Kernel: Python 3 (system-wide)

In [1]:
```
# EXECUTE FIRST

# computational imports
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score,
KFold, RandomizedSearchCV
import xgboost as xgb
from scipy.stats import uniform, randint
from GPyOpt.methods import BayesianOptimization
from tpot import TPOTRegressor
from pprint import pprint

# plotting imports
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")

# for reading files from urls
import urllib.request
# display imports
from IPython.display import display, IFrame
from IPython.core.display import HTML

# import notebook styling for tables and width etc.
response =
urllib.request.urlopen('https://raw.githubusercontent.com/DataScienceU
WL/DS775v2/master/ds755.css')
HTML(response.read().decode("utf-8"));

# import warnings
import warnings
```

Project 2 Homework

For this project you're going to apply hyperparameter optimization to both a regression and a classification problem. It looks like a lot to do below, but it's mostly a matter of modifying code from the presentation.

Objective

For each of the models in problems 1 and 2 below, apply the following 4 tuning methods from the presentation: GridSearchCV, RandomSearchCV, BayesianOptimization, and TPOT.

- **For TPOT**: In Problem 1 do only hyperparameter optimization. In Problem 2 do **both** hyperparameter optimization and also run TPOT and let it choose the model. See the presentation for examples of both.

What to submit

For each problem you need to include the following:

1. A pandas table that reports:
    - The best parameters for each tuning method
    - The optimized score from the test data
    - The number of model fits used in the optimization
2. A brief discussion about which hyperparameter optimization approach worked best

Notes:

- **For problem 1**: your pandas table should include the best parameters for each of the 4 tuning methods above.
- **For problem 2**: your pandas table should include the best parameters for each of the 5 tuning methods (the 4 methods above and the TPOT model search).
- **For GridSearchCV**: you should include at least 2 or 3 values for each hyperparameter and one of those values should be the default.
- **For BayesianOptimization**: you'll have to use `int()` or `bool()` to cast the float values of the hyperparameters inside your `cv_score()` function.
- **For TPOT**: you should use a finer grid than for GridSearchCV, but not more than 10 to 20 possible values for each hyperparameter. You could lower the number of possible values to keep the search space smaller.
    - If your code is too slow you can reduce the number of cross-validation folds to 3 and if your dataset is really large you can randomly choose a smaller subset of the rows.
- Use section headers to label your work. Your summary / discussion should be more than simply "XYZ is the best model", but it also shouldn't be more than a few paragraphs and a table.

Regarding data

- You can use either the specified dataset or you can choose your own.
    - If you use your own data it should have at least 500 rows and 10 features.
    - If your data has categorical features you'll need "one hot" encode it (convert categorical features into multiple binary features). Here is a nice tutorial. For categories with only two values you can remove one of the two hot encoded columns.

- If you do want to use your own data, we suggest first getting things working with the suggested datasets. Finding, cleaning, and preparing data can take a lot of time.

P2.1 - Optimize Random Forest Regression

Find optimized hyperparameters for a random forest regression model.

You may use either the diabetes data used in the presentation or a dataset that you choose. **You do not need to include the TPOT general search for this problem** (use TPOT to optimize RandomForestRegressor, but don't run TOPT to choose a model). Here are ranges for a subset of the hyperparameters:

Hyperparameter	Type	Default Value	Typical Range
n_estimators	discrete / integer	100	10 to 150
max_features	continuous / float	1.0	0.05 to 1.0
min_samples_split	discrete / integer	2	2 to 20
min_samples_leaf	discrete / integer	1	1 to 20
bootstrap	discrete / boolean	True	True, False

You can add other hyperparameters to the optimization if you wish. Documentation for sklearn RandomForestRegressor

*** 15 points:

In [2]:
```
# imports in first cell of notebook
# from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
print(diabetes.DESCR)
```

Out[2]:
```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression
one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, T-Cells (a type of white blood cells)
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, thyroid stimulating hormone
      - s5      ltg, lamotrigine
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and
scaled by the standard deviation times `n_samples` (i.e. the sum of
squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani
(2004) "Least Angle Regression," Annals of Statistics (with discussion),
407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

In [3]:
```python
# import numpy as np
X = np.array(diabetes.data)
y = np.array(diabetes.target)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=123)
```

In [4]:
```python
# from sklearn.linear_model import LinearRegression
model_lr = LinearRegression()
model_lr.fit(X_train,y_train) # this could be inside the function
below too

def my_regression_results(model):
    score_test = model.score(X_test,y_test)
    print('Model r-squared score from test data:
{:0.4f}'.format(score_test))

    y_pred = model.predict(X_test)
    # import matplotlib.pyplot as plt
    plt.figure(figsize=(9,6))
    plt.plot(y_test,y_pred,'k.')
    plt.xlabel('Test Values')
    plt.ylabel('Predicted Values');

    # from sklearn.metrics import mean_squared_error
    mse = mean_squared_error(y_test,y_pred)
    rmse = np.sqrt(mse)
    print('Mean squared error on test data: {:0.2f}'.format(mse))
    print('Root mean squared error on test data:
{:0.2f}'.format(rmse))

my_regression_results(model_lr)
```

Out[4]:

```
Model r-squared score from test data: 0.5676
Mean squared error on test data: 2724.24
Root mean squared error on test data: 52.19
```

Image in a Jupyter notebook

In [5]:

```
# from sklearn.ensemble import RandomForestRegressor
```

```
rf_model = RandomForestRegressor(random_state=0)
rf_model.fit(X_train,y_train)

my_regression_results(rf_model)
```

Out[5]:
```
Model r-squared score from test data: 0.5368
Mean squared error on test data: 2918.49
Root mean squared error on test data: 54.02
```

Image in a Jupyter notebook

In [6]:
```
# Perform GridSearchCV on diabetes data

