'Logistic Regression(C=10, Threshold=0.35): Test␣
 ↪Performance after training on balanced data')
```

Logistic Regression(C=10, Threshold=0.35): Test Performance after training on balanced data



Mesurements:

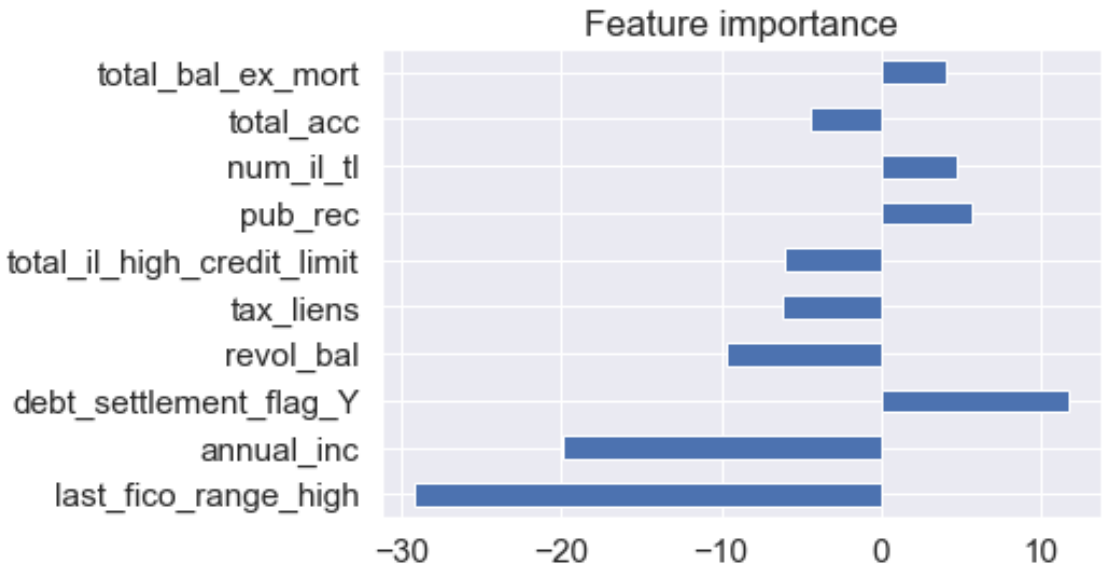
[18]:

	F1	Precision	Recall	AUC	Accuracy
values	65.51	91.62	50.98	74.40	82.92

0.1.13 test on logistic regression f1 score: 65.51

[19]:
```
model = LogisticRegression(C = 10, penalty='l1',solver='saga').
 ↪fit(X_oversampled, y_oversampled)

feat_importances = pd.Series(model.coef_[0], index=X_train.columns)
important = list(feat_importances.abs().nlargest(10).index)
feat_importances[important].plot(kind='barh', title = 'Feature importance')
```

[19]: <AxesSubplot:title={'center':'Feature importance'}>


Feature importance
total_bal_ex_mort
total_acc
num_il_tl
pub_rec
total_il_high_credit_limit
tax_liens
revol_bal
debt_settlement_flag_Y
annual_inc
last_fico_range_high
−30
−20
−10
0
10

[ ]:

[ ]:

[ ]:

[ ]: