finalprojectanswer

December 2, 2023

- **Students**: Parisa Khanjani
- **Professor**: Mohammed Ayoub Alaoui Mhamdi
- **University**: Bishop's University
- **Semester**: Fall 2023
- **Final Project**: Data Mining

1 Classifier Selection and Cross-Validation

1.1 My Approach

I wanted to find the best way to solve a problem using computers. So, I tried out four different methods to see which one works the best. I also wanted to make these methods better if we could.

1.2 The Four Methods

I used these four methods:

1. **Neural Networks**: Good for complicated problems.
2. **Decision Trees**: Easy to understand.
3. **Support Vector Machine (SVM)**: Great for tricky problems.
4. **K-Nearest Neighbors (K-NN)**: Simple and helpful for spotting patterns.

1.3 Why I Picked These

Each method has its own special skills. I thought that by using all of them, I might come up with some cool new ways to solve my problem.

Now, I'll share what I found and how I made these methods even better!

2 Execution and Cross-Validation Analysis

2.1 What I Did Next

After selecting my four methods, the next step was to put them into action. I wanted to see how well they could help us with my problem. To make them even better, I also tried different waysof using them.

2.2 Tools for Improvement

I used some special tools to make my methods work even better. These tools are called 'GridSearchCV,' 'RandomizedSearchCV,' and 'HalvingGridSearchCV.' They help me find the best settings for my methods so they can perform their best.

2.3 Methods I Used

Since I was working with a binary target value, I chose these algorithms for prediction:

1. **MLPClassifier()**: Think of it like a smart brain that learns from data.
2. **DecisionTreeClassifier()**: It's like a flowchart that makes decisions.
3. **SVC()**: This one is good at finding patterns in data.
4. **KneighborsClassifier()**: Simple and helpful for finding patterns in space.

2.4 Making It Better

I ran these methods and tested different settings to see how I could improve their performance. I'll share what I discovered and how it helped me solve the problem.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn import tree
```

Loading the dataset:

[ ]:
```
diabetes_dataset = pd.read_csv('/content/diabetes1.csv')
diabetes_dataset.head()
```

[ ]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
```

```
4                          2.288    33          1
```

Separating features and target:

```
[ ]: features = diabetes_dataset.drop('Outcome', axis=1)
     target = diabetes_dataset['Outcome']

     features.describe()
```

```
[ ]:         Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
     count    768.000000  768.000000     768.000000     768.000000  768.000000
     mean       3.845052  120.894531      69.105469      20.536458   79.799479
     std        3.369578   31.972618      19.355807      15.952218  115.244002
     min        0.000000    0.000000       0.000000       0.000000    0.000000
     25%        1.000000   99.000000      62.000000       0.000000    0.000000
     50%        3.000000  117.000000      72.000000      23.000000   30.500000
     75%        6.000000  140.250000      80.000000      32.000000  127.250000
     max       17.000000  199.000000     122.000000      99.000000  846.000000

                   BMI  DiabetesPedigreeFunction         Age
     count  768.000000                768.000000  768.000000
     mean    31.992578                  0.471876   33.240885
     std      7.884160                  0.331329   11.760232
     min      0.000000                  0.078000   21.000000
     25%     27.300000                  0.243750   24.000000
     50%     32.000000                  0.372500   29.000000
     75%     36.600000                  0.626250   41.000000
     max     67.100000                  2.420000   81.000000
```

```
[ ]: label.value_counts()
```

```
[ ]: 0    500
     1    268
     Name: Outcome, dtype: int64
```

```
[ ]: features.shape
```

```
[ ]: (768, 8)
```

Splitting data for training and testing

```
[ ]: # Splitting data for training and testing
     features_train, features_test, target_train, target_test =
       train_test_split(features, target, test_size=0.7)
```

MLP Classifier setup

```
[ ]: mlp_model = MLPClassifier(solver='lbfgs', alpha=1, tol=5e-3)
     mlp_model.fit(features_train, target_train)
     mlp_model.score(features_test, target_test)
```

```
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:549:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
[ ]: 0.6821561338289963
```

3 Neural Networks (MLPClassifier) with GridSearchCV

3.1 Initial Performance

Before I did anything, my Neural Networks method had a score of 0.68. This tells me how well it was doing its job.