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

# define the grid
params = {
    "n_estimators": [10, 100, 150],
    "max_features": [0.05, 1],
    "min_samples_split": [2, 20],
    "min_samples_leaf": [1, 20],
    "bootstrap": [True, False]
}

# setup the grid search
grid_search = GridSearchCV(rf_model,
                           param_grid=params,
                           cv=5,
                           verbose=1,
                           n_jobs=1,
                           return_train_score=True)

grid_search.fit(X_train, y_train)
```

Out[6]:
```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 240 out of 240 | elapsed:   31.4s finished
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=0), n_jobs=1,
             param_grid={'bootstrap': [True, False], 'max_features': [0.05, 1],
                         'min_samples_leaf': [1, 20],
                         'min_samples_split': [2, 20],
                         'n_estimators': [10, 100, 150]},
             return_train_score=True, verbose=1)
```

In [7]:
```
grid_search.best_params_
```

Out[7]:
```
{'bootstrap': True,
 'max_features': 0.05,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 150}
```

In [8]:
```
my_regression_results(grid_search)
```

Out[8]:
```
Model r-squared score from test data: 0.5144
Mean squared error on test data: 3059.13
Root mean squared error on test data: 55.31
```

Image in a Jupyter notebook

In [9]:
```
# Perform RandomizedSearchCV on diabetes data

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

params = {
    "n_estimators": [10, 100, 150],
    "max_features": uniform(0.05, 1),
    "min_samples_split": randint(2, 20),
    "min_samples_leaf": randint(1, 20),
    "bootstrap": [True, False]
}

random_search = RandomizedSearchCV(
    rf_model,
    param_distributions=params,
    random_state=8675309,
    n_iter=25,
    cv=5,
    verbose=1,
    n_jobs=1,
    return_train_score=True)

random_search.fit(X_train, y_train)
```

Out[9]:
```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done 125 out of 125 | elapsed:   16.2s finished
RandomizedSearchCV(cv=5,
estimator=RandomForestRegressor(random_state=0),
                   n_iter=25, n_jobs=1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_features':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7fa9c7f426d0>,
                                        'min_samples_leaf':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7fa9c7e58880>,
                                        'min_samples_split':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7fa9c7ee7940>,
                                        'n_estimators': [10, 100, 150]},
                   random_state=8675309, return_train_score=True,
verbose=1)
```

In [10]:
```
random_search.best_params_
```

Out[10]:
```
{'bootstrap': True,
 'max_features': 0.6856719244820269,
 'min_samples_leaf': 6,
 'min_samples_split': 3,
 'n_estimators': 150}
```

In [11]:
```
my_regression_results(random_search)
```

Out[11]:
```
Model r-squared score from test data: 0.5705
Mean squared error on test data: 2705.82
Root mean squared error on test data: 52.02
```

Image in a Jupyter notebook

In [12]:
```
# Perform Bayesian Optimization on diabetes data

np.random.seed(8675309)  # seed courtesy of Tommy Tutone
# from GPyOpt.methods import BayesianOptimization
# from sklearn.model_selection import cross_val_score, KFold

hp_bounds = [{
    'name': 'n_estimators',
    'type': 'discrete',
    'domain': (10, 100, 150)
}, {
    'name': 'max_features',
    'type': 'continuous',
    'domain': (0.05 ,1.0)
}, {
    'name': 'min_samples_split',
    'type': 'discrete',
    'domain': (2, 20)
}, {
    'name': 'min_samples_leaf',
    'type': 'discrete',
    'domain': (1, 20)
}, {
    'name': 'bootstrap',
    'type': 'discrete',
    'domain': (True, False)
}]


# Optimization objective
def cv_score(hyp_parameters):
    hyp_parameters = hyp_parameters[0]
    rf_model = RandomForestRegressor(random_state=0,
                                     n_estimators=int(hyp_parameters[0]),
                                     max_features=hyp_parameters[1],

min_samples_split=int(hyp_parameters[2]),

min_samples_leaf=int(hyp_parameters[3]),
                                     bootstrap=bool(hyp_parameters[4]))
    scores = cross_val_score(rf_model,
                             X=X_train,
                             y=y_train,
                             cv=KFold(n_splits=5))
    return np.array(scores.mean())  # return average of 5-fold scores


optimizer = BayesianOptimization(f=cv_score,
                                 domain=hp_bounds,
                                 model_type='GP',
                                 acquisition_type='EI',
                                 acquisition_jitter=0.05,
                                 exact_feval=True,
                                 maximize=True,
                                 verbosity=True)

optimizer.run_optimization(max_iter=20,verbosity=True)
```

Out[12]:
```
num acquisition: 1, time elapsed: 6.51s
num acquisition: 2, time elapsed: 9.52s
num acquisition: 3, time elapsed: 14.37s
num acquisition: 4, time elapsed: 18.42s
num acquisition: 5, time elapsed: 22.25s
num acquisition: 6, time elapsed: 25.93s
num acquisition: 7, time elapsed: 29.43s
num acquisition: 8, time elapsed: 33.49s
num acquisition: 9, time elapsed: 34.94s
num acquisition: 10, time elapsed: 37.46s
num acquisition: 11, time elapsed: 39.99s
num acquisition: 12, time elapsed: 44.37s
num acquisition: 13, time elapsed: 49.50s
num acquisition: 14, time elapsed: 55.48s
num acquisition: 15, time elapsed: 58.03s
num acquisition: 16, time elapsed: 64.23s
num acquisition: 17, time elapsed: 69.25s
num acquisition: 18, time elapsed: 74.38s
num acquisition: 19, time elapsed: 79.96s
num acquisition: 20, time elapsed: 84.62s
```

In [13]:
```
best_hyp_set = {}
for i in range(len(hp_bounds)):
    if hp_bounds[i]['type'] == 'continuous':
        best_hyp_set[hp_bounds[i]['name']] = optimizer.x_opt[i]
    else:
        best_hyp_set[hp_bounds[i]['name']] = int(optimizer.x_opt[i])
best_hyp_set
```

Out[13]:
```
{'n_estimators': 150,
 'max_features': 0.2659761103329934,
 'min_samples_split': 20,
 'min_samples_leaf': 1,
 'bootstrap': 0}
```

In [14]:
```
bayopt_search = RandomForestRegressor(random_state=0,**best_hyp_set)
bayopt_search.fit(X_train,y_train)
```

Out[14]:
```
RandomForestRegressor(bootstrap=0, max_features=0.2659761103329934,
                      min_samples_split=20, n_estimators=150,
random_state=0)
```

In [15]:
```
my_regression_results(bayopt_search)
```

Out[15]:
```
Model r-squared score from test data: 0.5365
Mean squared error on test data: 2920.38
Root mean squared error on test data: 54.04
```

Image in a Jupyter notebook

In [16]:
```
# Perform TPOT on diabetes data

# from tpot import TPOTRegressor

tpot_config = {
    'sklearn.ensemble.RandomForestRegressor': {
        'n_estimators': [10, 100, 150],
        'max_features': [0.05, 1.0],
        'min_samples_split': [2,20],
        'min_samples_leaf': [2,20],
        'bootstrap': [True,False],
        'random_state':[0]
    }
}

tpot = TPOTRegressor(scoring = 'r2',
                     generations=5,
                     population_size=20,
                     verbosity=2,
                     config_dict=tpot_config,
                     cv=5,
                     random_state=8675309)
tpot.fit(X_train, y_train)
tpot.export('tpot_RFregressor.py') # export the model
```

Out[16]:
```
Generation 1 - Current best internal CV score: 0.4289008403878453
Generation 2 - Current best internal CV score: 0.43155916443830017
Generation 3 - Current best internal CV score: 0.43155916443830017
Generation 4 - Current best internal CV score: 0.43155916443830017
Generation 5 - Current best internal CV score: 0.43155916443830017
Best pipeline: RandomForestRegressor(RandomForestRegressor(input_matrix,
bootstrap=True, max_features=1.0, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=0), bootstrap=True,
max_features=0.05, min_samples_leaf=2, min_samples_split=2,
n_estimators=150, random_state=0)
```

In [17]:
```
my_regression_results(tpot)
```

Out[17]:
```
Model r-squared score from test data: 0.5508
Mean squared error on test data: 2830.06
Root mean squared error on test data: 53.20
```

Image in a Jupyter notebook

In [20]:
```python
import pandas as pd
prob1 = [['Grid SearchCV', grid_search.best_params_['n_estimators'],
grid_search.best_params_['bootstrap'],
grid_search.best_params_['max_features'],
grid_search.best_params_['min_samples_leaf'],
grid_search.best_params_['min_samples_split'], 0.5144, 240],
         ['Randomized SearchCV',
random_search.best_params_['n_estimators'],
random_search.best_params_['bootstrap'],
random_search.best_params_['max_features'],
random_search.best_params_['min_samples_leaf'],
random_search.best_params_['min_samples_split'], 0.5705, 125],
         ['Bayesian', best_hyp_set['n_estimators'],
best_hyp_set['bootstrap'], best_hyp_set['max_features'],
best_hyp_set['min_samples_leaf'], best_hyp_set['min_samples_split'],
0.5365, 125],
         ['TPOT', 100, True, 1.0, 2, 2, 0.5508, 105]]
df = pd.DataFrame(prob1, columns = ['model','n_estimators',
'bootstrap', 'max_features', 'min_samples_leaf', 'min_samples_split',
'best_score', 'num_fits'])
df
```

Out[20]:

	model	n_estimators	bootstrap	max_features	min_samples_leaf	min_samples_
0	Grid SearchCV	150	True	0.050000	1	2
1	Randomized SearchCV	150	True	0.685672	6	3
2	Bayesian	150	0	0.265976	1	20
3	TPOT	100	True	1.000000	2	2

Summary:

*** 5 points:

The Randomized Search CV had the best score of all of the approaches used on this problem with an R-squared value of 0.5705. TPOT also ran well with an R-squared value of 0.5508. TPOT required less fits to run over Randomized Search. GridSearchCV performed the worst of the four approaches and required the greatest number of fits to complete while also taking the longest to finish. I think RandomizedSearchCV and TPOT are good approaches to take with this data as they return the highest R-squared and complete the quickest.

P2.2 - Optimize XGBoost Classifier

Find optimized hyperparameters for an xgboost classifier model.

This problem contains 5 parts.

Notes:

About the data

The first cell below loads a subset of the loans default data from DS705 and your job is to predict whether a loan defaults or not. The `status_bad` column is the target column and a 1 indicates a loan that defaulted. We have selected a subset of the original data that includes 2000 each of good and bad loans. The data has already been cleaned and encoded. You're welcome to look into a different dataset, but start by getting this working and then add your own data.

This is classification, not regression

The score for each model will be accuracy and not MSE. Your summary table should include accuracy, sensitivity, and precision for each optimized model applied to the test data. (Here is a nice overview of metrics for binary classification data) that includes definitions of accuracy and such.

For the models you'll mostly just need to change 'regressor' to 'classifier', e.g. `XGBClassifier` instead of `XGBRegressor` .

Hyperparameter	Type	Default Value	Typical Range
n_estimators	discrete / integer	100	50 to 150
max_depth	discrete / integer	3	1 to 10
min_child_weight	discrete / integer	1	1 to 20
learning_rate	continuous / float	0.1	0.001 to 1
sub_sample	continuous / float	1	0.05 to 1
reg_lambda	continuous / float	1	0 to 5
reg_alpha	continuous / float	0	0 to 5

P2.2a - Loading the Data

In [22]:
```
# Do not change this cell for loading and preparing the data
import pandas as pd
import numpy as np
import sklearn.metrics as skm

X = pd.read_csv('./data/loans_subset.csv')

# split into predictors and target
# convert to numpy arrays for xgboost, OK for other models too
y = np.array(X['status_Bad']) # 1 for bad loan, 0 for good loan
X = np.array(X.drop(columns = ['status_Bad']))

# split into test and training data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.1, random_state=0)
```

P2.2b - Display Results Function

Write a function called `my_classifier_results` modeled after `my_regression_results` that applies a model to the test data and prints out the accuracy, sensitivity, precision, and the confusion matrix. There is no need to make a plot.

*** 5 points - (don't delete this cell)

In [23]:
```
# check the code in Part 3 to see how to get the confusion matrix to
help you write your function
def my_classifier_results(model):
    y_pred = model.predict(X_test)
    conf = confusion_matrix(y_test, y_pred, labels=[1,0])
    cmtx = pd.DataFrame(
    confusion_matrix(y_test, y_pred, labels=[1,0]),
    index=['true:user', 'true:not user'],
    columns =['pred:user', 'pred: not user']
    )
    display(cmtx)
    accuracy = model.score(X_test, y_test)
    precision = skm.precision_score(y_test, y_pred)
    sensitivity = float(conf[1][1])/np.sum(conf[1])
    print('Accuracy: {:0.4f}'.format(accuracy))
    print('Precision: {:0.4f}'.format(precision))
    print('Sensitivity: {:0.4f}'.format(sensitivity))

    return accuracy, precision, sensitivity
```

P2.2c - Baseline Models

Start by training some baseline models using default values of the hyperparameters. We've included logistic regression in a cell below to get you started. Use `LogisticRegression`, `RandomForestClassifier`, and `GaussianNB` (Gaussian Naive Bayes) from `sklearn`. Also use `XGBClassifier` from `xgboost` where you may need to include `objective="binary:logistic"` as an option. The default scoring method for all of the `sklearn` classifiers is accuracy. Apply `my_classifier_results` to the test data for each model.

*** 10 points - (don't delete this cell)

In [24]:

```
# We've included this code to get you started

from sklearn.linear_model import LogisticRegression

# we do need to go higher than the default iterations for the solver
to get convergence
# and the explicity declaration of the solver avoids a warning
message, otherwise
# the parameters are defaults.
logreg_model = LogisticRegression(solver='lbfgs',max_iter=1000)

logreg_model.fit(X_train, y_train)

# Use score method to get accuracy of model
# score = logreg_model.score(X_test, y_test) # this is accuracy
# print(score)

# obtaining the confusion matrix and making it look nice

from sklearn.metrics import confusion_matrix
import pandas as pd

# must put true before predictions in confusion matrix function
my_classifier_results(logreg_model)
```

Out[24]:

	pred:user	pred: not user
true:user	126	71
true:not user	110	93

```
Accuracy: 0.5475
Precision: 0.5339
Sensitivity: 0.4581
```

```
(0.5475, 0.5338983050847458, 0.458128078817734)
```

In [25]:

```
logreg_model = RandomForestClassifier(random_state=0)
logreg_model.fit(X_train,y_train)
```

```
my_classifier_results(logreg_model)
```

Out[25]:

	pred:user	pred: not user
true:user	119	78
true:not user	75	128

```
Accuracy: 0.6175
Precision: 0.6134
Sensitivity: 0.6305
```

```
(0.6175, 0.6134020618556701, 0.6305418719211823)
```

In [26]:

```
from sklearn.naive_bayes import GaussianNB
logreg_model = GaussianNB()
logreg_model.fit(X_train, y_train)
my_classifier_results(logreg_model)
```

Out[26]:

	pred:user	pred: not user
true:user	160	37
true:not user	139	64

```
Accuracy: 0.5600
Precision: 0.5351
Sensitivity: 0.3153
```

```
(0.56, 0.5351170568561873, 0.31527093596059114)
```

In [27]:

```
xgbr_model = xgb.XGBClassifier(objective ='binary:logistic')
xgbr_model.fit(X_train,y_train)

my_classifier_results(xgbr_model)
```

Out[27]:

	pred:user	pred: not user
true:user	121	76
true:not user	70	133

```
Accuracy: 0.6350
Precision: 0.6335
Sensitivity: 0.6552
```

```
(0.635, 0.6335078534031413, 0.6551724137931034)
```

P2.2d - Hyperparameter Optimization

Now use the four hyperparameter optimization techniques on `XGBClassifier` and TPOT general model optimization. Apply `my_classifer_results` to the test data in each case.

- Feel free to use 3 folds instead of 5 for cross validation to speed things up.
- Choose a very small number of iterations, population size, etc. until you're sure things are working correctly, then turn up the numbers. General TPOT optimization will take a while (fair warning: it took about 30 minutes on my Macbook Pro with generations = 10, population_size=40, and cv=5)

- The hyperparameters to consider for are the same as they were in the presentation , but here they are again for convenience:

*** 10 points - (don't delete this cell)

In [28]:
```
# run GridSearchCV with our xgbr_model to find better hyperparameters
# from sklearn.model_selection import GridSearchCV

# define the grid
params = {
    "n_estimators": [5,10, 11],
    "max_depth": [1, 3, 10],
    "min_child_weight": [1, 20],
    "learning_rate": [0.001, 0.1, 1],
    "subsample": [0.05, 1],
    "reg_lambda": [0, 1, 5],
    "reg_alpha:": [0, 5]
}

# setup the grid search
grid_search = GridSearchCV(xgbr_model,
                           param_grid=params,
                           cv=3,
                           verbose=0,
                           n_jobs=1,
                           return_train_score=True)

grid_search.fit(X_train, y_train)
```

Out[28]:
```
[19:27:52] WARNING: /workspace/src/learner.cc:480: 
Parameters: { reg_alpha: } might not be used.

  This may not be accurate due to some parameters are only used in 
language bindings but
  passed down to XGBoost core.  Or some parameters are not used but 
slip through this
  verification. Please open an issue if you find above cases.


[19:27:52] WARNING: /workspace/src/learner.cc:480: 
Parameters: { reg_alpha: } might not be used.

  This may not be accurate due to some parameters are only used in 
language bindings but
  passed down to XGBoost core.  Or some parameters are not used but 
slip through this
  verification. Please open an issue if you find above cases.


[19:27:52] WARNING: /workspace/src/learner.cc:480: 
Parameters: { reg_alpha: } might not be used.

  This may not be accurate due to some parameters are only used in
```

In [29]:
```
grid_search.best_params_
```

Out[29]:
```
{'learning_rate': 0.1,
 'max_depth': 3,
 'min_child_weight': 1,
 'n_estimators': 11,
 'reg_alpha:': 0,
 'reg_lambda': 0,
 'subsample': 1}
```

In [42]:
```
grid_acc, grid_prec, grid_sens = my_classifier_results(grid_search)
```

Out[42]:

	pred:user	pred: not user
true:user	140	57
true:not user	86	117

```
Accuracy: 0.6425
Precision: 0.6195
Sensitivity: 0.5764
```

In [31]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

params = {
    "n_estimators": [5,10, 11],
    "max_depth": randint(1, 10),
    "min_child_weight": randint(1, 20),
    "learning_rate": uniform(0.001,1),
    "subsample": uniform(0.05, 1),
    "reg_lambda": uniform(0, 5),
    "reg_alpha:": uniform(0, 5)
}

random_search = RandomizedSearchCV(
    xgbr_model,
    param_distributions=params,
    random_state=8675309,
    n_iter=25,
    cv=5,
    verbose=0,
    n_jobs=1,
    return_train_score=True)

random_search.fit(X_train, y_train)
```

Out[31]:
```
[20:10:24] WARNING: /workspace/src/learner.cc:480:
Parameters: { reg_alpha: } might not be used.

  This may not be accurate due to some parameters are only used in
language bindings but
  passed down to XGBoost core.  Or some parameters are not used but
slip through this
  verification. Please open an issue if you find above cases.


[20:10:29] WARNING: /workspace/src/learner.cc:480:
Parameters: { reg_alpha: } might not be used.

  This may not be accurate due to some parameters are only used in
language bindings but
  passed down to XGBoost core.  Or some parameters are not used but
slip through this
  verification. Please open an issue if you find above cases.


[20:10:32] WARNING: /workspace/src/learner.cc:480:
Parameters: { reg_alpha: } might not be used.

  This may not be accurate due to some parameters are only used in
```

In [32]:
```
random_search.best_params_
```

Out[32]:
```
{'learning_rate': 0.27010055857700055,
 'max_depth': 4,
 'min_child_weight': 19,
 'n_estimators': 10,
 'reg_alpha:': 1.1726941967250077,
 'reg_lambda': 3.9259862132708676,
 'subsample': 0.9342635558121027}
```

In [43]:
```
rand_acc, rand_prec, rand_sens = my_classifier_results(random_search)
```

Out[43]:

	pred:user	pred: not user
true:user	135	62
true:not user	76	127

```
Accuracy: 0.6550
Precision: 0.6398
Sensitivity: 0.6256
```

In [34]:
```
np.random.seed(8675309)  # seed courtesy of Tommy Tutone
# from GPyOpt.methods import BayesianOptimization
# from sklearn.model_selection import cross_val_score, KFold

hp_bounds = [{
    'name': 'n_estimators',
    'type': 'discrete',
    'domain': (50, 150)
}, {
    'name': 'max_depth',
    'type': 'discrete',
    'domain': (1, 10)
}, {
    'name': 'min_child_weight',
    'type': 'discrete',
    'domain': (1, 20)
}, {
    'name': 'learning_rate',
    'type': 'continuous',
    'domain': (0.001, 1.0)
}, {
    'name': 'sub_sample',
    'type': 'continuous',
    'domain': (0.05, 1)
}, {
    'name': 'reg_lambda',
    'type': 'continuous',
    'domain': (0, 5)
}, {
    'name': 'reg_alpha',
    'type': 'continuous',
    'domain': (0, 5)
}]
# Optimization objective
def cv_score(hyp_parameters):
    hyp_parameters = hyp_parameters[0]
    xgb_model = xgb.XGBClassifier(objective='reg:squarederror',
                                 n_estimators=hyp_parameters[0],
                                 max_depth=int(hyp_parameters[1]),

min_child_weight=int(hyp_parameters[2]),
                                 learning_rate=hyp_parameters[3],
                                 subsample=int(hyp_parameters[4]),
                                 reg_lambda=hyp_parameters[5],
                                 reg_alpha=hyp_parameters[6])
    scores = cross_val_score(xgb_model,
                             X=X_train,
                             y=y_train,
                             cv=KFold(n_splits=5))
    return np.array(scores.mean())  # return average of 5-fold scores


optimizer = BayesianOptimization(f=cv_score,
                                 domain=hp_bounds,
                                 model_type='GP',
                                 acquisition_type='EI',
                                 acquisition_jitter=0.05,
                                 exact_feval=True,
                                 maximize=True,
                                 verbosity=True)

optimizer.run_optimization(max_iter=20,verbosity=True)
```

Out[34]:
```
num acquisition: 1, time elapsed: 23.40s
num acquisition: 2, time elapsed: 47.06s
num acquisition: 3, time elapsed: 71.23s
num acquisition: 4, time elapsed: 99.31s
num acquisition: 5, time elapsed: 126.30s
num acquisition: 6, time elapsed: 154.12s
num acquisition: 7, time elapsed: 182.36s
num acquisition: 8, time elapsed: 208.51s
num acquisition: 9, time elapsed: 235.93s
num acquisition: 10, time elapsed: 261.64s
num acquisition: 11, time elapsed: 286.61s
num acquisition: 12, time elapsed: 314.71s
num acquisition: 13, time elapsed: 347.53s
num acquisition: 14, time elapsed: 384.76s
num acquisition: 15, time elapsed: 421.30s
num acquisition: 16, time elapsed: 456.46s
num acquisition: 17, time elapsed: 491.91s
num acquisition: 18, time elapsed: 530.75s
num acquisition: 19, time elapsed: 568.40s
num acquisition: 20, time elapsed: 612.71s
```

In [35]:
```python
best_hyp_set = {}
for i in range(len(hp_bounds)):
    if hp_bounds[i]['type'] == 'continuous':
        best_hyp_set[hp_bounds[i]['name']] = optimizer.x_opt[i]
    else:
        best_hyp_set[hp_bounds[i]['name']] = int(optimizer.x_opt[i])
best_hyp_set
```

Out[35]:
```
{'n_estimators': 150,
 'max_depth': 1,
 'min_child_weight': 1,
 'learning_rate': 0.35488033434114763,
 'sub_sample': 0.5687129957781784,
 'reg_lambda': 3.950325217852195,
 'reg_alpha': 0.2536401881692735}
```

In [36]:
```python
bayopt_search =
xgb.XGBClassifier(objective='reg:squarederror',**best_hyp_set)
bayopt_search.fit(X_train,y_train)
```

Out[36]:
```
[20:35:00] WARNING: /workspace/src/learner.cc:480:
Parameters: { sub_sample } might not be used.

  This may not be accurate due to some parameters are only used in
language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip
through this
  verification. Please open an issue if you find above cases.



XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.35488033434114763, max_delta_step=0,
max_depth=1,
              min_child_weight=1, missing=nan,
monotone_constraints='()',
              n_estimators=150, n_jobs=0, num_parallel_tree=1,
              objective='reg:squarederror', random_state=0,
              reg_alpha=0.2536401881692735,
reg_lambda=3.950325217852195,
              scale_pos_weight=1, sub_sample=0.5687129957781784,
subsample=1,
              tree_method='exact', validate_parameters=1,
verbosity=None)
```

In [44]:
```
bay_acc, bay_prec, bay_sense = my_classifier_results(bayopt_search)
```

Out[44]:

	pred:user	pred: not user
true:user	127	70
true:not user	70	133

```
Accuracy: 0.6500
Precision: 0.6447
Sensitivity: 0.6552
```

In [38]:
```
# from tpot import TPOTRegressor

tpot_config = {
    'xgboost.XGBClassifier': {
        'n_estimators': [50,100,150],
        'max_depth': range(1, 10),
        'learning_rate': [1e-3, 1e-2, 1e-1, 0.5, 1.],
        'subsample': np.arange(0.05, 1.01, 0.05),
        'min_child_weight': range(1, 21),
        'reg_alpha': range(1, 6),
        'reg_lambda': range(1, 6),
        'nthread': [1],
        'objective': ['binary:logistic']
    }
}

tpot = TPOTRegressor(scoring = 'r2',
                     generations=5,
                     population_size=20,
                     verbosity=2,
                     config_dict=tpot_config,
                        cv=5,
                        random_state=8675309)
tpot.fit(X_train, y_train)
tpot.export('tpot_XGBClassifier.py') # export the model
```

Out[38]:

```
Generation 1 - Current best internal CV score: -0.37084846902878205
Generation 2 - Current best internal CV score: -0.37084846902878205
Generation 3 - Current best internal CV score: -0.37084846902878205
Generation 4 - Current best internal CV score: -0.36534497157926876
Generation 5 - Current best internal CV score: -0.36534497157926876
Best pipeline: XGBClassifier(input_matrix, learning_rate=0.1,
max_depth=4, min_child_weight=7, n_estimators=50, nthread=1,
objective=binary:logistic, reg_alpha=2, reg_lambda=5, subsample=0.1)
```

In [39]:

```
my_classifier_results(tpot)
```

Out[39]:

	pred:user	**pred: not user**
true:user	124	73
true:not user	66	137

```
Accuracy: -0.3903
Precision: 0.6526
Sensitivity: 0.6749
```

```
(-0.3903128203845867, 0.6526315789473685, 0.6748768472906403)
```

In [40]:

```
# from tpot import TPOTRegressor

tpot = TPOTRegressor(scoring = 'r2',
                        generations=10,
                        population_size=40,
                        verbosity=2,
                        cv=5,
                        random_state=8675309)
tpot.fit(X_train, y_train)
print(tpot.score(X_test, y_test))
tpot.export('tpot_optimal_pipeline.py')
```

Out[40]:

```
Generation 1 - Current best internal CV score: 0.15893273789162218
Generation 2 - Current best internal CV score: 0.15893273789162218
Generation 3 - Current best internal CV score: 0.15893273789162218
Generation 4 - Current best internal CV score: 0.15893273789162218
Generation 5 - Current best internal CV score: 0.15893273789162218
Generation 6 - Current best internal CV score: 0.15893273789162218
Generation 7 - Current best internal CV score: 0.15893273789162218
Generation 8 - Current best internal CV score: 0.15909261621229195
Generation 9 - Current best internal CV score: 0.1591057663867082
Generation 10 - Current best internal CV score: 0.1591057663867082
Best pipeline:
ElasticNetCV(VarianceThreshold(RobustScaler(input_matrix),
threshold=0.05), l1_ratio=0.5, tol=1e-05)
0.13514677595189883
```

P2.2e - Summary

- In addition to your summary table, answer:

- The bank isn't as concerned about misclassifying some truly good loans as they are interested in correctly predicting truly bad loans. Which model should they use? Why?

