Introduction

Diabetes is one of the most common and hazardous diseases on the planet. It requires a lot of care and proper medication to keep the disease in control. In this data mining project, this project teaches us to develop a classification system to detect whether the patient has diabetes or not. As part of this project, I will practice the usage of the classifiers to predict diabetes for patients of a given dataset.

I- Firstly, we choose the four classifiers and compare between them and to use the cross validation. Moreover, we try to increase the scores and have innovation in the experiment to improve these scores.

- ✓ Neural Networks
- ✓ Decision Trees
- ✓ Support Vector machine
- ✓ K-Nearest neighbours

II- Secondly, we Execute four mentioned classifiers, compare them and use the cross validation by changing the CV of GoridSearchCV', 'RandomizedSearchCV' and 'HalvingGridSearchCV' in each classifier to do more analysis and improve their score.

As the target value of the diabetes dataset is binary and is not continues, we use the below algorithms for prediction:

- ❖ MLPClassifier()
- ❖ DecisionTreeClassifier()
- ❖ SVC()
- ❖ KneighborsClassifier()

1- Neural Networks (MLPClassifier):

I- Gridsearchcv

```
tuned_parameters = {'hidden_layer_sizes':[(5,7), (20,22), (50,65), (100,110),(150,160)]
                    ,'alpha':[0.0001,0.001, 0.01, 1, 2,4],'max_iter':[300,350,400,500],
                   'activation':['relu','logestic', 'identity','tanh']   }
clf=GridSearchCV(MLPClassifier(solver='lbfgs',tol=5e-3,random_state=1234),
                 tuned_parameters,refit=True, cv=5)
clf.fit(X_train, y_train)

print(f" the best estimator of model is:  {clf.best_estimator_}")
print(f" the best score of model is:  {clf.best_score_}")
print(f" the best parameter of model is:  {clf.best_params_}")
print(f" max of mean_test_score is:   {max(clf.cv_results_['mean_test_score'])}")

```

The score of this algorithm before using grid search was 0.68 and the score of the first grid search by using tuned parameters obtain 0.77, so we use iteration (loop by for) in grid search to change CV (cross validation) in order to get the better score as below:

```
tuned_parameters = {'hidden_layer_sizes':[(5,7), (20,22), (50,65), (100,110),(150,160)],
                    'alpha':[0.0001,0.001, 0.01, 1, 2,4],'max_iter':[300,350,400,500],
                   'activation':['relu','logestic', 'identity','tanh']   }
Gscore=[GridSearchCV(MLPClassifier(solver='lbfgs',tol=5e-3,random_state=1234),
        tuned_parameters,refit=True, cv=n).fit(X_train, y_train).best_score_
        for n in range(2,10)]

print(f" max gscore is: {max(Gscore)}")
```

In this code we change CV from 2 to 10 and obtain different scores as below:

[0.804, 0.804, 0.813, 0.804, 0.796, 0.804, 0.792, 0.8177]

Which max of them is 0.81.

And this process repeats for random grid search as below:

II- Rrandomgridsearchcv

```
#RandomizedSearchCV
mlp=MLPClassifier(solver='lbfgs',tol=5e-3,random_state=1234)
tuned_parameters = {'hidden_layer_sizes':[(5,7), (20,22), (50,65), (100,110),(150,160)],
                    'alpha':[0.0001,0.001, 0.01, 1, 2,4],'max_iter':[300,350,400,500],
                   'activation':['relu','logestic', 'identity','tanh']   }
Rscore=[RandomizedSearchCV(mlp, tuned_parameters, n_jobs=-1, random_state=123,
      refit=True, n_iter=100,cv=n).fit(X_train, y_train).best_score_ for n in range(2,10)]

# train the random search meta-estimator to find the best model

```

max(Rscore)=0.77

III- HalvingGridSearchCV

```
#HalvingGridSearchCV
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV

mlp=MLPClassifier(solver='lbfgs',tol=5e-3,random_state=1234)
tuned_parameters = {'hidden_layer_sizes':[(5,7), (20,22), (50,65), (100,110),(150,160)]
          ,'alpha':[0.0001,0.001, 0.01, 1, 2,4],'max_iter':[300,350,400,500],
          'activation':['relu', 'identity','tanh']   }

Hscore=[HalvingGridSearchCV(mlp, tuned_parameters, n_jobs=-1, random_state=541,
   refit=True,v=n).fit(X_train, y_train).best_score_ for n in range(2,10)]

```

max(Hscore)= 0.80

the best score obtain by grid search cv with score of 0.81.

