Sameeullah_File1_HW7

December 15, 2021

[ ]:
```
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats

pd.pandas.set_option('display.max_columns', None)
%matplotlib inline
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
#test dataset

df_test=pd.read_csv('/content/drive/MyDrive/test.csv')
test = df_test.copy()
```

[ ]:
```
df=pd.read_csv('/content/drive/MyDrive/train.csv')

data=df.copy()
```

[ ]:
```
categorical = [var for var in data.columns if data[var].dtypes=='O']
print(categorical)
print(len(categorical))
```

```
[]
0
```

[ ]:
```
numerical = [var for var in data.columns if data[var].dtypes!='O']
print(numerical)
print(len(numerical))
```

```
['Id', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22',
```

```
'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'V29', 'Target']
31
```

[ ]:
```
discrete = [var for var in numerical if len(data[var].unique()) < 20]
print(discrete)
print(f'There are {len(discrete)} discrete variables')
```

```
['Target']
There are 1 discrete variables
```

[ ]:
```
missing_data_columns= list(data.columns[data.isnull().mean()>0.0])
print(missing_data_columns)
len(missing_data_columns)
```

```
['V1', 'V20']
```

[ ]:
```
2
```

[ ]:
```
#divide dataset into train, test

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.drop(['Target','Id'],␣
 ↪axis=1),
                                                    data['Target'],
                                                    test_size=0.2,
                                                    random_state=0)

X_train.shape, X_test.shape
X_train.head()
y_train.head()
```

[ ]:
```
12739    0
3665     0
24445    0
19162    0
17492    0
Name: Target, dtype: int64
```

[ ]:
```
df_test=df_test.drop(['Id'], axis=1)
df_test.shape
```

[ ]:
```
(24846, 29)
```

[ ]:
```
!pip install feature-engine

from sklearn.preprocessing import RobustScaler
from feature_engine.imputation import MeanMedianImputer
from imblearn.pipeline import Pipeline
from feature_engine.transformation import YeoJohnsonTransformer
```

```
Collecting feature-engine
  Downloading feature_engine-1.1.2-py2.py3-none-any.whl (180 kB)
     || 180 kB 8.7 MB/s
Requirement already satisfied: scikit-learn>=0.22.2 in
/usr/local/lib/python3.7/dist-packages (from feature-engine) (1.0.1)
Collecting statsmodels>=0.11.1
  Downloading statsmodels-0.13.1-cp37-cp37m-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.8 MB)
     || 9.8 MB 45.0 MB/s
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.7
/dist-packages (from feature-engine) (1.19.5)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-
packages (from feature-engine) (1.4.1)
Requirement already satisfied: pandas>=1.0.3 in /usr/local/lib/python3.7/dist-
packages (from feature-engine) (1.1.5)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.0.3->feature-engine)
(2.8.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=1.0.3->feature-engine) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.7.3->pandas>=1.0.3->feature-engine) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7
/dist-packages (from scikit-learn>=0.22.2->feature-engine) (3.0.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.22.2->feature-engine) (1.1.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-
packages (from statsmodels>=0.11.1->feature-engine) (0.5.2)
Installing collected packages: statsmodels, feature-engine
  Attempting uninstall: statsmodels
    Found existing installation: statsmodels 0.10.2
    Uninstalling statsmodels-0.10.2:
      Successfully uninstalled statsmodels-0.10.2
Successfully installed feature-engine-1.1.2 statsmodels-0.13.1
```

[ ]:
```
data_pre_process = Pipeline([

    # missing data imputation
   ('mean_median_imputation', MeanMedianImputer(imputation_method='median',
                                    variables=missing_data_columns)),

    # Transforming Numerical Variables
   ('yjt', YeoJohnsonTransformer()),

    # feature Scaling
     ('scaler', RobustScaler())
```

```
])

data_pre_process.fit(X_train,y_train)
```

[ ]:
```
Pipeline(steps=[('mean_median_imputation',
                 MeanMedianImputer(variables=['V1', 'V20'])),
                ('yjt', YeoJohnsonTransformer()), ('scaler', RobustScaler())])
```

[ ]:
```
X_train=pd.DataFrame(data_pre_process.transform(X_train),columns=X_train.
 ↪columns)
X_test=pd.DataFrame(data_pre_process.transform(X_test),columns=X_test.columns)
```

[ ]:
```
df_test=pd.DataFrame(data_pre_process.transform(df_test),columns=df_test.
 ↪columns)
```

[ ]:
```
#Making f2 scorer for Grid search CV
from sklearn.metrics import fbeta_score, make_scorer
ftwo_scorer = make_scorer(fbeta_score, beta=2)
ftwo_scorer
```

[ ]:
```
make_scorer(fbeta_score, beta=2)
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from  sklearn.ensemble import ExtraTreesClassifier
from  sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.metrics import f1_score
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import fbeta_score
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import SVMSMOTE
from imblearn.over_sampling import ADASYN

import pickle
```

