modelling_log_reg

September 23, 2019

[109]:
```
import pickle

%matplotlib inline
import matplotlib.pyplot as plt

import numpy as np

import pandas as pd
pd.options.display.max_columns = None

from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSplit,␣
↪learning_curve, GridSearchCV, train_test_split
from sklearn.metrics import classification_report, confusion_matrix,␣
↪precision_score, recall_score
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.utils import resample

from imblearn.over_sampling import SMOTE
```

[2]:
```
# Load preprocessed dataset

PREPROCESSED_DF_PATH = 'preprocessed_applications_df.dataframe.pd'
main_df = pickle.load( open(PREPROCESSED_DF_PATH, 'rb') )
```

[3]:
```
# Remove redundant features

target_ids = main_df['appl_id']
target_labels = main_df['df']

features_to_drop = [
    'appl_id', 'client_id',
    'app_crtime', 'birth', 'pass_bdate', 'lived_since',␣
↪'is_same_reg_lived_since', 'jobsworksince',
    'df'  # target feature
```

```
]
main_df = main_df.drop( features_to_drop, axis=1 )
```

[4]:
```
# Encode several features

features_to_encode = [
    'gender',
    'top_browser', 'top_platform',
    'total_devices_cnt',
    'visit_top_dayofweek', 'visit_top_dayhour',
    'binned_visit_top_dayhour',
    'binned_fam_status', 'binned_quantity_child', 'binned_max_age_child',
    'binned_property',
    'binned_region', 'binned_region_reg',
    'binned_work_experience', 'binned_empl_state', 'binned_empl_type',␣
 ↪'binned_empl_worker_count',
    'binned_education_area', 'binned_education',
    'binned_days_from_birth', 'binned_days_from_passbdate',
    'app_month_num'
]

for feature_name in features_to_encode:
    main_df[feature_name] = main_df[feature_name].astype('category')

main_encoded_df = pd.get_dummies(
    main_df, columns=features_to_encode, drop_first=True
)
main_df.shape, main_encoded_df.shape
```

[4]: ((8068, 55), (8068, 127))

[5]:
```
def draw_learning_curve(estimator, X_tr, y_tr):
    train_sizes, train_scores, val_scores = learning_curve(
        estimator, X_tr, y_tr, train_sizes=np.linspace(0.1, 1.0, 5), cv=3
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    val_scores_mean = np.mean(val_scores, axis=1)
    val_scores_std = np.std(val_scores, axis=1)
    plt.grid()
    plt.fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1, color="r"
    )
    plt.fill_between(
        train_sizes,
```

```
        val_scores_mean - val_scores_std,
        val_scores_mean + val_scores_std,
        alpha=0.1, color="g"
    )
    plt.plot(
        train_sizes,
        train_scores_mean,
        'o-', color="r", label="Training score"
    )
    plt.plot(
        train_sizes,
        val_scores_mean,
        'o-', color="g", label="Cross-validation score"
    )
    plt.legend(loc="best")
    plt.show()
```

[6]:
```
# Scale data logistic regression

standard_scl = StandardScaler()

main_df = standard_scl.fit_transform( main_df )
```

[7]:
```
# Prepare train df and train target labels

train_labels = target_labels[ target_labels.isnull() == False ]
display('train labels shape: {0}'.format(train_labels.shape))

train_df = main_encoded_df.loc[train_labels.index, :]
display('train data shape: {0}'.format(train_df.shape))
```

```
'train labels shape: (3996,)'
```

```
'train data shape: (3996, 127)'
```

[8]:
```
# Encode target labels

enc_train_labels = train_labels.map({
    'bad': 0, 'good': 1
})
```

[9]:
```
# Stratified CV

skf = StratifiedKFold(
    n_splits=3,
    shuffle=True
)
```

```
for train_idx, val_idx in skf.split(train_df, enc_train_labels):
    X_tr, X_val = train_df.iloc[train_idx, :], train_df.iloc[val_idx, :]
    y_tr, y_val = enc_train_labels.iloc[train_idx], enc_train_labels.
 ↪iloc[val_idx]

    lr_model = LogisticRegression(
        C=0.21544346900318845,
        solver='lbfgs',
        max_iter=100000,
        n_jobs=-1
    )
    lr_model.fit(X_tr, y_tr)
    lr_val_pred = lr_model.predict(X_val)
    print(confusion_matrix(y_val, lr_val_pred))

    draw_learning_curve( lr_model, X_tr, y_tr )


# For skf, the best log reg params are
# {'C': 0.21544346900318845, 'solver': 'lbfgs'}
```