As Gridsearchcv in MLPClassifier by changing hyper parameters which we use and loops 'for' was really time consuming and each one long 12houre, I could not draw a plot like next classifier.

2. Decision Trees (DecisionTreeClassifier)

```
clf = tree.DecisionTreeClassifier(random_state=42)
par = {'criterion':['gini','entropy'],
'max_depth':list(range(1,100,5)),
'min_samples_leaf':list(range(1,100,5)),
 'class_weight':['balanced',{0:0.3,1:0.7},{0:0.4,1:0.6},{0:0.2,1:0.8},
                 {0:0.5,1:0.5}]      }
GS = GridSearchCV(clf, param_grid=par, cv=4)
GS.fit(X_train, y_train)
print(f" the best estimator of model is:  {GS.best_estimator_}")
print(f" the best score of model is:  {GS.best_score_}")
print(f" the best parameter of model is:  {GS.best_params_}")
print(f" max of mean_test_score is:   {max(GS.cv_results_['mean_test_score'])}")


```

```
the best estimator of model is:  DecisionTreeClassifier(class_weight={0: 0.5, 1: 0.5}, criterion='entropy',
                     max_depth=7, min_samples_leaf=61, random_state=42)
the best score of model is:  0.7508169934640523
the best parameter of model is:  {'class_weight': {0: 0.5, 1: 0.5}, 'criterion': 'entropy', 'max_depth': 7,
max of mean_test_score is:   0.7508169934640523
```

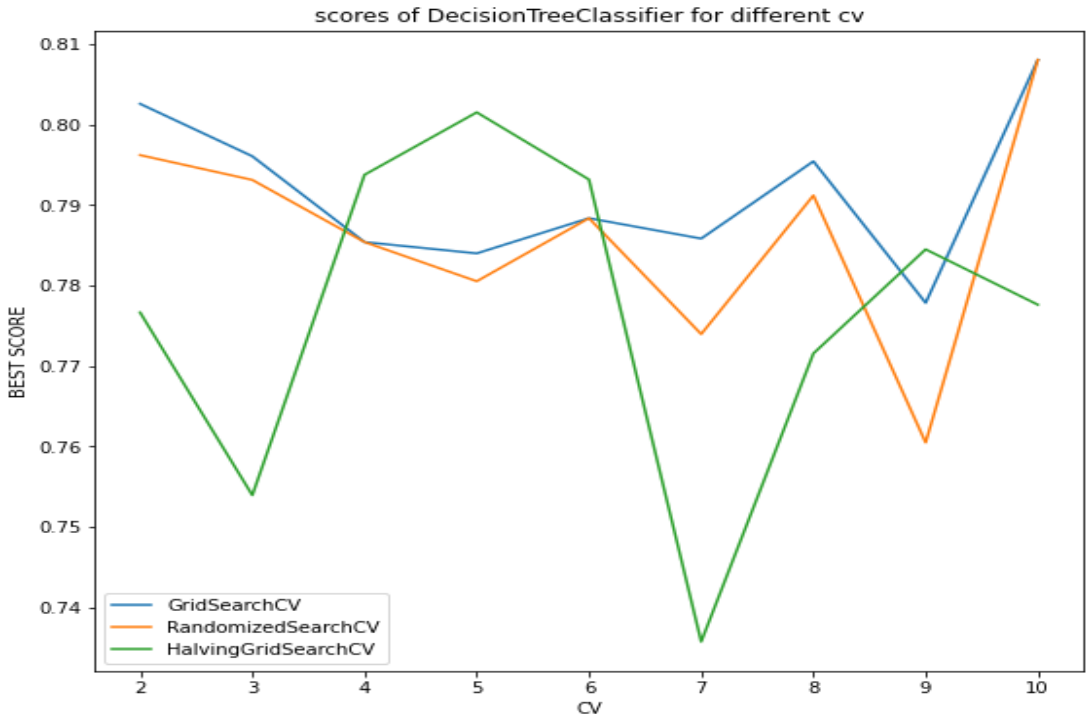
The score of this algorithm before using grid search was 0.74 and the score of the first grid search by using tuned parameters obtain 0.75. Using max_depth and min_samples_split parameter in Decision Trees avoid from overfitting.