[ ]:
```
#Tried all possibilites for k neighbours like 10,50,100 to avoid overfitting␣
 ↪but no luck
```

```
pipe_xgb_svmsmote = Pipeline([('svmsmote', SVMSMOTE()), ('model',␣
 ↪XGBClassifier(random_state=42,early_stopping_rounds=2,tree_method = 'hist',
                                                                        ␣
 ↪objective= 'binary:logistic'))])
param_grid = {
    # try different feature engineering parameters
    'svmsmote__k_neighbors': [2],
    'model__n_estimators': [60],
    'model__max_depth': [5],
    'model__subsample': [0.9]
}

#apply grid search
grid_svmsmote_xgb= GridSearchCV(pipe_xgb_svmsmote, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring=ftwo_scorer)
grid_svmsmote_xgb.fit(X_train, y_train)
grid_svmsmote_xgb.best_estimator_
```

[ ]:
```
Pipeline(steps=[('svmsmote', SVMSMOTE(k_neighbors=2)),
                ('model',
                 XGBClassifier(early_stopping_rounds=2, max_depth=5,
                               n_estimators=60, random_state=42, subsample=0.9,
                               tree_method='hist'))])
```

[ ]:
```
logreg = LogisticRegression()
param_grid = {

    'class_weight': [{0:1,1:10}],
    'C': [0.01],
    'penalty': ['l2']
}
#apply grid search
grid_logreg= GridSearchCV(logreg, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring=ftwo_scorer)
grid_logreg.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [0.01], 'class_weight': [{0: 1, 1: 10}],
                         'penalty': ['l2']},
             scoring=make_scorer(fbeta_score, beta=2))
```

[ ]:
```
# GridSearch with oversampling
pipe_knn_adasyn = Pipeline([('adasyn', SVMSMOTE()), ('model',␣
 ↪KNeighborsClassifier())])
param_grid = {
    # try different feature engineering parameters
    'adasyn__k_neighbors': [10],
    #'model__p': [3,4,5,6],
```

```
    'model__n_neighbors' : [10],
    #'model__weights': ['uniform','distance']
}

#apply grid search
grid_adasyn_knn= GridSearchCV(pipe_knn_adasyn, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring=ftwo_scorer)
grid_adasyn_knn.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-
packages/joblib/externals/loky/process_executor.py:705: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

[ ]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('adasyn', SVMSMOTE()),
                                       ('model', KNeighborsClassifier())]),
             n_jobs=-1,
             param_grid={'adasyn__k_neighbors': [10],
                         'model__n_neighbors': [10]},
             scoring=make_scorer(fbeta_score, beta=2))
```

[ ]:
```
#class weight balanced, balanced_subsample
etc2 = ExtraTreesClassifier(random_state=42)
param_grid = {

    'class_weight': ['balanced'],
    'n_estimators': [500],
    'max_features': ['auto'],
    'max_depth' : [10],
    'criterion' :['gini']
}

#apply grid search
grid_etc2= GridSearchCV(etc2, param_grid, cv=5, n_jobs=-1, scoring=ftwo_scorer)
grid_etc2.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=5, estimator=ExtraTreesClassifier(random_state=42), n_jobs=-1,
             param_grid={'class_weight': ['balanced'], 'criterion': ['gini'],
                         'max_depth': [10], 'max_features': ['auto'],
                         'n_estimators': [500]},
             scoring=make_scorer(fbeta_score, beta=2))
```

[ ]:
```
from sklearn.neural_network import MLPClassifier
pipe_mlp_svmsmote = Pipeline([('svmsmote', SVMSMOTE()), ('model',␣
 ↪MLPClassifier())])
param_grid = {
    # try different feature engineering parameters
```

```
    'svmsmote__k_neighbors': [10],
    'model__alpha': [0.9],
    'model__solver': ['sgd']
}

#apply grid search
grid_svmsmote_mlp= GridSearchCV(pipe_mlp_svmsmote, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring=ftwo_scorer)
grid_svmsmote_mlp.fit(X_train, y_train)

print("Best parameters: {}".format(grid_svmsmote_mlp.best_params_))
print("Best Mean cross-validation score: {:.5f}".format(grid_svmsmote_mlp.
 ↪best_score_))
print(f'Train score is {grid_svmsmote_mlp.score(X_train,y_train)}')
print(f'Test score is {grid_svmsmote_mlp.score(X_test,y_test)}')
```

```
Best parameters: {'model__alpha': 0.9, 'model__solver': 'sgd',
'svmsmote__k_neighbors': 10}
Best Mean cross-validation score: 0.75028
Train score is 0.7792792792792793
Test score is 0.7508532423208191
```