*** 5 points - (don't delete this cell)

I think the TPOT is best here, in terms of accuracy. I think my function didn't like something about the information it was fed from TPOT so the accuracy looks lower than I think it should be. However, this also took a very long time to run and the Bayesian model returned better precision and sensitivity than TPOT and a somewhat high accuracy. If I take my table for face value then I think we're best off using the Bayesian Optimizaiton here. It took the least amount of time to run and returned similar results to the Grid Search and Randomized Search methods. It also has the highest accuracy as reported from the my_classifier_results function and since the domain of the problem is to correctly pick out bad loans, I think the Bayesian Optimization did the best here.

In [53]:
```
import pandas as pd
prob1 = [['Grid SearchCV', grid_search.best_params_['n_estimators'],
grid_search.best_params_['max_depth'],
grid_search.best_params_['min_child_weight'],
grid_search.best_params_['learning_rate'],
grid_search.best_params_['subsample'],
          grid_search.best_params_['reg_lambda'], grid_acc, grid_prec,
grid_sens],
         ['Randomized SearchCV',
random_search.best_params_['n_estimators'],
random_search.best_params_['max_depth'],
random_search.best_params_['min_child_weight'],
random_search.best_params_['learning_rate'],
random_search.best_params_['subsample'],
          random_search.best_params_['reg_lambda'],  rand_acc,
rand_prec, rand_sens],
         ['Bayesian', best_hyp_set['n_estimators'],
best_hyp_set['max_depth'], best_hyp_set['min_child_weight'],
best_hyp_set['learning_rate'], best_hyp_set['sub_sample'],
best_hyp_set['reg_lambda'],
          bay_acc, bay_prec, bay_sense],
         ['TPOT', 50, 4, 7, 0.1, 0.1, 5, 0.3903, 0.6526, 0.6749]]
df = pd.DataFrame(prob1, columns = ['model','n_estimators',
'max_depth', 'min_child_weight', 'learning_rate', 'subsample',
'reg_lambda', 'accuracy', 'precision', 'sensitivity'])
df
```

Out[53]:

	model	n_estimators	max_depth	min_child_weight	learning_rate	subsample	reg
0	Grid SearchCV	11	3	1	0.100000	1.000000	0.0
1	Randomized SearchCV	10	4	19	0.270101	0.934264	3.9
2	Bayesian	150	1	1	0.354880	0.568713	3.9
3	TPOT	50	4	7	0.100000	0.100000	5.0

P2.3 - Extra Credit - Problem 3 (up to 10 points)

Show how to use the `pycaret` package to do model selection for one of the two problems above. We've never used `pycaret` but it looks promising. We have used the `caret` package in R and it simplifies many machine learning tasks considerably.

Use Google to search for `pycaret` to get started.

In [61]:
```
from pycaret.classification import *
from pycaret.datasets import get_data

diabetes = get_data('diabetes')
test = setup(diabetes, target='Class variable')
```

Out[61]:
```
Setup Succesfully Completed!
```

	Description	Value
0	session_id	7467
1	Target Type	Binary
2	Label Encoded	0: 0, 1: 1
3	Original Data	(768, 9)
4	Missing Values	False
5	Numeric Features	7
6	Categorical Features	1
7	Ordinal Features	False
8	High Cardinality Features	False
9	High Cardinality Method	None
10	Sampled Data	(768, 9)
11	Transformed Train Set	(537, 24)
12	Transformed Test Set	(231, 24)
13	Numeric Imputer	mean
14	Categorical Imputer	constant
15	Normalize	False

In [62]:
```
compare_models()
```

Out[62]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
0	Linear Discriminant Analysis	0.7749	0.8326	0.5784	0.7488	0.6432	0.4835	0.4986	0.0288
1	Ridge Classifier	0.7731	0.0000	0.5734	0.7462	0.6405	0.4792	0.4936	0.0361
2	CatBoost Classifier	0.7673	0.8389	0.5784	0.7082	0.6297	0.4642	0.4740	4.2729
3	Logistic Regression	0.7657	0.8184	0.5678	0.7157	0.6298	0.4617	0.4708	0.2913
4	Light Gradient Boosting Machine	0.7597	0.8232	0.6099	0.6855	0.6368	0.4600	0.4679	0.0919
5	Extra Trees Classifier	0.7543	0.7934	0.5249	0.6933	0.5923	0.4239	0.4344	0.1411
6	Gradient Boosting Classifier	0.7542	0.8192	0.5728	0.6808	0.6150	0.4381	0.4460	0.1631
7	Extreme Gradient Boosting	0.7522	0.8095	0.6050	0.6723	0.6264	0.4439	0.4530	0.1060
8	K Neighbors Classifier	0.7468	0.7661	0.5892	0.6521	0.6160	0.4289	0.4322	0.0171
9	Ada Boost Classifier	0.7468	0.8095	0.5994	0.6553	0.6219	0.4329	0.4371	0.0981
10	Random Forest Classifier	0.7319	0.7873	0.4550	0.6850	0.5390	0.3621	0.3812	0.0285
11	Decision Tree Classifier	0.6965	0.6683	0.5737	0.5638	0.5638	0.3334	0.3363	0.0146
12	Naive Bayes	0.6874	0.7437	0.1988	0.7051	0.2973	0.1750	0.2354	0.0085
13	Quadratic Discriminant Analysis	0.5793	0.6140	0.4529	0.4788	0.3775	0.0993	0.1107	0.0039
14	SVM - Linear Kernel	0.5698	0.0000	0.6211	0.4176	0.4728	0.1484	0.1738	0.0060

```
LinearDiscriminantAnalysis(n_components=None, priors=None,
shrinkage=None,
                           solver='svd', store_covariance=False,
tol=0.0001)
```