So I have innovation for increasing the score of unbalanced data by writing a function which calculate the score with recall_score.

```
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix

def weighted_mean_recall(y, yhat):
    r0 = recall_score(y, yhat, pos_label=0)
    r1 = recall_score(y, yhat, pos_label=1)
    return 0.25*r0 + 0.75*r1

from sklearn.metrics import make_scorer
my_scorer = make_scorer(weighted_mean_recall, greater_is_better=True)

```

Now we use this my_score function for scoring and iteration(loop by for)) in grid search to change CV (cross validation) in order to improve score. In this algorithms we run three grid search together and draw plot as below:

```
clf = tree.DecisionTreeClassifier(random_state=42)
par = {'criterion':['gini','entropy'],
      'max_depth':list(range(1,100,5)),
      'min_samples_leaf':list(range(1,100,5)),'class_weight':['balanced',{0:0.3,1:0.7},
                          {0:0.4,1:0.6},{0:0.2,1:0.8},{0:0.5,1:0.5}]
       }
Gscores=[GridSearchCV(clf, param_grid=par, cv=n,scoring=my_scorer).fit(X_train, y_train)
.best_score_ for n in range(2,11)]
Rscores=[RandomizedSearchCV(clf, param_distributions=par,n_iter=100, cv=n,scoring=my_scorer)
.fit(X_train, y_train).best_score_ for n in range(2,11)]
Hscores=[HalvingGridSearchCV(clf, param_grid=par, cv=n,scoring=my_scorer).fit(X_train, y_train)
.best_score_ for n in range(2,11)]
fig = plt.figure(figsize=(20, 8))
fig.add_subplot(122)
plt.plot(range(2,11),Gscores,label='GridSearchCV')
plt.plot(range(2,11),Rscores,label='RandomizedSearchCV')
plt.plot(range(2,11),Hscores,label='HalvingGridSearchCV')
plt.title("scores of DecisionTreeClassifier for different cv")
plt.xlabel("CV")
plt.ylabel("BEST SCORE")
plt.legend()
```



As can be seen in the above plot grid search cv and random search cv give us the more score than having which is near **81** with **CV=10**.

3- SVM (SVC):

As SVC use different algorithms at first we test different kernel after that by the best kernel use loops 'for' in the gridsearchcv to change CV.

- **Rbf kernel**

```
clf=SVC()
par = {'gamma':list(np.arange(0.0,1,0.001)),'class_weight':['balanced',{0:0.3,1:0.7}
                                ,{0:0.4,1:0.6},{0:0.2,1:0.8},{0:0.5,1:0.5}]}

GS = GridSearchCV(clf, param_grid=par, cv=4,scoring=my_scorer)
GS.fit(X_train, y_train)
print(f" the best estimator of model is:  {GS.best_estimator_}")

print(f" the best score of model is:  {GS.best_score_}")
```

```
the best estimator of model is:  SVC(class_weight={0: 0.2, 1: 0.8}, gamma=0.001)
the best score of model is:  0.828890931372549
```

The score of this algorithm before using grid search cv was 0.63. we use my_score function for scoring and grid search to improve the score. the default kernel in svc is ‘rbf ‘.

So the above code use ‘rbf’ kernel and obtain the score 0.82.