[10]:
```
# Stratified Shuffle CV

sss = StratifiedShuffleSplit(
    n_splits=3,
    test_size=0.4,
)

for train_idx, val_idx in sss.split(train_df, enc_train_labels):
    X_tr, X_val = train_df.iloc[train_idx, :], train_df.iloc[val_idx, :]
    y_tr, y_val = enc_train_labels.iloc[train_idx], enc_train_labels.
 ↪iloc[val_idx]

    # Not enough variance to explain the data
    model_logreg = LogisticRegression(
        C=0.046415888336127774,
        solver='lbfgs',
        max_iter=100000,
        n_jobs=-1
    )
    model_logreg.fit( X_tr, y_tr )
    logreg_val_pred = model_logreg.predict( X_val )
    print(confusion_matrix(y_val, logreg_val_pred))

    draw_learning_curve( lr_model, X_tr, y_tr )
```

```
# For sss, the best log reg params are
# {'C': 0.046415888336127774, 'solver': 'lbfgs'}
```

[11]:
```
# Adding more features and shuffling data didn't help high bias problem
```

[68]:
```
# Try resampling the data (x2 with replacement)

# train_df['df'] = enc_train_labels

# resampled_train = resample(
#     train_df,
#     replace=True,
#     n_samples=6000
# )

# display('classes before:', train_df['df'].value_counts())
# display('classes after:', resampled_train['df'].value_counts())
# display('unique indices before:', np.unique(train_df.index).shape)
# display('unique indices after:', np.unique(resampled_train.index).shape)

# # Prepare for CV
# resampled_train_labels = resampled_train['df']
# resampled_train = resampled_train.drop( ['df'], axis=1 )

# Resampling whole dataset doesn't work - same really low recall
```

```
'classes before:'
```

```
1    3006
0     990
Name: df, dtype: int64
```

```
'classes after:'
```

```
1    4552
0    1448
Name: df, dtype: int64
```

```
'unique indices before:'
```

```
(3996,)
```

```
'unique indices after:'
```

```
(3091,)
```

[180]:
```
# Try oversampling, way 2: imblearn.over_sampling.SMOTE, knn-based algorithm

train_target = train_df['df']
train_features = train_df.drop(['df'], axis=1)

# Divide into train/valid sets before oversampling
VALID_SIZE = 0.4
X_tr, X_val, y_tr, y_val = train_test_split(
    train_features, train_target,
    test_size=VALID_SIZE
)

# Resample using SMOTE
sm = SMOTE(ratio=0.65)
X_tr_res, y_tr_res = sm.fit_sample( X_tr, y_tr )
display( X_tr_res.shape, y_tr_res.shape, pd.Series(y_tr_res).value_counts() )

# Create a model
lr_model = LogisticRegression(
    solver='lbfgs',
    C=0.004,
    n_jobs=-1
)
lr_model.fit( X_tr_res, y_tr_res )

# View the metrics
display('Validation results')
val_y_pred = lr_model.predict(X_val)
print( 'accuracy', lr_model.score(X_val, y_val) )
print( 'recall', recall_score(y_val, val_y_pred ) )
print( 'precision', precision_score(y_val, val_y_pred) )
print( '\nconfusion matrix:\n', confusion_matrix(y_val, val_y_pred) )
print( classification_report(y_val, val_y_pred) )
```

```
(2970, 127)
```

```
(2970,)
```

```
1    1800
0    1170
```

```
dtype: int64


'Validation results'


accuracy 0.7198248905565978
recall 0.9038142620232172
precision 0.7665260196905767

confusion matrix:
 [[  61  332]
 [ 116 1090]]
              precision    recall  f1-score   support

           0       0.34      0.16      0.21       393
           1       0.77      0.90      0.83      1206

    accuracy                           0.72      1599
   macro avg       0.56      0.53      0.52      1599
weighted avg       0.66      0.72      0.68      1599
```

[ ]:
```
# We want to specifically classify bad loans - they are more important.
# Therefore, we should watch for recall metric - number of correctly predicted␣
↪positives / total number of positives.
```