[ ]:
```
stack1 =StackingClassifier(estimators =
                           [('xgb_over', grid_svmsmote_xgb.best_estimator_),
                           ('knn_over', grid_adasyn_knn.best_estimator_),
                           ('cost_trees', grid_etc2.best_estimator_)
                           ], final_estimator =␣
 ↪XGBClassifier(random_state=42,early_stopping_rounds=2,tree_method =␣
 ↪'hist',objective= 'binary:logistic'))

stack1_params = {
    'final_estimator__n_estimators':[50],
    'final_estimator__max_depth': [5],
    'final_estimator__subsample':[.9],
}

stack1_grid = GridSearchCV(stack1,stack1_params, cv = 5,␣
 ↪n_jobs=-1,return_train_score=True)
stack1_grid.fit(X_train,y_train)
```

[ ]:
```
GridSearchCV(cv=5,
             estimator=StackingClassifier(estimators=[('xgb_over',
Pipeline(steps=[('svmsmote',
SVMSMOTE(k_neighbors=2)),
                                                      ('model',
XGBClassifier(early_stopping_rounds=2,
      max_depth=5,
```

```
      n_estimators=60,
      random_state=42,
      subsample=0.9,
      tree_method='hist'))])),
                                                    ('knn_over',
Pipeline(steps=[('adasyn',
SVMSMOTE(k_neighbors=10)),
                                                                ('model',
KNeighborsClassifier(n_n...
                                                    ('cost_trees',
ExtraTreesClassifier(class_weight='balanced',
max_depth=10,
n_estimators=500,
random_state=42))],
final_estimator=XGBClassifier(early_stopping_rounds=2,
random_state=42,
tree_method='hist')),
             n_jobs=-1,
             param_grid={'final_estimator__max_depth': [5],
                         'final_estimator__n_estimators': [50],
                         'final_estimator__subsample': [0.9]},
             return_train_score=True)
```

[ ]:
```
print(f'Best Mean Cross Validation Score is {stack1_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {stack1_grid.best_params_}')
print(f'Train score is {stack1_grid.score(X_train,y_train)}')
print(f'Test score is {stack1_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9979372081545879
Best Mean Cross Validation Score is {'final_estimator__max_depth': 5,
'final_estimator__n_estimators': 50, 'final_estimator__subsample': 0.9}
Train score is 0.9988931374522036
Test score is 0.996579476861167
```

[ ]:
```
stack2 =StackingClassifier(estimators =
                           [('xgb_over', grid_svmsmote_xgb.best_estimator_),
                           ('logreg', grid_logreg.best_estimator_),
                           ('mlp', grid_svmsmote_mlp.best_estimator_)
                           ], final_estimator =␣
 ↪XGBClassifier(random_state=42,tree_method =␣
 ↪'hist',early_stopping_rounds=2,objective= 'binary:logistic'))

stack2_params = {
    'final_estimator__n_estimators':[60],
    'final_estimator__max_depth': [5],
    'final_estimator__subsample':[1],
}
```

```
stack2_grid = GridSearchCV(stack2,stack2_params, cv = 5,␣
 ↪n_jobs=-1,return_train_score=True)
stack2_grid.fit(X_train,y_train)
```

[ ]:
```
GridSearchCV(cv=5,
             estimator=StackingClassifier(estimators=[('xgb_over',
Pipeline(steps=[('svmsmote',
SVMSMOTE(k_neighbors=2)),
                                                      ('model',
XGBClassifier(early_stopping_rounds=2,
      max_depth=5,
      n_estimators=60,
      random_state=42,
      subsample=0.9,
      tree_method='hist'))])),
                                                     ('logreg',
LogisticRegression(C=0.01,
class_weight={0: 1,
        1: 10})),
                                                     ('mlp',
Pipeline(steps=[('svmsmote',
SVMSMOTE(k_neighbors=10)),
                                                      ('model',
MLPClassifier(alpha=0.9,
      solver='sgd'))]))],
final_estimator=XGBClassifier(early_stopping_rounds=2,
random_state=42,
tree_method='hist')),
             n_jobs=-1,
             param_grid={'final_estimator__max_depth': [5],
                         'final_estimator__n_estimators': [60],
                         'final_estimator__subsample': [1]},
             return_train_score=True)
```

[ ]:
```
print(f'Best Mean Cross Validation Score is {stack2_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {stack2_grid.best_params_}')
print(f'Train score is {stack2_grid.score(X_train,y_train)}')
print(f'Test score is {stack2_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9978868430710047
Best Mean Cross Validation Score is {'final_estimator__max_depth': 5,
'final_estimator__n_estimators': 60, 'final_estimator__subsample': 1}
Train score is 0.9986918897162407
Test score is 0.9963782696177063
```

The first stacking method has a slightly lower test score but it takes significantly less time to train, with a 13 minute deficit between the first and second stacking method. This is likely due to the application of a nueral network in the 2nd stacking model's base.

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Sameeullah_File1_HW7.ipynb')
```

```
File colab_pdf.py already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```