- **Poly kernel**

```
clf=SVC(kernel='poly',degree=2)
par = {'gamma':[0.001,0.1,1],'class_weight':['balanced',{0:0.3,1:0.7},{0:0.4,1:0.6}
                                        ,{0:0.2,1:0.8},{0:0.5,1:0.5}]}

GS = GridSearchCV(clf, param_grid=par, cv=4,scoring=my_scorer)
GS.fit(X_train, y_train)
print(f" the best estimator of model is:  {GS.best_estimator_}")

print(f" the best score of model is:  {GS.best_score_}")
```

```
the best estimator of model is:  SVC(class_weight={0: 0.2, 1: 0.8}, degree=2, gamma=0.001, kernel='poly
the best score of model is:  0.7119601889338731
```

the score of the grid search cv by poly kernel and my_score function obtain 0.71.

- **Linear kernel**

```
clf=SVC(kernel='linear')
par = {'C':list(range(1,100,5)),'class_weight':['balanced',{0:0.3,1:0.7},
                          {0:0.4,1:0.6},{0:0.2,1:0.8},{0:0.5,1:0.5}]}

GS = GridSearchCV(clf, param_grid=par, cv=4,scoring=my_scorer)
GS.fit(X_train, y_train)
print(f" the best estimator of model is:  {GS.best_estimator_}")

print(f" the best score of model is:  {GS.best_score_}")
```

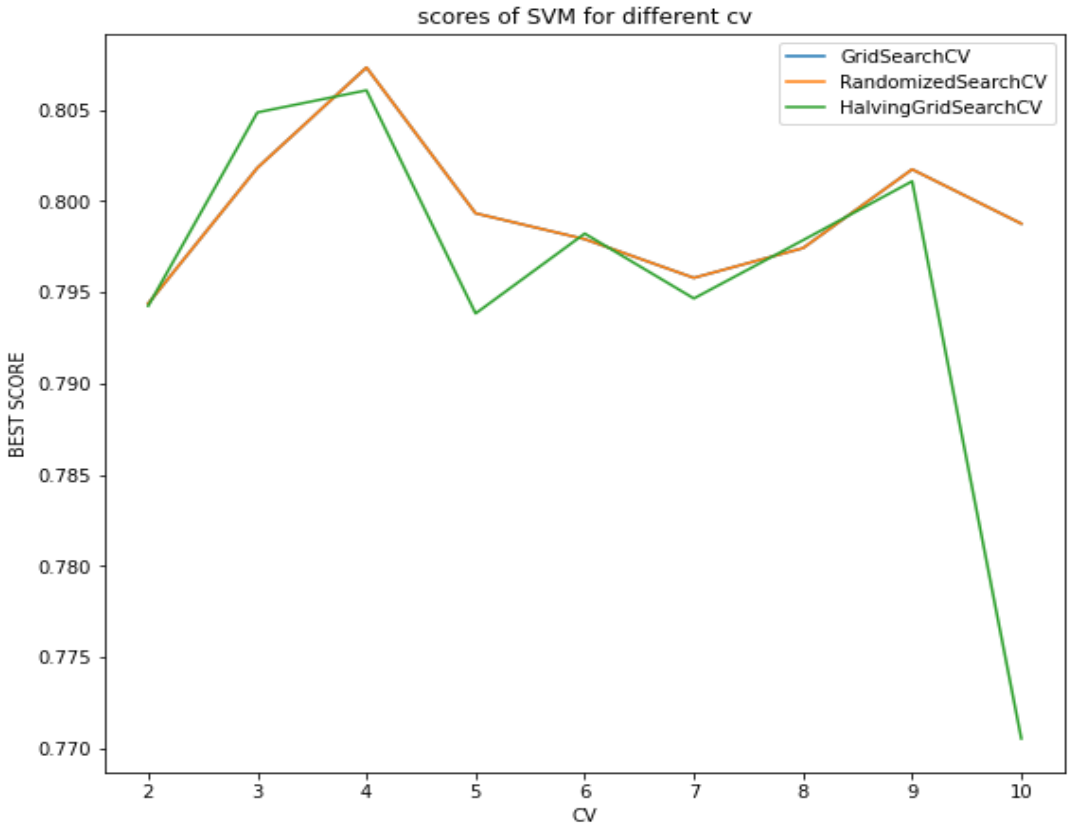
```
the best estimator of model is:  SVC(C=81, class_weight={0: 0.2, 1: 0.8}, kernel='linear')
the best score of model is:  0.7874915654520918
```

the score of the grid search cv by linear kernel and my_score function obtain 0.78.