In [63]:
```
lda = create_model('lda')
```

Out[63]:

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.6852	0.7398	0.5789	0.5500	0.5641	0.3180	0.3182
1	0.8148	0.8887	0.5263	0.9091	0.6667	0.5507	0.5902
2	0.7593	0.7805	0.4211	0.8000	0.5517	0.4081	0.4474
3	0.6481	0.7218	0.6316	0.5000	0.5581	0.2723	0.2775
4	0.8333	0.8692	0.6842	0.8125	0.7429	0.6209	0.6259
5	0.7407	0.8496	0.5263	0.6667	0.5882	0.4028	0.4088
6	0.7963	0.8767	0.5263	0.8333	0.6452	0.5123	0.5389
7	0.8302	0.8714	0.6667	0.8000	0.7273	0.6055	0.6108
8	0.7736	0.8397	0.5000	0.7500	0.6000	0.4508	0.4688
9	0.8679	0.8889	0.7222	0.8667	0.7879	0.6931	0.6992
Mean	0.7749	0.8326	0.5784	0.7488	0.6432	0.4835	0.4986
SD	0.0653	0.0592	0.0903	0.1281	0.0808	0.1297	0.1312

In [64]:
```
plot_model(lda)
```

Out[64]: Image in a Jupyter notebook

In [66]:
```
predictions = predict_model(lda)
```

Out[66]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Linear Discriminant Analysis	0.7273	0.8064	0.5556	0.625	0.5882	0.3854	0.3869

In [67]:
```
finalize_model(lda)
```

Out[67]:
```
INFO:logs:Initializing create_model()
INFO:logs:create_model(estimator=LinearDiscriminantAnalysis(n_components
=None, priors=None, shrinkage=None,
                           solver='svd', store_covariance=False,
tol=0.0001), ensemble=False, method=None, fold=10, round=4,
cross_validation=True, verbose=False, system=False)
INFO:logs:Checking exceptions
INFO:logs:Preloading libraries
INFO:logs:Preparing display monitor
INFO:logs:Copying training dataset
INFO:logs:Importing libraries
INFO:logs:Defining folds
INFO:logs:Declaring metric variables
INFO:logs:Importing untrained model
INFO:logs:Declaring custom model
INFO:logs:Linear Discriminant Analysis Imported succesfully
INFO:logs:Checking ensemble method
INFO:logs:Initializing Fold 1
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 2
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 3
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 4
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 5
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 6
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 7
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 8
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 9
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Initializing Fold 10
INFO:logs:Fitting Model
INFO:logs:Evaluating Metrics
INFO:logs:Compiling Metrics
INFO:logs:Calculating mean and std
INFO:logs:Creating metrics dataframe
INFO:logs:Finalizing model
INFO:logs:Uploading results into container
```

```
INFO:logs:Uploading model into container now
INFO:logs:create_model_container: 3
INFO:logs:master_model_container: 3
INFO:logs:display_container: 5
INFO:logs:LinearDiscriminantAnalysis(n_components=None, priors=None,
shrinkage=None,
                           solver='svd', store_covariance=False,
tol=0.0001)
INFO:logs:create_model() succesfully
completed......................................
INFO:logs:create_model_container: 3
INFO:logs:master_model_container: 3
INFO:logs:display_container: 5
INFO:logs:LinearDiscriminantAnalysis(n_components=None, priors=None,
shrinkage=None,
                           solver='svd', store_covariance=False,
tol=0.0001)
INFO:logs:finalize_model() succesfully
completed......................................
```

```
LinearDiscriminantAnalysis(n_components=None, priors=None,
shrinkage=None,
                           solver='svd', store_covariance=False,
tol=0.0001)
```

In [68]:

```
save_model(lda, 'diabetes_lda')
```

Out[68]:
```
INFO:logs:Initializing save_model()
INFO:logs:save_model(model=LinearDiscriminantAnalysis(n_components=None,
priors=None, shrinkage=None,
                           solver='svd', store_covariance=False,
tol=0.0001), model_name=diabetes_lda, model_only=False, verbose=True)
INFO:logs:Adding model into prep_pipe
INFO:logs:diabetes_lda.pkl saved in current working directory
INFO:logs:Pipeline(memory=None,
         steps=[('dtypes',
                 DataTypes_Auto_infer(categorical_features=[],
                                      display_types=True,
features_todrop=[],
                                      ml_usecase='classification',
                                      numerical_features=[],
                                      target='Class variable',
                                      time_features=[])),
                ('imputer',
                 Simple_Imputer(categorical_strategy='not_available',
                                numeric_strategy='mean',
                                target_variable=None)),
                ('new_levels1',
                 New_Catagori...
                ('cluster_all', Empty()),
                ('dummy', Dummify(target='Class variable')),
                ('fix_perfect', Empty()), ('clean_names',
Clean_Colum_Names()),
                ('feature_select', Empty()), ('fix_multi', Empty()),
                ('dfs', Empty()), ('pca', Empty()),
                ['trained model',
                 LinearDiscriminantAnalysis(n_components=None,
priors=None,
                                            shrinkage=None,
solver='svd',
                                            store_covariance=False,
                                            tol=0.0001)]],
         verbose=False)
INFO:logs:save_model() succesfully
completed......................................
Transformation Pipeline and Model Succesfully Saved
```

In [0]: