Real_Fake_Classification

August 4, 2020

1 Fake or real Job Classification

- Data Shape: rows: 17880, columns: 17
- Missing data Percentage: 27.84%
- Goal : Classify Real or Fake job applications
- Evaluation Metric: ROC - AUC

2 Results

Best Model: XGBoost Classfier

Best Mean Cross Validation Score is 0.9577 Best Mean Cross Validation Score is {'learning_rate': 0.5, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 100, 'subsample': 0.7} Train score is 0.9688 Test score is 0.9666

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
# from feature-engine
from feature_engine import missing_data_imputers as mdi
# for one hot encoding with feature-engine
from feature_engine.categorical_encoders import OneHotCategoricalEncoder
from feature_engine.categorical_encoders import RareLabelCategoricalEncoder
from sklearn.pipeline import Pipeline
```

[4]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from  sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from  sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import StackingClassifier
```

3 Data Loading and Exploration

[5]:
```
# load data

data=pd.read_csv('fake_job_postings.csv')
```

[6]:
```
data.head()
```

[6]:

	job_id	title	location	department	salary_range	company_profile	description	requirements
0	1	Marketing Intern	US, NY, New York	Marketing	NaN	We're Food52, and we've created a groundbreaki...	Food52, a fast-growing, James Beard Award-winn...	Experience with content management systems a m...
1	2	Customer Service - Cloud Video Production	NZ, , Auckland	Success	NaN	90 Seconds, the worlds Cloud Video Production ...	Organised - Focused - Vibrant - Awesome!Do you...	What we expect from you:Your key responsibilit...
2	3	Commissioning Machinery Assistant (CMA)	US, IA, Wever	NaN	NaN	Valor Services provides Workforce Solutions th...	Our client, located in Houston, is actively se...	Implement pre-commissioning and commissioning ...
3	4	Account Executive - Washington DC	US, DC, Washington	Sales	NaN	Our passion for improving quality of life thro...	THE COMPANY: ESRI Environmental Systems Rese...	EDUCATION:ăBachelors or Masters in GIS, busi...
4	5	Bill Review Manager	US, FL, Fort Worth	NaN	NaN	SpotSource Solutions LLC is a Global Human Cap...	JOB TITLE: Itemization Review ManagerLOCATION:...	QUALIFICATIONS:RN license in the State of Texa...

```
                                              benefits  telecommuting  \
0                                                  NaN              0
1  What you will get from usThrough being part of...               0
2                                                  NaN              0
3  Our culture is anything but corporatewe have ...               0
4                            Full Benefits Offered                  0

   has_company_logo  has_questions employment_type required_experience  \
0                 1              0           Other          Internship
1                 1              0       Full-time      Not Applicable
2                 1              0             NaN                 NaN
3                 1              0       Full-time    Mid-Senior level
4                 1              1       Full-time    Mid-Senior level

  required_education                   industry              function  \
0                NaN                        NaN             Marketing
1                NaN  Marketing and Advertising      Customer Service
2                NaN                        NaN                 NaN
3  Bachelor's Degree          Computer Software                Sales
4  Bachelor's Degree     Hospital & Health Care  Health Care Provider

   fraudulent
0           0
1           0
2           0
3           0
4           0
```

[7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 18 columns):
job_id                 17880 non-null int64
title                  17880 non-null object
location               17534 non-null object
department             6333 non-null object
salary_range           2868 non-null object
company_profile        14572 non-null object
description            17879 non-null object
requirements           15185 non-null object
benefits               10670 non-null object
telecommuting          17880 non-null int64
has_company_logo       17880 non-null int64
has_questions          17880 non-null int64
employment_type        14409 non-null object
required_experience    10830 non-null object
```

```
required_education     9775 non-null object
industry               12977 non-null object
function               11425 non-null object
fraudulent             17880 non-null int64
dtypes: int64(5), object(13)
memory usage: 2.5+ MB
```

[7]:
```
data.shape
```

[7]:
```
(17880, 18)
```

3.1 Total Percentage Missing values in Data set

[8]:
```
print('Total missing values in the dataset are',100* data.isnull().sum().sum()/
 ↪(data.count().sum()),'%')
```

```
Total missing values in the dataset are 27.84771408255441 %
```

[12]:
```
# calculate null values

data.isnull().sum()/len(data)
```

[12]:
```
job_id                 0.000000
title                  0.000000
location               0.019351
department             0.645805
salary_range           0.839597
company_profile        0.185011
description            0.000056
requirements           0.150727
benefits               0.403244
telecommuting          0.000000
has_company_logo       0.000000
has_questions          0.000000
employment_type        0.194128
required_experience    0.394295
required_education     0.453300
industry               0.274217
function               0.361018
fraudulent             0.000000
dtype: float64
```

[13]:
```
# drop not useful columns

data=data.
 ↪drop(['job_id','title','location','company_profile','description','requirements','benefits'
                'salary_range'],axis=1)
```

[14]:
```
data.columns
```

[14]:
```
Index(['department', 'telecommuting', 'has_company_logo', 'has_questions',
       'employment_type', 'required_experience', 'required_education',
       'industry', 'function', 'fraudulent'],
      dtype='object')
```

[11]:
```
data.shape
```

[11]:
```
(17880, 10)
```

3.2 Output variable imbalance rate

[15]:
```
data['fraudulent'].value_counts()/len(data['fraudulent'])
```

[15]:
```
0    0.951566
1    0.048434
Name: fraudulent, dtype: float64
```

4 Extracting first part of the salary range

[ ]:
```
data['sal']=data['salary_range'].str.split('-')
```

[ ]:
```
data['sal'].head(10)
```

[ ]:
```
data['sal'].dtype
```

[ ]:
```
data['sal'][6][1]
```

[ ]:
```
# 0 is start point and 1 is end but in python indexing doesnot go to last hence␣
 ↪0
data['sal1']=data['sal'].str[0]
data['sal1'].head(7)
```

[ ]:
```
data['sal2']=data['sal'].str[1]
data['sal2'].head(7)
```

[ ]:
```
# for loop for that

for i in range(len(data['sal'])):
    data['sal1'][i]=data['sal'][i].str[0]
    data['sal2'][i]=data['sal'][i].str[1]
```

[ ]:
```
#data['sal1']=data['sal1'].astype(int) this is giving errors
data['sal1'] = pd.to_numeric(data['sal1'], errors='coerce')
#data = data.dropna(subset=['sal1'])
data['sal1']=data['sal1'].astype(float)
#link: https://stackoverflow.com/questions/47333227/
 ↪pandas-valueerror-cannot-convert-float-nan-to-integer
# putting int was giving an error
```

[ ]:
```
data['sal1'].dtype
```

[ ]:
```
data['sal1']
```

[ ]:
```
#data['sal2']=data['sal2'].astype('Int32/64')cnot working lets try other way
#https://stackoverflow.com/questions/47333227/
 ↪pandas-valueerror-cannot-convert-float-nan-to-integer
data['sal2'] = pd.to_numeric(data['sal2'], errors='coerce')
#data = data.dropna(subset=['sal2'])
data['sal2']=data['sal2'].astype(float)
# putting int was giving an error
```

[ ]:
```
data['sal2'].dtype
```

[ ]:
```
data['sal2']
```

[ ]:
```
# lets take avrage of the sal1 and sal2

data['Avg_sal']=data[['sal1','sal2']].mean(axis=1)
```

[ ]:
```
data['Avg_sal'].head(7)
```

[ ]:
```
data.columns
```

lets frop all the salaries as 83% of data is missing and imputing will add bias to the dataset drop sal 1 and sal 2 and salary_range

[ ]:
```
data=data.drop(['sal1','sal2','salary_range','sal'],axis=1)
```

[ ]:
```
data.columns
```

[ ]:
```
100*data['dept'].value_counts()
```

4.1 Data Exploration

[133]:
```
categorical=['department','employment_type','industry','function','required_experience',
 ↪'required_education','has_company_logo','has_questions','telecommuting']

for var in categorical:
    data[var]=data[var].astype('object')
```

[ ]:
```
100*data['department'].value_counts()
```

[14]:
```
data['function'].value_counts()/len(data['function'])
```

[14]:
```
Information Technology    0.097819
Sales                     0.082103
Engineering               0.075391
Customer Service          0.068736
Marketing                 0.046421
Administrative            0.035235
Design                    0.019016
Health Care Provider      0.018904
Education                 0.018177
```

```
Other                     0.018177
Management                0.017729
Business Development      0.012752
Accounting/Auditing       0.011857
Human Resources           0.011465
Project Management        0.010235
Finance                   0.009620
Consulting                0.008054
Art/Creative              0.007383
Writing/Editing           0.007383
Production                0.006488
Product Management        0.006376
Quality Assurance         0.006208
Advertising               0.005034
Business Analyst          0.004698
Data Analyst              0.004586
Public Relations          0.004251
Manufacturing             0.004139
General Business          0.003803
Research                  0.002796
Legal                     0.002629
Strategy/Planning         0.002573
Training                  0.002125
Supply Chain              0.002013
Financial Analyst         0.001846
Distribution              0.001342
Purchasing                0.000839
Science                   0.000783
Name: function, dtype: float64
```

[9]:
```
data['industry'].value_counts()/len(data['industry'])
```

[9]:
```
Information Technology and Services    0.096980
Computer Software                      0.076957
Internet                               0.059396
Marketing and Advertising              0.046309
Education Management                   0.045973
Financial Services                     0.043568
Hospital & Health Care                 0.027796
Consumer Services                      0.020022
Telecommunications                     0.019128
Oil & Energy                           0.016051
Retail                                 0.012472
Real Estate                            0.009787
Accounting                             0.008893
Construction                           0.008837
E-Learning                             0.007774
Management Consulting                  0.007271
```

```
Design                                0.007215
Staffing and Recruiting               0.007103
Health, Wellness and Fitness          0.007103
Insurance                             0.006879
Automotive                            0.006711
Logistics and Supply Chain            0.006264
Human Resources                       0.006040
Online Media                          0.005649
Apparel & Fashion                     0.005425
Legal Services                        0.005425
Facilities Services                   0.005257
Hospitality                           0.004922
Computer Games                        0.004810
Banking                               0.004698
                                        ...
Religious Institutions                0.000336
Motion Pictures and Film              0.000336
Investment Management                 0.000336
Animation                             0.000280
Capital Markets                       0.000280
Packaging and Containers              0.000280
Package/Freight Delivery              0.000280
Import and Export                     0.000280
Fishery                               0.000224
Wireless                              0.000224
Commercial Real Estate                0.000224
Investment Banking                    0.000224
Luxury Goods & Jewelry                0.000224
Philanthropy                          0.000224
Furniture                             0.000168
Public Policy                         0.000168
Plastics                              0.000168
Performing Arts                       0.000168
Mining & Metals                       0.000168
Maritime                              0.000168
Libraries                             0.000112
Nanotechnology                        0.000112
Military                              0.000112
Textiles                              0.000112
Shipbuilding                          0.000056
Sporting Goods                        0.000056
Ranching                              0.000056
Museums and Institutions              0.000056
Alternative Dispute Resolution        0.000056
Wine and Spirits                      0.000056
Name: industry, Length: 131, dtype: float64
```

[10]:
```
data['employment_type'].value_counts()/len(data['employment_type'])
```

[10]:
```
Full-time     0.649888
Contract      0.085235
Part-time     0.044575
Temporary     0.013479
Other         0.012696
Name: employment_type, dtype: float64
```

5 Train Test Split

[118]:
```
x_train, x_test, y_train,y_test= train_test_split(data.
 ↪drop('fraudulent',axis=1),data['fraudulent'],test_size=0.30
                                                    ,random_state=0)

x_train.shape,x_test.shape, y_train.shape
```

[118]:
```
((12516, 9), (5364, 9), (12516,))
```

[119]:
```
x_train.dtypes
```

[119]:
```
department              object
telecommuting           object
has_company_logo        object
has_questions           object
employment_type         object
required_experience     object
required_education      object
industry                object
function                object
dtype: object
```

5.1 Pipeline

[120]:
```
# pipeline for basic imputation
# 0.017

fakejob_pipe = Pipeline([


   ('imputer_cat_freq',mdi.
 ↪FrequentCategoryImputer(variables=['department','employment_type','industry','function',
                                                                 ␣
 ↪'required_experience','required_education'])),

       # categorical encoding
    ('encoder_rare_label_dept',
```

```
    RareLabelCategoricalEncoder(tol=0.004235,
                                n_categories=10,
                                variables=['department'])),


    ('encoder_rare_label_industry',RareLabelCategoricalEncoder(tol=0.02,
                                n_categories=10,
                                variables=['industry','function'])),

   ('categorical_encoder',
    OneHotCategoricalEncoder( top_categories=None,
                             variables=categorical, # we can select which␣
 ↪variables to encode
                             drop_last=True)),

])
```

[121]:
```
fakejob_pipe.fit(x_train,y_train)
```

[121]:
```
Pipeline(memory=None,
         steps=[('imputer_cat_freq',
                 FrequentCategoryImputer(variables=['department',
                                                    'employment_type',
                                                    'industry', 'function',
                                                    'required_experience',
                                                    'required_education'])),
                ('encoder_rare_label_dept',
                 RareLabelCategoricalEncoder(n_categories=10, tol=0.004235,
                                             variables=['department'])),
                ('encoder_rare_label_industry',
                 RareLabelCategoricalEncoder(n_categories=10, tol=0.02,
                                             variables=['industry',
                                                        'function'])),
                ('categorical_encoder',
                 OneHotCategoricalEncoder(drop_last=True, top_categories=None,
                                          variables=['department',
                                                     'employment_type',
                                                     'industry', 'function',
                                                     'required_experience',
                                                     'required_education',
                                                     'has_company_logo',
                                                     'has_questions',
                                                     'telecommuting']))],
         verbose=False)
```

[122]:
```
X_test=fakejob_pipe.transform (x_test)
X_train=fakejob_pipe.transform(x_train)
```

[49]:
```
pip install imblearn
```

```
Collecting imblearn
  Downloading https://files.pythonhosted.org/packages/81/a7/4179e6ebfd654bd0eac0
b9c06125b8b4c96a9d0a8ff9e9507eb2a26d2d7e/imblearn-0.0-py2.py3-none-any.whl
Collecting imbalanced-learn (from imblearn)
  Downloading https://files.pythonhosted.org/packages/c8/73/36a13185c2acff44d601
dc6107b5347e075561a49e15ddd4e69988414c3e/imbalanced_learn-0.6.2-py3-none-any.whl
(163kB)
Requirement already satisfied: scikit-learn>=0.22 in
c:\users\ajink\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(0.22.2.post1)
Requirement already satisfied: numpy>=1.11 in c:\users\ajink\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.16.4)
Requirement already satisfied: joblib>=0.11 in
c:\users\ajink\anaconda3\lib\site-packages (from imbalanced-learn->imblearn)
(0.13.2)
Requirement already satisfied: scipy>=0.17 in c:\users\ajink\anaconda3\lib\site-
packages (from imbalanced-learn->imblearn) (1.2.1)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.6.2 imblearn-0.0
Note: you may need to restart the kernel to use updated packages.
```

6 Simple Data Models

6.1 Naive Alogorithm

[48]:
```
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score
model_dummy = DummyClassifier(strategy='stratified',random_state=123)
model_dummy.fit(X_train,y_train)
cv_scores = cross_val_score(model_dummy, X_train, y_train,scoring='roc_auc')
# Mean Cross validation Score
print("Mean Cross-validation scores: {}".format(cv_scores.mean()))
print()
# Check test data set performance
print("Naive Algorithm Test Performance: ", model_dummy.score(X_test,y_test))
```

```
Mean Cross-validation scores: 0.4947302186371939

Naive Algorithm Test Performance:  0.9095824011931395
```

6.2 Logistic Regression

[49]:
```
clf = LogisticRegression(max_iter=1000).fit(X_train, y_train)

cv_scores = cross_val_score(clf, X_train, y_train, scoring='roc_auc')

# Mean Cross validation Score
```

```
print("Mean Cross-validation scores: {}".format(cv_scores.mean()))
print()

# Print Co-efficients
print("Logistic.coef_:", clf.coef_)
print("Logistic.intercept_:", clf.intercept_)

# Check test data set performance
print("Logistic Train Performance: ", clf.score(X_train,y_train))
print("Logistic Test Performance: ", clf.score(X_test,y_test))
```

```
Mean Cross-validation scores: 0.865556789672872

Logistic.coef_: [[ 1.06372644e-01  8.97898238e-01 -9.87259673e-01
2.10454026e+00
  -5.48775776e-01  1.59830457e+00 -1.02634344e+00 -4.89684537e-01
   2.91518077e-01  3.50222630e-01  9.03508248e-01  9.08113216e-01
  -1.91966004e-01 -1.93639794e+00  2.06865393e-03 -2.32582288e+00
  -5.57011708e-01 -2.69358953e+00  2.49541600e-01 -3.27276261e-01
   1.99419024e-01  5.79575534e-01  1.58593282e+00 -8.28496456e-01
   7.69073973e-02  1.17943945e+00  1.97780581e-01 -4.30288541e-01
  -1.07921870e-01 -8.29777738e-01  3.17469113e-01 -4.81185310e-01
  -2.29782131e-01  7.39545498e-01 -2.67032502e-01 -1.52618707e-01
  -1.54789875e-01  1.86291591e-01  9.19005718e-02 -2.91115967e-01
  -8.49456942e-01 -3.92312491e-01  1.66981808e+00 -2.93800519e-01
  -2.44373532e+00 -3.33130448e-01 -3.30769126e-01]]
Logistic.intercept_: [-1.72218706]
Logistic Train Performance:  0.9517417705337169
Logistic Test Performance:  0.9599179716629381
```

6.3 Ridge Regression

[50]:
```
ridge = Ridge()

#define a list of parameters
param_ridge = {'alpha':[0.001, 0.01,0.20] }

grid_ridge = GridSearchCV(ridge, param_ridge, cv=10,scoring='roc_auc',␣
 ↪return_train_score = True)
grid_ridge.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.4f}".format(grid_ridge.best_score_))

#find best parameters
print('Ridge parameters: ', grid_ridge.best_params_)
```

```
# Check test data set performance
print("Ridge Train Performance: ", grid_ridge.score(X_train,y_train))
print("Ridge Test Performance: ", grid_ridge.score(X_test,y_test))
```

```
Best Mean Cross-validation score: 0.8582
Ridge parameters:  {'alpha': 0.01}
Ridge Train Performance:  0.8627751089249048
Ridge Test Performance:  0.8421587703959955
```

6.4 Lasso Regression

[51]:
```
lasso = Lasso()

#define a list of parameters
param_lasso = {'alpha':[0.001, 0.01,0.20] }

grid_lasso = GridSearchCV(lasso, param_ridge, cv=10,scoring='roc_auc',␣
 ↪return_train_score = True)
grid_lasso.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.4f}".format(grid_lasso.best_score_))

#find best parameters
print('Ridge parameters: ', grid_lasso.best_params_)


# Check test data set performance
print("Ridge Train Performance: ", grid_lasso.score(X_train,y_train))
print("Ridge Test Performance: ", grid_lasso.score(X_test,y_test))
```

```
Best Mean Cross-validation score: 0.8443
Ridge parameters:  {'alpha': 0.001}
Ridge Train Performance:  0.846635170118124
Ridge Test Performance:  0.8339017527671322
```

6.5 Elastic Net

[107]:
```
from sklearn.linear_model import ElasticNet
elasticnet = ElasticNet()

#define a list of parameters
param_elasticnet = {'alpha':[0.001, 0.01,0.20], 'l1_ratio' :[0.001, 0.01,0.20]}

grid_elasticnet = GridSearchCV(elasticnet , param_elasticnet, cv=10,␣
 ↪return_train_score = True)
```

```
grid_elasticnet.fit(X_train, y_train)

grid_elasticnet_train_score = grid_elasticnet.score(X_train, y_train)
grid_elasticnet_test_score = grid_elasticnet.score(X_test, y_test)

# Find Best parameters
print('Best parameters: ', grid_elasticnet.best_params_)
print('Best cross-validation score:', grid_elasticnet.best_score_)
print()
# Performance
print('Training set score: ', grid_elasticnet_train_score)
print('Test score: ', grid_elasticnet_test_score)
```

```
Best parameters:  {'alpha': 0.001, 'l1_ratio': 0.001}
Best cross-validation score: 0.11646492980181855

Training set score:  0.12489338567852638
Test score:  0.09755978134690091
```

6.6 KNN

[56]:
```
knn = KNeighborsClassifier()

# define a list of parameters

param_knn = {'n_neighbors': range(1,5)}

#apply grid search
grid_knn = GridSearchCV(knn, param_knn, cv=10, n_jobs=2, scoring='roc_auc')
grid_knn.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.4f}".format(grid_knn.best_score_))
print()

#find best parameters
print('KNN parameters: ', grid_knn.best_params_)

# Check train data set performance
print("KNN Train Performance: ", grid_knn.score(X_train,y_train))

# Check test data set performance
print("KNN Test Performance: ", grid_knn.score(X_test,y_test))
```

```
Best Mean Cross-validation score: 0.7966

KNN parameters:  {'n_neighbors': 4}
```

```
KNN Train Performance:  0.8544179957237744
KNN Test Performance:  0.8043845333886747
```

6.7 Decision Tree

[57]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0)

#define a list of parameters
param_dtree = {'max_depth': range(1,20)}

#apply grid search
grid_dtree = GridSearchCV(dtree, param_dtree, cv=10, return_train_score = True,␣
 ↪scoring = 'roc_auc')
grid_dtree.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.4f}".format(grid_dtree.best_score_))
print()

#find best parameters
print('Decision Tree parameters: ', grid_dtree.best_params_)

# Check test data set performance
print("Decision Tree Performance: ", grid_dtree.score(X_train,y_train))

# Check test data set performance
print("Decision Tree Performance: ", grid_dtree.score(X_test,y_test))
```

```
Best Mean Cross-validation score: 0.8698

Decision Tree parameters:  {'max_depth': 10}
Decision Tree Performance:  0.9404839880027454
Decision Tree Performance:  0.8677067727525708
```

6.8 SVM

[ ]:
```
from sklearn import svm
supvm = svm.SVC(random_state=0)
C = [0.1, 1, 10]
param_svm = [{'kernel': ['rbf'],
              'C': C,
              'gamma': [0.01, 0.1, 1]},
             {'kernel': ['linear'],
              'C': C}]
```

```
grid_svm = GridSearchCV(supvm, param_svm, cv=5,
return_train_score=True,scoring = 'roc_auc')
grid_svm.fit(X_train, y_train)
```

[ ]:
```
print('train score: ', grid_svm.score(X_train, y_train))
print('test score: ', grid_svm.score(X_train, y_train))
print("Best parameters: {}".format(grid_svm.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_svm.best_score_))
```

7 Ensemble Models

7.1 Bagging

[27]:
```
bag_dtree = BaggingClassifier(DecisionTreeClassifier(max_depth= 8,␣
 ↪max_leaf_nodes=5, min_samples_split= 3,
                                                      splitter= 'random'),␣
 ↪bootstrap=True, random_state=0,
                               oob_score=True)

bag_dtree_param = {
                 'max_samples':  [0.8,1],
                 'n_estimators': [10,25,100]}
bag_dtree_grid = GridSearchCV(bag_dtree, bag_dtree_param,cv=5,␣
 ↪return_train_score=True, )
bag_dtree_grid.fit(X_train,y_train)
```

[27]:
```
GridSearchCV(cv=5, error_score=nan,
estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=8,
max_features=None,
max_leaf_nodes=5,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=3,
min_weight_fraction_leaf=0.0,
presort='deprecated',
random_state=None,
splitter='random'),
                                        bootstrap=True,
                                        bootstrap_features=False,
                                        max_features=1.0, max_samples=1.0,
                                        n_estimators=10, n_jobs=None,
                                        oob_score=True, random_state=0,
                                        verbose=0, warm_start=False),
```

```
                iid='deprecated', n_jobs=None,
                param_grid={'max_samples': [0.8, 1],
                            'n_estimators': [10, 25, 100]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring=None, verbose=0)
```

[33]:
```
print(f'Best Mean Cross Validation Score is {bag_dtree_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {bag_dtree_grid.best_params_}')
print(f'Train score is {bag_dtree_grid.score(X_train,y_train)}')
print(f'Test score is {bag_dtree_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9519813603867053
Best Mean Cross Validation Score is {'max_samples': 0.8, 'n_estimators': 100}
Train score is 0.953579418344519
Test score is 0.9634601043997018
```

7.2 Pasting

[28]:
```
paste_dtree = BaggingClassifier(DecisionTreeClassifier(max_depth= 8,␣
 ↪max_leaf_nodes=5, min_samples_split= 3,
                                                        splitter= 'random'),␣
 ↪bootstrap=False, random_state=0,
                                oob_score=True)

bag_dtree_param = {
                 'max_samples':  [0.8,1],
                 'n_estimators': [10,25,100]}
paste_dtree_grid = GridSearchCV(bag_dtree, bag_dtree_param,cv=5,␣
 ↪return_train_score=True, )
paste_dtree_grid.fit(X_train,y_train)
```

[28]:
```
GridSearchCV(cv=5, error_score=nan,
estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=8,
max_features=None,
max_leaf_nodes=5,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=3,
min_weight_fraction_leaf=0.0,
presort='deprecated',
random_state=None,
splitter='random'),
                                          bootstrap=True,
                                          bootstrap_features=False,
```

```
                                       max_features=1.0, max_samples=1.0,
                                       n_estimators=10, n_jobs=None,
                                       oob_score=True, random_state=0,
                                       verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_samples': [0.8, 1],
                         'n_estimators': [10, 25, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

[29]:
```
print(f'Best Mean Cross Validation Score is {paste_dtree_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {paste_dtree_grid.best_params_}')
print(f'Train score is {paste_dtree_grid.score(X_train,y_train)}')
print(f'Test score is {paste_dtree_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9519813603867053
Best Mean Cross Validation Score is {'max_samples': 0.8, 'n_estimators': 100}
Train score is 0.953579418344519
Test score is 0.9634601043997018
```

7.3 Random Forrest

[179]:
```
from sklearn.ensemble import RandomForestClassifier
rfc =RandomForestClassifier(random_state=42,oob_score=True)
rfc_param = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [2,4,5,6,7,8],

    'criterion' :['gini', 'entropy']
}

rfc_grid = GridSearchCV(rfc, rfc_param,cv=5, return_train_score=True, )
rfc_grid.fit(X_train,y_train)
```

[179]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
```

```
                                             oob_score=True, random_state=42,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [200, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

[36]:
```
rfc_grid.get_params().keys()
```

[36]:
```
dict_keys(['cv', 'error_score', 'estimator__bootstrap', 'estimator__ccp_alpha',
'estimator__class_weight', 'estimator__criterion', 'estimator__max_depth',
'estimator__max_features', 'estimator__max_leaf_nodes',
'estimator__max_samples', 'estimator__min_impurity_decrease',
'estimator__min_impurity_split', 'estimator__min_samples_leaf',
'estimator__min_samples_split', 'estimator__min_weight_fraction_leaf',
'estimator__n_estimators', 'estimator__n_jobs', 'estimator__oob_score',
'estimator__random_state', 'estimator__verbose', 'estimator__warm_start',
'estimator', 'iid', 'n_jobs', 'param_grid', 'pre_dispatch', 'refit',
'return_train_score', 'scoring', 'verbose'])
```

[180]:
```
print(f'Best Mean Cross Validation Score is {rfc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {rfc_grid.best_params_}')
print(f'Train score is {rfc_grid.score(X_train,y_train)}')
print(f'Test score is {rfc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9527005931540298
Best Mean Cross Validation Score is {'criterion': 'gini', 'max_depth': 8,
'max_features': 'auto', 'n_estimators': 500}
Train score is 0.9534196228827101
Test score is 0.9606636838180462
```

7.4 Extra Trees

[60]:
```
from  sklearn.ensemble import ExtraTreesClassifier
etc= ExtraTreesClassifier(random_state=42)
etc_param = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [2,4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
etc_grid = GridSearchCV(etc, etc_param,cv=5, return_train_score=True, )
etc_grid.fit(X_train,y_train)
```

[60]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0,
```

```
                                       class_weight=None, criterion='gini',
                                       max_depth=None, max_features='auto',
                                       max_leaf_nodes=None,
                                       max_samples=None,
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1,
                                       min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=100, n_jobs=None,
                                       oob_score=False, random_state=42,
                                       verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [200, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

[38]:
```
etc_grid.get_params().keys()
```

[38]:
```
dict_keys(['cv', 'error_score', 'estimator__bootstrap', 'estimator__ccp_alpha',
'estimator__class_weight', 'estimator__criterion', 'estimator__max_depth',
'estimator__max_features', 'estimator__max_leaf_nodes',
'estimator__max_samples', 'estimator__min_impurity_decrease',
'estimator__min_impurity_split', 'estimator__min_samples_leaf',
'estimator__min_samples_split', 'estimator__min_weight_fraction_leaf',
'estimator__n_estimators', 'estimator__n_jobs', 'estimator__oob_score',
'estimator__random_state', 'estimator__verbose', 'estimator__warm_start',
'estimator', 'iid', 'n_jobs', 'param_grid', 'pre_dispatch', 'refit',
'return_train_score', 'scoring', 'verbose'])
```

[61]:
```
print(f'Best Mean Cross Validation Score is {etc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {etc_grid.best_params_}')
print(f'Train score is {etc_grid.score(X_train,y_train)}')
print(f'Test score is {etc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9530201457930995
Best Mean Cross Validation Score is {'criterion': 'gini', 'max_depth': 8,
'max_features': 'auto', 'n_estimators': 500}
Train score is 0.9534995206136145
Test score is 0.9606636838180462
```

7.5 Ada Boost

[62]:
```
from sklearn.ensemble import AdaBoostClassifier
adc_dtree␣
 ↪=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),random_state=42)
adc_dtree_param = {
              'base_estimator__criterion' : ["gini", "entropy"],
              'base_estimator__splitter' :   ["best", "random"],
              'base_estimator__max_depth' : [2,4,6],
              'n_estimators' : [100,150],
              'learning_rate' : [0.5,1.0,2],
             }
adc_dtree_grid = GridSearchCV(adc_dtree, adc_dtree_param,cv=5,␣
 ↪return_train_score=True, )
adc_dtree_grid.fit(X_train,y_train)
```

[62]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=None,
max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
presort='...
                                         learning_rate=1.0, n_estimators=50,
                                         random_state=42),
             iid='deprecated', n_jobs=None,
             param_grid={'base_estimator__criterion': ['gini', 'entropy'],
                         'base_estimator__max_depth': [2, 4, 6],
                         'base_estimator__splitter': ['best', 'random'],
                         'learning_rate': [0.5, 1.0, 2],
                         'n_estimators': [100, 150]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

[40]:
```
adc_dtree_grid.get_params().keys()
```

[40]:
```
dict_keys(['cv', 'error_score', 'estimator__algorithm',
'estimator__base_estimator__ccp_alpha',
'estimator__base_estimator__class_weight',
'estimator__base_estimator__criterion', 'estimator__base_estimator__max_depth',
'estimator__base_estimator__max_features',
'estimator__base_estimator__max_leaf_nodes',
```

```
'estimator__base_estimator__min_impurity_decrease',
'estimator__base_estimator__min_impurity_split',
'estimator__base_estimator__min_samples_leaf',
'estimator__base_estimator__min_samples_split',
'estimator__base_estimator__min_weight_fraction_leaf',
'estimator__base_estimator__presort', 'estimator__base_estimator__random_state',
'estimator__base_estimator__splitter', 'estimator__base_estimator',
'estimator__learning_rate', 'estimator__n_estimators',
'estimator__random_state', 'estimator', 'iid', 'n_jobs', 'param_grid',
'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

[63]:
```
print(f'Best Mean Cross Validation Score is {adc_dtree_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {adc_dtree_grid.best_params_}')
print(f'Train score is {adc_dtree_grid.score(X_train,y_train)}')
print(f'Test score is {adc_dtree_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9574145211050255
Best Mean Cross Validation Score is {'base_estimator__criterion': 'gini',
'base_estimator__max_depth': 6, 'base_estimator__splitter': 'random',
'learning_rate': 0.5, 'n_estimators': 100}
Train score is 0.9721156919143497
Test score is 0.9679343773303505
```

7.6 Gradient Boost

[64]:
```
from  sklearn.ensemble import GradientBoostingClassifier
gbc= GradientBoostingClassifier(random_state=42)
gbc_param = {
              'max_depth' : [2,3,4],
              'n_estimators' : [100,150],
              'learning_rate' : [0.5,1.0,2],
             }
gbc_grid = GridSearchCV(gbc, gbc_param,cv=5, return_train_score=True, )
gbc_grid.fit(X_train,y_train)
```

[64]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
```

```
                                                   n_iter_no_change=None,
                                                   presort='deprecated',
                                                   random_state=42,
                                                   subsample=1.0, tol=0.0001,
                                                   validation_fraction=0.1,
                                                   verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.5, 1.0, 2], 'max_depth': [2, 3, 4],
                         'n_estimators': [100, 150]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

[42]:
```
gbc_grid.get_params().keys()
```

[42]:
```
dict_keys(['cv', 'error_score', 'estimator__ccp_alpha', 'estimator__criterion',
'estimator__init', 'estimator__learning_rate', 'estimator__loss',
'estimator__max_depth', 'estimator__max_features', 'estimator__max_leaf_nodes',
'estimator__min_impurity_decrease', 'estimator__min_impurity_split',
'estimator__min_samples_leaf', 'estimator__min_samples_split',
'estimator__min_weight_fraction_leaf', 'estimator__n_estimators',
'estimator__n_iter_no_change', 'estimator__presort', 'estimator__random_state',
'estimator__subsample', 'estimator__tol', 'estimator__validation_fraction',
'estimator__verbose', 'estimator__warm_start', 'estimator', 'iid', 'n_jobs',
'param_grid', 'pre_dispatch', 'refit', 'return_train_score', 'scoring',
'verbose'])
```

[106]:
```
print(f'Best Mean Cross Validation Score is {gbc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {gbc_grid.best_params_}')
print(f'Train score is {gbc_grid.score(X_train,y_train)}')
print(f'Test score is {gbc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9554170139602445
Best Mean Cross Validation Score is {'learning_rate': 0.5, 'max_depth': 4,
'n_estimators': 150}
Train score is 0.9653243847874721
Test score is 0.9623415361670395
```

7.7 XGBoost

[66]:
```
from xgboost import XGBClassifier
from  xgboost import XGBClassifier
xgbc= XGBClassifier(random_state=42,early_stopping_rounds=2,objective= 'binary:
 ↪logistic')
xgbc_param = {
              'max_depth' : [2,4,6],
              'n_estimators' : [50,100,150],
              'learning_rate' : [0.1,0.5,0.6,0.8],
               'min_child_weight' : [1,3,5,7],
                'subsample':[0.6,0.7,0.8,0.9,1]
```

```
            }
xgbc_grid = GridSearchCV(xgbc, xgbc_param,cv=5, return_train_score=True, )
```

[45]:
```
xgbc_grid.get_params().keys()
```

[45]:
```
dict_keys(['cv', 'error_score', 'estimator__objective', 'estimator__base_score',
'estimator__booster', 'estimator__colsample_bylevel',
'estimator__colsample_bynode', 'estimator__colsample_bytree',
'estimator__gamma', 'estimator__gpu_id', 'estimator__importance_type',
'estimator__interaction_constraints', 'estimator__learning_rate',
'estimator__max_delta_step', 'estimator__max_depth',
'estimator__min_child_weight', 'estimator__missing',
'estimator__monotone_constraints', 'estimator__n_estimators',
'estimator__n_jobs', 'estimator__num_parallel_tree', 'estimator__random_state',
'estimator__reg_alpha', 'estimator__reg_lambda', 'estimator__scale_pos_weight',
'estimator__subsample', 'estimator__tree_method',
'estimator__validate_parameters', 'estimator__verbosity',
'estimator__early_stopping_rounds', 'estimator', 'iid', 'n_jobs', 'param_grid',
'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

[68]:
```
xgbc_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {xgbc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {xgbc_grid.best_params_}')
print(f'Train score is {xgbc_grid.score(X_train,y_train)}')
print(f'Test score is {xgbc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.957734137565273
Best Mean Cross Validation Score is {'learning_rate': 0.5, 'max_depth': 6,
'min_child_weight': 1, 'n_estimators': 100, 'subsample': 0.7}
Train score is 0.9688398849472675
Test score is 0.9666293810589113
```

8 Cost Sensitive Algorithms

[84]:
```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
```

9 # Logistic Regresion

[101]:
```
param_grid = {
'class_weight': [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}],
    'penalty':['l1','l2'],
}
#apply grid search
cgrid_logreg= GridSearchCV(LogisticRegression(solver='lbfgs'), param_grid,␣
 ↪cv=10, n_jobs=-1, scoring='roc_auc')
cgrid_logreg.fit(X_train, y_train)
print("Best parameters: {}".format(cgrid_logreg.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(cgrid_logreg.
 ↪best_score_))
```

```
Best parameters: {'class_weight': {0: 1, 1: 10}, 'penalty': 'l2'}
Best Mean cross-validation score: 0.87
```

[80]:
```
cgrid_logreg.get_params().keys()
```

[80]:
```
dict_keys(['cv', 'error_score', 'estimator__C', 'estimator__class_weight',
'estimator__dual', 'estimator__fit_intercept', 'estimator__intercept_scaling',
'estimator__l1_ratio', 'estimator__max_iter', 'estimator__multi_class',
'estimator__n_jobs', 'estimator__penalty', 'estimator__random_state',
'estimator__solver', 'estimator__tol', 'estimator__verbose',
'estimator__warm_start', 'estimator', 'iid', 'n_jobs', 'param_grid',
'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

9.1 SVC

[102]:
```
param_grid = {
'class_weight': [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}],
}
#apply grid search
cgrid_svc= GridSearchCV(SVC(), param_grid, cv=10, n_jobs=-1, scoring='roc_auc')
cgrid_svc.fit(X_train, y_train)
print("Best parameters: {}".format(cgrid_svc.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(cgrid_svc.best_score_))
```

```
Best parameters: {'class_weight': {0: 1, 1: 10}}
Best Mean cross-validation score: 0.90
```

[89]:
```
cgrid_svc.get_params().keys()
```

[89]:
```
dict_keys(['cv', 'error_score', 'estimator__C', 'estimator__break_ties',
'estimator__cache_size', 'estimator__class_weight', 'estimator__coef0',
'estimator__decision_function_shape', 'estimator__degree', 'estimator__gamma',
'estimator__kernel', 'estimator__max_iter', 'estimator__probability',
```

```
'estimator__random_state', 'estimator__shrinking', 'estimator__tol',
'estimator__verbose', 'estimator', 'iid', 'n_jobs', 'param_grid',
'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

9.2 Decision Tree

[103]:
```
#decison tree
param_grid = {
'class_weight': [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}],
}
#apply grid search
cgrid_dtree= GridSearchCV(DecisionTreeClassifier(), param_grid, cv=10,␣
 ↪n_jobs=-1, scoring='roc_auc')
cgrid_dtree.fit(X_train, y_train)
print("Best parameters: {}".format(cgrid_dtree.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(cgrid_dtree.
 ↪best_score_))
```

```
Best parameters: {'class_weight': {0: 10, 1: 1}}
Best Mean cross-validation score: 0.85
```

[ ]:
```
cgrid_dtree.get_params().keys()
```

9.3 XG BOOST

[104]:
```
param_grid = {
'scale_pos_weight': [1, 10, 25, 50, 75, 99, 100, 1000],
}
#apply grid search
cgrid_xgboost= GridSearchCV(XGBClassifier(), param_grid, cv=10, n_jobs=-1,␣
 ↪scoring='roc_auc')
cgrid_xgboost.fit(X_train, y_train)
print("Best parameters: {}".format(cgrid_xgboost.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(cgrid_xgboost.
 ↪best_score_))
```

```
Best parameters: {'scale_pos_weight': 1}
Best Mean cross-validation score: 0.92
```

[97]:
```
cgrid_xgboost.get_params().keys()
```

[97]:
```
dict_keys(['cv', 'error_score', 'estimator__objective', 'estimator__base_score',
'estimator__booster', 'estimator__colsample_bylevel',
'estimator__colsample_bynode', 'estimator__colsample_bytree',
'estimator__gamma', 'estimator__gpu_id', 'estimator__importance_type',
'estimator__interaction_constraints', 'estimator__learning_rate',
```

```
'estimator__max_delta_step', 'estimator__max_depth',
'estimator__min_child_weight', 'estimator__missing',
'estimator__monotone_constraints', 'estimator__n_estimators',
'estimator__n_jobs', 'estimator__num_parallel_tree', 'estimator__random_state',
'estimator__reg_alpha', 'estimator__reg_lambda', 'estimator__scale_pos_weight',
'estimator__subsample', 'estimator__tree_method',
'estimator__validate_parameters', 'estimator__verbosity', 'estimator', 'iid',
'n_jobs', 'param_grid', 'pre_dispatch', 'refit', 'return_train_score',
'scoring', 'verbose'])
```

9.4 Random Forest

[151]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
rf = RandomForestClassifier(n_estimators=100, class_weight='balanced')
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores_rf = cross_val_score(rf, X_train, y_train, scoring='roc_auc', cv=cv,␣
 ↪n_jobs=-1)
print('Mean ROC AUC: %.3f' % scores_rf.mean())
```

```
Mean ROC AUC: 0.897
```

10 Extra Trees

[150]:
```
rf = ExtraTreesClassifier(n_estimators=50, class_weight='balanced')
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores_etc = cross_val_score(rf, X_train, y_train, scoring='roc_auc', cv=cv,␣
 ↪n_jobs=-1)
print('Mean ROC AUC: %.3f' % scores_etc.mean())
```

```
Mean ROC AUC: 0.882
```

10.1 Bagging Decision Tree

[152]:
```
from imblearn.ensemble import BalancedBaggingClassifier
b_dtree = BalancedBaggingClassifier()
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores_btree = cross_val_score(b_dtree, X_train, y_train, scoring='roc_auc',␣
 ↪cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % scores_btree.mean())
```

```
Mean ROC AUC: 0.914
```

11 Data Sampling Algorithm

SMOTE

[109]:
```
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
```

11.1 Logistic Regression

[136]:
```
#logistic regression
pipe_roc_lg = Pipeline([('smote',SMOTE()),('lg',LogisticRegression())])
param_roc_lg = {'smote__k_neighbors': [1,2,3,4,5]}
logrid_lg= GridSearchCV(pipe_roc_lg,param_roc_lg, cv=10, n_jobs=-1,␣
 ↪scoring='roc_auc')
logrid_lg.fit(X_train, y_train)
print("Best parameters: {}".format(logrid_lg.best_params_))
print("Best Mean cross-validation score: {:.4f}".format(logrid_lg.best_score_))
```

```
Best parameters: {'smote__k_neighbors': 1}
Best Mean cross-validation score: 0.8637
```

[111]:
```
logrid_lg.get_params().keys()
```

[111]:
```
dict_keys(['cv', 'error_score', 'estimator__memory', 'estimator__steps',
'estimator__verbose', 'estimator__smote', 'estimator__lg',
'estimator__smote__k_neighbors', 'estimator__smote__n_jobs',
'estimator__smote__random_state', 'estimator__smote__sampling_strategy',
'estimator__lg__C', 'estimator__lg__class_weight', 'estimator__lg__dual',
'estimator__lg__fit_intercept', 'estimator__lg__intercept_scaling',
'estimator__lg__l1_ratio', 'estimator__lg__max_iter',
'estimator__lg__multi_class', 'estimator__lg__n_jobs', 'estimator__lg__penalty',
'estimator__lg__random_state', 'estimator__lg__solver', 'estimator__lg__tol',
'estimator__lg__verbose', 'estimator__lg__warm_start', 'estimator', 'iid',
'n_jobs', 'param_grid', 'pre_dispatch', 'refit', 'return_train_score',
'scoring', 'verbose'])
```

11.2 Ridge Regression using SMOTE

[115]:
```
pipe_rand_smote = Pipeline([('smote', SMOTE()), ('model', Ridge())])
param_grid = {
    # try different feature engineering parameters
    'smote__k_neighbors': range(1,10),
    #'model__n_neighbors':range(1,5),
    #'model__max_depth': [2,3,4,5,6],
    #'model__alpha': [0.1,1,10,100,80,70]
    'model__alpha': range(60,88,1)
}

#apply grid search
gridridge_smote= GridSearchCV(pipe_rand_smote, param_grid, cv=5, n_jobs=2,␣
 ↪scoring='roc_auc')
```

```
gridridge_smote.fit(X_train, y_train)

print("Best parameters: {}".format(gridridge_smote.best_params_))
print("Best Mean cross-validation score: {:.4f}".format(gridridge_smote.
 ↪best_score_))
```

```
Best parameters: {'model__alpha': 60, 'smote__k_neighbors': 1}
Best Mean cross-validation score: 0.8654
```

[ ]:
```
gridridge_smote.get_params().keys()
```

11.3 KNN regression using SMOTE

[48]:
```
pipe_rand_smote = Pipeline([('smote', SMOTE()), ('model',␣
 ↪KNeighborsClassifier())])
param_grid = {
    # try different feature engineering parameters
    'smote__k_neighbors': range(1,10),
    #'model__n_neighbors':range(1,5),
    #'model__max_depth': [2,3,4,5,6],
}

#apply grid search
grid_smote= GridSearchCV(pipe_rand_smote, param_grid, cv=5, n_jobs=2,␣
 ↪scoring='roc_auc')
grid_smote.fit(X_train, y_train)

print("Best parameters: {}".format(grid_smote.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_smote.best_score_))
```

```
Best parameters: {'smote__k_neighbors': 6}
Best Mean cross-validation score: 0.85
```

[49]:
```
print("KNN Train Performance: ", grid_smote.score(X_train,y_train))


# Check test data set performance
print("KNN Test Performance: ", grid_smote.score(X_test,y_test))
```

```
KNN Train Performance:  0.9138722769103182
KNN Test Performance:  0.8760542629742415
```

11.4 Lasso Regression Using Smote

[106]:
```
pipe_rand_smote = Pipeline([('smote', SMOTE()), ('model', Lasso())])
param_grid = {
    # try different feature engineering parameters
    #'smote__k_neighbors': range(1,10),
    #'model__n_neighbors':range(1,5),
    #'model__max_depth': [2,3,4,5,6],
    'model__alpha':[0.001,0.1,1,10],

}

#apply grid search
grid_smote= GridSearchCV(pipe_rand_smote, param_grid, cv=10, n_jobs=-1,␣
 ↪scoring='roc_auc')
grid_smote.fit(X_train, y_train)

print("Best parameters: {}".format(grid_smote.best_params_))
print("Best Mean cross-validation score: {:.4f}".format(grid_smote.best_score_))
```

```
Best parameters: {'model__alpha': 0.001}
Best Mean cross-validation score: 0.8637
```

[91]:
```
grid_smote.get_params().keys()
```

[91]:
```
dict_keys(['cv', 'error_score', 'estimator__memory', 'estimator__steps',
'estimator__verbose', 'estimator__smote', 'estimator__model',
'estimator__smote__k_neighbors', 'estimator__smote__n_jobs',
'estimator__smote__random_state', 'estimator__smote__sampling_strategy',
'estimator__model__alpha', 'estimator__model__copy_X',
'estimator__model__fit_intercept', 'estimator__model__max_iter',
'estimator__model__normalize', 'estimator__model__positive',
'estimator__model__precompute', 'estimator__model__random_state',
'estimator__model__selection', 'estimator__model__tol',
'estimator__model__warm_start', 'estimator', 'iid', 'n_jobs', 'param_grid',
'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

11.5 Decision tree

[123]:
```
pipe_roc_dtree =␣
 ↪Pipeline([('smote',SMOTE()),('dtree',DecisionTreeClassifier(max_depth=5))])
param_roc_dtree = {'smote__k_neighbors': [1,2,3,4,5],
                   'dtree__max_leaf_nodes': [10,20],
                    'dtree__max_depth':[10,20,40]
                   }
ogrid_dtree= GridSearchCV(pipe_roc_dtree,param_roc_dtree, cv=10, n_jobs=2,␣
 ↪scoring='roc_auc')
ogrid_dtree.fit(X_train, y_train)
```

```
print("Best parameters: {}".format(ogrid_dtree.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(ogrid_dtree.
 ↪best_score_))
```

```
Best parameters: {'dtree__max_depth': 10, 'dtree__max_leaf_nodes': 20,
'smote__k_neighbors': 3}
Best Mean cross-validation score: 0.86
```

[117]:
```
ogrid_dtree.get_params().keys()
```

[117]:
```
dict_keys(['cv', 'error_score', 'estimator__memory', 'estimator__steps',
'estimator__verbose', 'estimator__smote', 'estimator__dtree',
'estimator__smote__k_neighbors', 'estimator__smote__n_jobs',
'estimator__smote__random_state', 'estimator__smote__sampling_strategy',
'estimator__dtree__ccp_alpha', 'estimator__dtree__class_weight',
'estimator__dtree__criterion', 'estimator__dtree__max_depth',
'estimator__dtree__max_features', 'estimator__dtree__max_leaf_nodes',
'estimator__dtree__min_impurity_decrease',
'estimator__dtree__min_impurity_split', 'estimator__dtree__min_samples_leaf',
'estimator__dtree__min_samples_split',
'estimator__dtree__min_weight_fraction_leaf', 'estimator__dtree__presort',
'estimator__dtree__random_state', 'estimator__dtree__splitter', 'estimator',
'iid', 'n_jobs', 'param_grid', 'pre_dispatch', 'refit', 'return_train_score',
'scoring', 'verbose'])
```

11.6 Randomforest

[176]:
```
pipe_roc_rf =␣
 ↪Pipeline([('smote',SMOTE()),('rf',RandomForestClassifier(n_estimators=100,criterion='gini',
                                                                    ␣
 ↪max_features='auto'))])
param_roc_rf = {'smote__k_neighbors': [4,5,10,20,50,70,100],
                'rf__max_depth' : [4,5,10,20,30]
               }
ogrid_rf= GridSearchCV(pipe_roc_rf,param_roc_rf, cv=10, n_jobs=-1,␣
 ↪scoring='roc_auc')
ogrid_rf.fit(X_train, y_train)
print("Best parameters: {}".format(ogrid_rf.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(ogrid_rf.best_score_))
```

```
Best parameters: {'rf__max_depth': 20, 'smote__k_neighbors': 5}
Best Mean cross-validation score: 0.91
```

[126]:
```
ogrid_rf.get_params().keys()
```

[126]:
```
dict_keys(['cv', 'error_score', 'estimator__memory', 'estimator__steps',
'estimator__verbose', 'estimator__smote', 'estimator__rf',
```

```
'estimator__smote__k_neighbors', 'estimator__smote__n_jobs',
'estimator__smote__random_state', 'estimator__smote__sampling_strategy',
'estimator__rf__bootstrap', 'estimator__rf__ccp_alpha',
'estimator__rf__class_weight', 'estimator__rf__criterion',
'estimator__rf__max_depth', 'estimator__rf__max_features',
'estimator__rf__max_leaf_nodes', 'estimator__rf__max_samples',
'estimator__rf__min_impurity_decrease', 'estimator__rf__min_impurity_split',
'estimator__rf__min_samples_leaf', 'estimator__rf__min_samples_split',
'estimator__rf__min_weight_fraction_leaf', 'estimator__rf__n_estimators',
'estimator__rf__n_jobs', 'estimator__rf__oob_score',
'estimator__rf__random_state', 'estimator__rf__verbose',
'estimator__rf__warm_start', 'estimator', 'iid', 'n_jobs', 'param_grid',
'pre_dispatch', 'refit', 'return_train_score', 'scoring', 'verbose'])
```

[183]:
```
print("Train Score",ogrid_rf.score(X_train, y_train))

print("Test Score",ogrid_rf.score(X_test, y_test))
```

```
Train Score 0.9628272474132682
Test Score 0.9213638807682709
```

11.7 SVM

[ ]:
```
pipe_roc_svm = Pipeline([('smote',SMOTE()),('svm',svm.SVC(C=10,gamma=0.
 ↪1,kernel='rbf'))])
param_roc_svm = {'smote__k_neighbors': [1,2,3,4,5]}
ogrid_svm= GridSearchCV(pipe_roc_svm,param_roc_svm, cv=10, n_jobs=-1,␣
 ↪scoring='roc_auc')
ogrid_svm.fit(X_train, y_train)
print("Best parameters: {}".format(ogrid_svm.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(ogrid_svm.best_score_))
```

11.8 XGB

[135]:
```
pipe_roc_xgb = Pipeline([('smote',SMOTE()),('xgb',XGBClassifier(random_state=42,
                                                                ␣
 ↪early_stopping_rounds=2,
                                                                ␣
 ↪n_estimators=100,
                                                                 objective␣
 ↪='binary:logistic',
                                                                 max_depth = 4,
                                                                ))])
param_roc_xgb = {'smote__k_neighbors': [1,2,3,4,5],
'xgb__learning_rate' : [0.1,0.5]}
ogrid_xgb= GridSearchCV(pipe_roc_xgb,param_roc_xgb, cv=10, n_jobs=-1,␣
 ↪scoring='roc_auc')
```

```
ogrid_xgb.fit(X_train, y_train)
print("Best parameters: {}".format(ogrid_xgb.best_params_))
print("Best Mean cross-validation score: {:.4f}".format(ogrid_xgb.best_score_))
```

```
Best parameters: {'smote__k_neighbors': 4, 'xgb__learning_rate': 0.1}
Best Mean cross-validation score: 0.90
```

12 Top models

[162]:
```
classifiers={'knn': grid_knn,
             'dtree':grid_dtree,
              'bag_dtree':bag_dtree_grid,
             'paste_dtree': paste_dtree_grid,
             'rfc': rfc_grid,
              'etc': etc_grid,
              'adc_dtree':adc_dtree_grid,
              'gbc': gbc_grid,
              'xgbc': xgbc_grid,
             'Elastic':grid_elasticnet,
              'Ridge':grid_ridge,
              'Lasso':grid_lasso,
              'XGB Smote':ogrid_xgb,
              'rfc_smote':ogrid_rf,
              'ridge_smote':gridridge_smote,
              'tree_smote':ogrid_dtree,
              'cost_logistic':cgrid_logreg,
              'cost_svc':cgrid_svc,
              'cost_tree':cgrid_dtree,
              'cost_xgboost':cgrid_xgboost,

             }
```

[163]:
```
results_mean_std = []
for  key, value in classifiers.items():
    mean = value.cv_results_['mean_test_score'][value.best_index_]
    std=value.cv_results_['std_test_score'][value.best_index_]

    results_mean_std.append({
        "model": key,
        "mean": mean,
        "std": std
    })
```

[164]:
```
# Create a Pandas DataFrame with the mean+std results
accuracy_df = pd.DataFrame(results_mean_std, columns=['model', 'mean', 'std'])
```

[165]:
```
# Show the accuracy dataframe

accuracy_df.sort_values(by=['mean'], inplace=True,ascending=False)
accuracy_df
```

[165]:

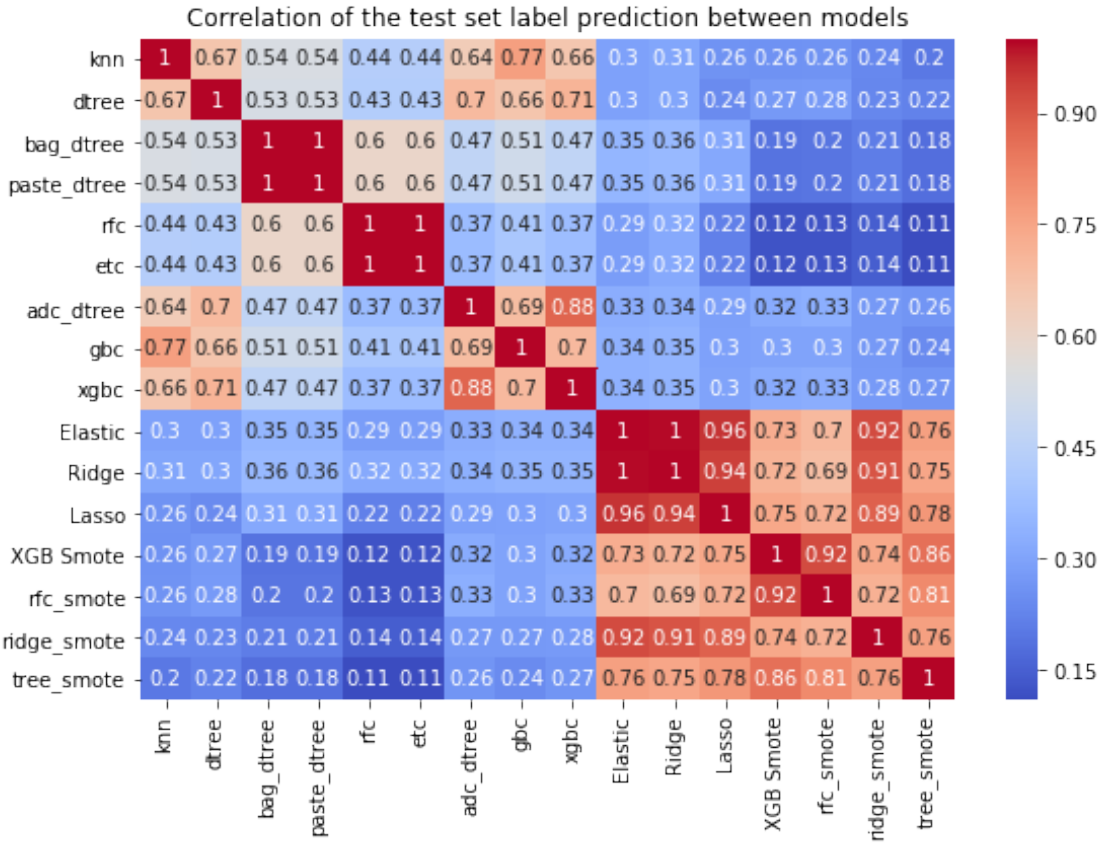
	model	mean	std
8	xgbc	0.957734	0.003495
6	adc_dtree	0.957415	0.002996
7	gbc	0.955417	0.003233
5	etc	0.953020	0.001442
4	rfc	0.952701	0.001441
2	bag_dtree	0.951981	0.001589
3	paste_dtree	0.951981	0.001589
19	cost_xgboost	0.917265	0.012525
13	rfc_smote	0.903501	0.013991
17	cost_svc	0.903121	0.014274
12	XGB Smote	0.899929	0.015279
16	cost_logistic	0.873235	0.017760
1	dtree	0.869793	0.027037
14	ridge_smote	0.865384	0.008775
10	Ridge	0.858237	0.024254
15	tree_smote	0.857915	0.017975
18	cost_tree	0.854562	0.032401
11	Lasso	0.844302	0.026594
0	knn	0.796630	0.030336
9	Elastic	0.116465	0.021860

[142]:
```
# Create a prediction of all models on the test set
predictions_all = {}
for  key, value in classifiers.items():
    # Get best estimator
    best_model = value.best_estimator_

    # Predict test labels
    predictions = best_model.predict(X_test)

    # Save predictions to a list
    predictions_all[key] = predictions
```

[143]:
```
pred = pd.DataFrame(predictions_all)
```

[144]:
```
# Plot a heatmap of all correlations for easier visualization
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(9,6))
g = sns.heatmap(pred.corr(), annot=True, cmap='coolwarm', ax=ax)
g.set_title('Correlation of the test set label prediction between models')
```

[144]:
```
Text(0.5, 1, 'Correlation of the test set label prediction between models')
```



[145]:
```
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=True)
    return au_corr[0:n]
```

[146]:
```
print("Top Absolute Correlations")
print(get_top_abs_correlations(pred, 5))
```

```
Top Absolute Correlations
rfc  tree_smote    0.109838
etc  tree_smote    0.109838
```

```
rfc  XGB Smote     0.124444
etc  XGB Smote     0.124444
rfc  rfc_smote     0.126478
dtype: float64
```

12.1 Stacking

[166]:
```
#top 5 estimators
estimators_ = [('XGBC',xgbc_grid.best_estimator_),
('Ada Boost',adc_dtree_grid.best_estimator_),('GBC',gbc_grid.best_estimator_),
('Extra Trees',etc_grid.best_estimator_),('Random Forrest',rfc_grid.
 ↪best_estimator_)]
```

[172]:
```
from sklearn.ensemble import StackingClassifier
sclf2 = StackingClassifier(estimators= estimators_,
                           final_estimator=LogisticRegression())
sclf2_param = {'final_estimator__C' : [0.1,0.2],
               'stack_method':['auto']
              }

sclf2_grid = GridSearchCV(sclf2, sclf2_param,cv=10,␣
 ↪return_train_score=True,scoring='roc_auc' )
sclf2_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {sclf2_grid.best_score_}')
print(f'Best Mean Cross Validation params is {sclf2_grid.best_params_}')
print(f'Train score is {sclf2_grid.score(X_train,y_train)}')
print(f'Test score is {sclf2_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.902891367359539
Best Mean Cross Validation params is {'final_estimator__C': 0.2, 'stack_method':
'auto'}
Train score is 0.9699396645559605
Test score is 0.9247255164441441
```

[171]:
```
from sklearn.ensemble import StackingClassifier
sclf2 = StackingClassifier(estimators= estimators_,
                           final_estimator=XGBClassifier())
sclf2_param = {'final_estimator__C' : [0.1,0.2],
               'stack_method':['auto']
              }

sclf2_grid = GridSearchCV(sclf2, sclf2_param,cv=10,␣
 ↪return_train_score=True,scoring='roc_auc' )
sclf2_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {sclf2_grid.best_score_}')
print(f'Best Mean Cross Validation params is {sclf2_grid.best_params_}')
print(f'Train score is {sclf2_grid.score(X_train,y_train)}')
print(f'Test score is {sclf2_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.9100134401167036
Best Mean Cross Validation params is {'final_estimator__C': 0.1, 'stack_method':
'auto'}
Train score is 0.9509360617692855
Test score is 0.9193606203246895
```

[169]:
```
sclf2_grid.get_params().keys()
```

[169]:
```
dict_keys(['cv', 'error_score', 'estimator__cv', 'estimator__estimators',
'estimator__final_estimator__C', 'estimator__final_estimator__class_weight',
'estimator__final_estimator__dual', 'estimator__final_estimator__fit_intercept',
'estimator__final_estimator__intercept_scaling',
'estimator__final_estimator__l1_ratio', 'estimator__final_estimator__max_iter',
'estimator__final_estimator__multi_class', 'estimator__final_estimator__n_jobs',
'estimator__final_estimator__penalty',
'estimator__final_estimator__random_state',
'estimator__final_estimator__solver', 'estimator__final_estimator__tol',
'estimator__final_estimator__verbose', 'estimator__final_estimator__warm_start',
'estimator__final_estimator', 'estimator__n_jobs', 'estimator__passthrough',
'estimator__stack_method', 'estimator__verbose', 'estimator__XGBC',
'estimator__Ada Boost', 'estimator__GBC', 'estimator__Extra Trees',
'estimator__Random Forrest', 'estimator__XGBC__objective',
'estimator__XGBC__base_score', 'estimator__XGBC__booster',
'estimator__XGBC__colsample_bylevel', 'estimator__XGBC__colsample_bynode',
'estimator__XGBC__colsample_bytree', 'estimator__XGBC__gamma',
'estimator__XGBC__gpu_id', 'estimator__XGBC__importance_type',
'estimator__XGBC__interaction_constraints', 'estimator__XGBC__learning_rate',
'estimator__XGBC__max_delta_step', 'estimator__XGBC__max_depth',
'estimator__XGBC__min_child_weight', 'estimator__XGBC__missing',
'estimator__XGBC__monotone_constraints', 'estimator__XGBC__n_estimators',
'estimator__XGBC__n_jobs', 'estimator__XGBC__num_parallel_tree',
'estimator__XGBC__random_state', 'estimator__XGBC__reg_alpha',
'estimator__XGBC__reg_lambda', 'estimator__XGBC__scale_pos_weight',
'estimator__XGBC__subsample', 'estimator__XGBC__tree_method',
'estimator__XGBC__validate_parameters', 'estimator__XGBC__verbosity',
'estimator__XGBC__early_stopping_rounds', 'estimator__Ada Boost__algorithm',
'estimator__Ada Boost__base_estimator__ccp_alpha', 'estimator__Ada
Boost__base_estimator__class_weight', 'estimator__Ada
Boost__base_estimator__criterion', 'estimator__Ada
Boost__base_estimator__max_depth', 'estimator__Ada
Boost__base_estimator__max_features', 'estimator__Ada
Boost__base_estimator__max_leaf_nodes', 'estimator__Ada
Boost__base_estimator__min_impurity_decrease', 'estimator__Ada
Boost__base_estimator__min_impurity_split', 'estimator__Ada
Boost__base_estimator__min_samples_leaf', 'estimator__Ada
Boost__base_estimator__min_samples_split', 'estimator__Ada
Boost__base_estimator__min_weight_fraction_leaf', 'estimator__Ada
Boost__base_estimator__presort', 'estimator__Ada
```

```
Boost__base_estimator__random_state', 'estimator__Ada
Boost__base_estimator__splitter', 'estimator__Ada Boost__base_estimator',
'estimator__Ada Boost__learning_rate', 'estimator__Ada Boost__n_estimators',
'estimator__Ada Boost__random_state', 'estimator__GBC__ccp_alpha',
'estimator__GBC__criterion', 'estimator__GBC__init',
'estimator__GBC__learning_rate', 'estimator__GBC__loss',
'estimator__GBC__max_depth', 'estimator__GBC__max_features',
'estimator__GBC__max_leaf_nodes', 'estimator__GBC__min_impurity_decrease',
'estimator__GBC__min_impurity_split', 'estimator__GBC__min_samples_leaf',
'estimator__GBC__min_samples_split', 'estimator__GBC__min_weight_fraction_leaf',
'estimator__GBC__n_estimators', 'estimator__GBC__n_iter_no_change',
'estimator__GBC__presort', 'estimator__GBC__random_state',
'estimator__GBC__subsample', 'estimator__GBC__tol',
'estimator__GBC__validation_fraction', 'estimator__GBC__verbose',
'estimator__GBC__warm_start', 'estimator__Extra Trees__bootstrap',
'estimator__Extra Trees__ccp_alpha', 'estimator__Extra Trees__class_weight',
'estimator__Extra Trees__criterion', 'estimator__Extra Trees__max_depth',
'estimator__Extra Trees__max_features', 'estimator__Extra
Trees__max_leaf_nodes', 'estimator__Extra Trees__max_samples', 'estimator__Extra
Trees__min_impurity_decrease', 'estimator__Extra Trees__min_impurity_split',
'estimator__Extra Trees__min_samples_leaf', 'estimator__Extra
Trees__min_samples_split', 'estimator__Extra Trees__min_weight_fraction_leaf',
'estimator__Extra Trees__n_estimators', 'estimator__Extra Trees__n_jobs',
'estimator__Extra Trees__oob_score', 'estimator__Extra Trees__random_state',
'estimator__Extra Trees__verbose', 'estimator__Extra Trees__warm_start',
'estimator__Random Forrest__bootstrap', 'estimator__Random Forrest__ccp_alpha',
'estimator__Random Forrest__class_weight', 'estimator__Random
Forrest__criterion', 'estimator__Random Forrest__max_depth', 'estimator__Random
Forrest__max_features', 'estimator__Random Forrest__max_leaf_nodes',
'estimator__Random Forrest__max_samples', 'estimator__Random
Forrest__min_impurity_decrease', 'estimator__Random
Forrest__min_impurity_split', 'estimator__Random Forrest__min_samples_leaf',
'estimator__Random Forrest__min_samples_split', 'estimator__Random
Forrest__min_weight_fraction_leaf', 'estimator__Random Forrest__n_estimators',
'estimator__Random Forrest__n_jobs', 'estimator__Random Forrest__oob_score',
'estimator__Random Forrest__random_state', 'estimator__Random Forrest__verbose',
'estimator__Random Forrest__warm_start', 'estimator', 'iid', 'n_jobs',
'param_grid', 'pre_dispatch', 'refit', 'return_train_score', 'scoring',
'verbose'])
```