- **Using loops 'for' in different grid search cv**

As 'rbf' kernel obtained the best score of 0.82, we use 'rbf' kernel in loops in different grid search cv.

```
svm=SVC(kernel='rbf')
par = {'gamma':list((0,0.001,0.01)),'class_weight':['balanced',{0:0.3,1:0.7}
                                  ,{0:0.2,1:0.8}]}
Gscores=[GridSearchCV(svm, param_grid=par, cv=n,scoring=my_scorer)
.fit(X_train, y_train).best_score_ for n in range(2,11)]
Rscores=[RandomizedSearchCV(svm, param_distributions=par,n_iter=100,
  cv=n,scoring=my_scorer).fit(X_train, y_train).best_score_ for n in range(2,11)]
Hscores=[HalvingGridSearchCV(svm, param_grid=par, cv=n,scoring=my_scorer).
        fit(X_train, y_train).best_score_ for n in range(2,11)]
fig = plt.figure(figsize=(20, 8))
fig.add_subplot(122)
plt.plot(range(2,11),Gscores,label='GridSearchCV')
plt.plot(range(2,11),Rscores,label='RandomizedSearchCV')
plt.plot(range(2,11),Hscores,label='HalvingGridSearchCV')
plt.title("scores of SVM for different cv")
plt.xlabel("CV")
plt.ylabel("BEST SCORE")
plt.legend()

```



Gscores is [0.794, 0.801, 0.807, 0.799, 0.797, 0.7957913216220273, 0.797, 0.8017382154882156, 0.798] max of it is: 0.807
Rscores is [0.794, 0.801, 0.807, 0.799, 0.797, 0.795, 0.797, 0.801, 0.798]
max of it is: 0.807
Hscores is [0.794, 0.804, 0.806, 0.793, 0.798, 0.794, 0.797, 0.801, 0.770]
max of it is: 0.8060711806163289

As can be seen in the above plot grid search cv and random search cv give us the same score and in the plot coincide together and the more score than having which is near 81 with CV=10.

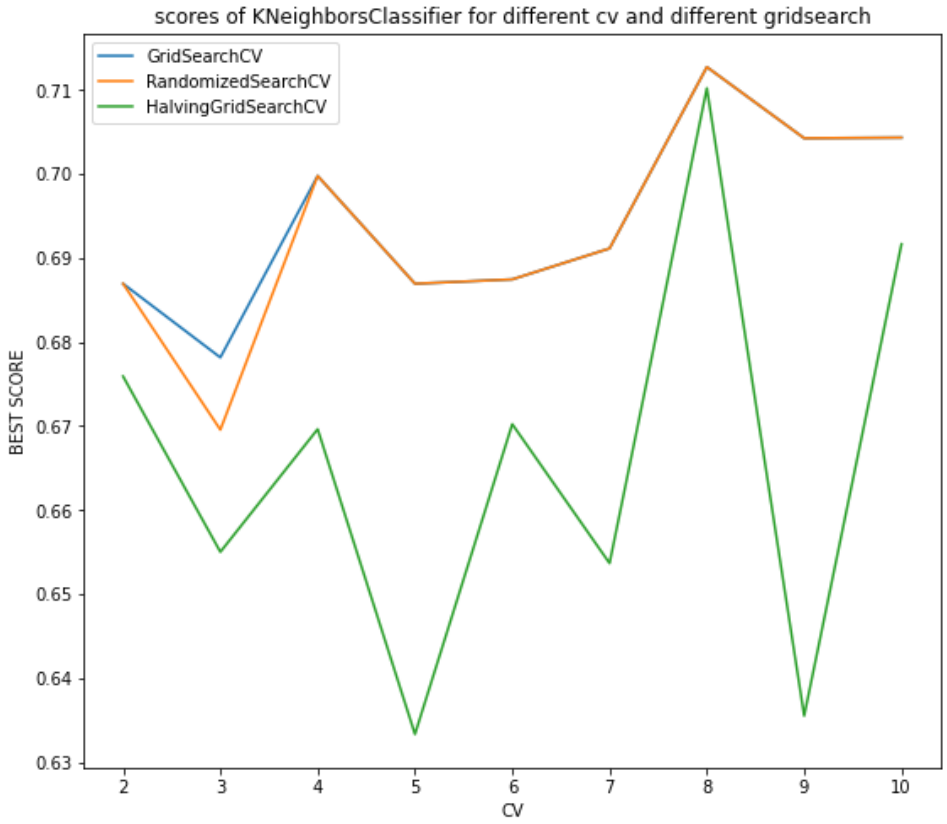
4- K-Nearest neighbours (KNeighborsClassifier):

```
from sklearn.metrics import make_scorer
my_scorer = make_scorer(weighted_mean_recall, greater_is_better=True)

par = {'weights':['uniform', 'distance'], 'n_neighbors':list(np.arange(1,15)) }
knn=KNeighborsClassifier(n_jobs=3)
GS = GridSearchCV(knn, param_grid=par, cv=5)
GS.fit(X_train, y_train)
print(f" the best estimator of model is:  {GS.best_estimator_}")

print(f" the best score of model is:  {GS.best_score_}")

```

```
the best estimator of model is:  KNeighborsClassifier(n_jobs=3, n_neighbors=8, weights='distance')
the best score of model is:  0.6869565217391305
```

the score of the first grid search by using tuned parameters obtain 0.68, so we use iteration (loop by for) in grid search to change CV (cross validation) in order to get the better score as below:

```
knn=KNeighborsClassifier(n_jobs=3)
par = {'weights':['uniform', 'distance'], 'n_neighbors':list(np.arange(1,30)) }

Gscores=[GridSearchCV(knn, param_grid=par, cv=n,).fit(X_train, y_train).best_score_
         for n in range(2,11)]
Rscores=[RandomizedSearchCV(knn, param_distributions=par,n_iter=50, cv=n).fit(X_train, y_train).best_score_
         for n in range(2,11)]
Hscores=[HalvingGridSearchCV(knn, param_grid=par, cv=n).fit(X_train, y_train).best_score_
         for n in range(2,11)]
fig = plt.figure(figsize=(20, 8))
fig.add_subplot(122)
plt.plot(range(2,11),Gscores,label='GridSearchCV')
plt.plot(range(2,11),Rscores,label='RandomizedSearchCV')
plt.plot(range(2,11),Hscores,label='HalvingGridSearchCV')

plt.title("scores of KNeighborsClassifier for different cv and different gridsearch")
plt.xlabel("CV")
plt.ylabel("BEST SCORE")
plt.legend()
```



As can be seen in the above plot grid and random search cv give us the more score than halving which is near 72 with CV=8.

Comparing the score of each classifier

Classifier/Score	MLP Classifier/CV	DecisionTree Classifier /CV	SVC/CV	Kneighbors Classifier /CV
Gridsearchcv Score	0.81/cv=4	0.807/cv=10	0.81/cv=4	0.72/cv=8

Rrandomgridsearchcv Score	0.77/cv=5	0.807/cv=10	0.81/cv=4	0.72/cv=5
HalvingGridSearchCV Score	0.80/cv=4	0.804/cv=5	0.806/cv=4	0.71/cv=8

According the above table MLP Classifier and SVC obtained the best score of 0.81 by cv=4 and Gridsearchcv.

III-We set Pipeline to compare the performance of four classifiers

```
clf1 = MLPClassifier(solver='lbfgs',tol=5e-3,max_iter=500,random_state=1234)
clf2 =tree.DecisionTreeClassifier(random_state=42)
clf3=SVC(random_state=42)
clf4=KNeighborsClassifier(n_jobs=3)
# Initiaze the hyperparameters for each dictionary
param1 = {'classifier': [clf1],'classifier__hidden_layer_sizes':(1,200)
,'classifier__alpha':[0.001,0.01, 1, 2],
                      }
param2 = {'classifier': [clf2],
      'classifier__max_depth':list(range(1,30)),
      'classifier__min_samples_leaf':list(range(1,10)),'classifier__class_weight':
      ['balanced',{0:0.3,1:0.7},{0:0.4,1:0.6},{0:0.2,1:0.8},{0:0.5,1:0.5}]
       }
param3 = {'classifier': [clf3], 'classifier__C': [10**-2, 10**-1, 10**0, 10**1, 10**2]
,'classifier__class_weight': ['balanced', {0:1,1:5}, {0:1,1:10}, {0:1,1:25}]
}
param4 = {'classifier': [clf4],'classifier__weights':['uniform', 'distance'],
          'classifier__n_neighbors':list(np.arange(2,30))}
pipeline = Pipeline([('classifier', clf1)])
params = [param1, param2, param3, param4]
```

- **GridSearchCV**

```
# Train the grid search model
gs = GridSearchCV(pipeline, params, cv=5, n_jobs=-1, scoring='roc_auc').fit(X_train, y_train)
print(f" the best params of gridsearccv is: {gs.best_params_}")
print(f" the best score of gridsearccv is: {gs.best_score_}")
```

```
the best params of gridsearccv is: {'classifier': SVC(C=10, class_weight='balanced', random_state=42), 'classifier__C': 10, 'classifier__class_weight
the best score of gridsearccv is: 0.8047038831054139
```

- **RandomizedSearchCV**

```
# Train the random search model
rs = RandomizedSearchCV(pipeline, params, cv=5, n_jobs=-1, scoring='roc_auc').fit(X_train, y_train)
print(f" the best params of gridsearccv is: {rs.best_params_}")
print(f" the best score of gridsearccv is: {rs.best_score_}")
```

```
the best params of gridsearccv is: {'classifier__min_samples_leaf': 8, 'classifier__max_depth': 6, 'classifier__class_weight': {0: 0.3, 1:
                    min_samples_leaf=8, random_state=42)}
the best score of gridsearccv is: 0.7595738887253459
```

- **HalvingGridSearchCV**

```
# Train the halving search model
rs = HalvingGridSearchCV(pipeline, params, cv=5, n_jobs=-1, scoring='roc_auc').fit(X_train, y_train)
print(f" the best params of HalvingGridSearchCV is: {rs.best_params_}")
print(f" the best score of HalvingGridSearchCV is: {rs.best_score_}")
```

```
the best params of HalvingGridSearchCV is: {'classifier': SVC(C=1, class_weight='balanced', random_state=42), 'classifier__C': 1, 'classifier__class_
the best score of HalvingGridSearchCV is: 0.8146025835005662
```

Comparing of result of the Pipeline in different Gridsearch

Classifier/Score	Best Score	Best classifier	Best parameter
Gridsearchcv	0.804	SVC	{C=10, class_weight='balanced'}
Rrandomgridsearch cv	0.75	DecisionTree	{class_weight={0: 0.3, 1: 0.7}, max_depth=6, min_samples_leaf=8 }
HalvingGridSearchCV	0.81	SVC	{C=1, class_weight='balanced'}

IV- conclusion

According to result of part 2 which we run each classifier separately and use different CV(cross validation) and different grid search(grid search cv, random and halving) which CV change from 2 to 10, the best classifier for this Diabetes dataset are MLP Classifier and SVC obtained score of 0.81 by cv=4 and Gridsearchcv.

Also bases on result of above table in part 3 which extract from pipeline which use different CV(cross validation) by only use cv=5, the best classifier for this Diabetes dataset is Support Vector
machine (SVC), with score= 0.81 and parameter ={C=1, class_weight='balanced'} obtained from HalvingGridSerachCv.

Therefore, we conclude that the best score obtains by MLP Classifier and SVC which was 0.81.

We predict that there are additional opportunities to enhance the outcomes, such as testing more parameter options and values for each classifier, evaluating other classifiers in order to produce a better model.

Additionally, there are some problems with our prediction:

- For one, we only selected a subset of the parameters that are crucial to each classifier for, not all of them;
- secondly, we selected a limited range of values for each parameter to avoid an excessively long execution time.