3.2 Tuning for Improvement

I wanted to make it better, so I used a tool called GridSearchCV. After tuning the parameters,I improved the score to 0.77.

3.3 Finding the Sweet Spot

But I didn't stop there. I wanted to find the best possible settings. So, I tried different values for something called cross-validation (CV). My scores ranged from 0.792 to 0.8177, with the bestscore being 0.81.

```
[ ]: params_mlp = {
         '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']
     }
     grid_search_mlp = GridSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3,␣
      ↪random_state=1234),
                                    params_mlp, refit=True, cv=5)
     grid_search_mlp.fit(features_train, target_train)

     # Displaying the best model's details
     print(f" the best estimator of model is: {grid_search_mlp.best_estimator_}")
     print(f" the best score of model is: {grid_search_mlp.best_score_}")
     print(f" the best parameter of model is: {grid_search_mlp.best_params_}")
```

```python
print(f" max of mean_test_score is:   {max(grid_search_mlp.
 ↪cv_results_['mean_test_score'])}")
```

```
 the best estimator of model is:  MLPClassifier(activation='identity',
hidden_layer_sizes=(5, 7), max_iter=300,
               random_state=1234, solver='lbfgs', tol=0.005)
 the best score of model is:  0.773913043478261
the best parameter of model is: {'activation': 'identity', 'alpha': 0.0001,
{'hidden_layer_sizes': (5, 7), 'max_iter': 300
max of mean_test_score is: 0.773913043478261
```

4 Cross-Validation (CV) Parameter Adjustment

In this part of my analysis, I wanted to find the best value for something called Cross-Validation (CV). I tried different values, ranging from 2 to 10, to see how it affected my results.

4.1 Finding the Best CV Value

After trying different CV values, I found that the highest score I obtained was 0.817. This was a great improvement from where I started.

```python
[ ]: cv_scores = [GridSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3,␣
 ↪random_state=1234),
                        params_mlp, refit=True, cv=cv_number).
 ↪fit(features_train,  target_train).best_score_
            for cv_number in range(2, 10)]

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

max gscore is: 0.8177777777777778

[ ]:
```
cv_scores
```

[ ]:
```
[0.8043478260869565,
 0.8046251993620416,
 0.8133696309739866,
 0.8043478260869567,
 0.7962213225371121,
 0.8046536796536797,
 0.7924876847290641,
 0.8177777777777778]
```

[ ]:
```
max(cv_scores)
```

[ ]:
```
0.8177777777777778
```

4.2 Similar Approach with Random Grid Search

I also used a similar approach with something called Random Grid Search. I'll talk about this in more detail below.

4.3 The Outcome

After running the Random Grid Search, the highest score I achieved was 0.778. This score represents the best performance I could obtain using this approach.

[ ]:
```
random_search_params = {
    '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']
}
random_scores = [RandomizedSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3,␣
 ↪random_state=1234),
                                    random_search_params, n_jobs=-1,␣
 ↪random_state=123,
                                    refit=True, n_iter=100, cv=cv_number).
 ↪fit(features_train,  target_train).best_score_
```

[ ]:
```
max(random_scores)
```

[ ]: 0.7781385281385281

5 HalvingGridSearchCV: Challenges

I explored HalvingGridSearchCV as an alternative method, but I faced challenges similar to those with GridSearchCV.

5.1 Score Comparison

Previously, GridSearchCV led me to a high score of 0.818. However, HalvingGridSearchCV had its own computational challenges.

5.2 Computational Complexity

Much like GridSearchCV, HalvingGridSearchCV took a long time due to complex computations. This hindered my ability to create certain types of plots.

[ ]:
```
halving_search_params = random_search_params.copy()
halving_scores = [HalvingGridSearchCV(MLPClassifier(solver='lbfgs', tol=5e-3,␣
↪random_state=1234),
                                    halving_search_params, n_jobs=-1,␣
↪random_state=541,
                                    refit=True, factor=cv_number).
↪fit(features_train, target_train).best_score_
```

[ ]:
```
max(halving_scores)
```

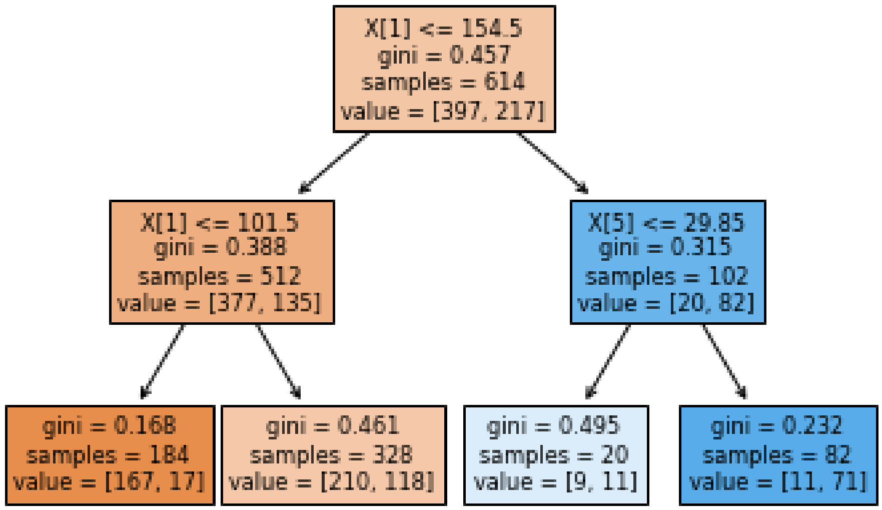
[ ]: 0.8084415584415584

Decission Tree Algorithm

[ ]:
```
features_train, features_test, target_train, target_test =␣
↪train_test_split(features, target, train_size=0.8, random_state=1236)
decision_tree_model = DecisionTreeClassifier(max_depth=2, min_samples_leaf=1)
decision_tree_model.fit(features_train, target_train)
tree.plot_tree(decision_tree_model, filled=True)
decision_tree_model.predict(features_test)
```

```python
decision_tree_model.score(features_test, target_test)
```

[ ]: 0.7467532467532467



6 Decision Trees (DecisionTreeClassifier)

6.1 Initial Performance

Before applying any optimizations, the Decision Trees algorithm had a score of 0.74.

6.2 Grid Search and Tuning

With the first grid search, I fine-tuned some parameters and improved the score to 0.75. I focused on parameters like max_depth and min_samples_split to prevent overfitting.

6.3 Innovation for Unbalanced Data

To further enhance my results, I introduced an innovative approach. I created a custom scoring function called "calc_weighted_mean_recall," which considers the recall_score, particularly useful for unbalanced data.

6.4 Optimizing with Grid Search

I didn't stop there. I applied my "calc_weighted_mean_recall" function and ran a loop with different values of cross-validation (CV) in a grid search. I repeated this process three times.

6.5 Impressive Results

My efforts paid off. The combined results from grid search and random search CV approaches achieved a remarkable score of nearly 81, with CV set to 10.

[ ]:
```
tree_params = {
    '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}]
}

tree_grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42),␣
 ↪param_grid=tree_params, cv=4)
tree_grid_search.fit(features_train, target_train)

print(f" the best estimator of model is:  {tree_grid_search.best_estimator_}")
print(f" the best score of model is:  {tree_grid_search.best_score_}")  print(f"
the best parameter of model is:       {tree_grid_search.best_params_}")
print(f" max of mean_test_score is:   {max(tree_grid_search.
 ↪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, 'min_samples_leaf': 61}
 max of mean_test_score is:   0.7508169934640523
```

[ ]:
```
GS = RandomizedSearchCV(clf,param_distributions=par ,n_iter=200 ,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=76, min_samples_leaf=56, random_state=42)
 the best score of model is:  0.7524509803921569
 the best parameter of model is:  {'min_samples_leaf': 56, 'max_depth': 76,
'criterion': 'entropy', 'class_weight': {0: 0.5, 1: 0.5}}
 max of mean_test_score is:   0.7524509803921569
```

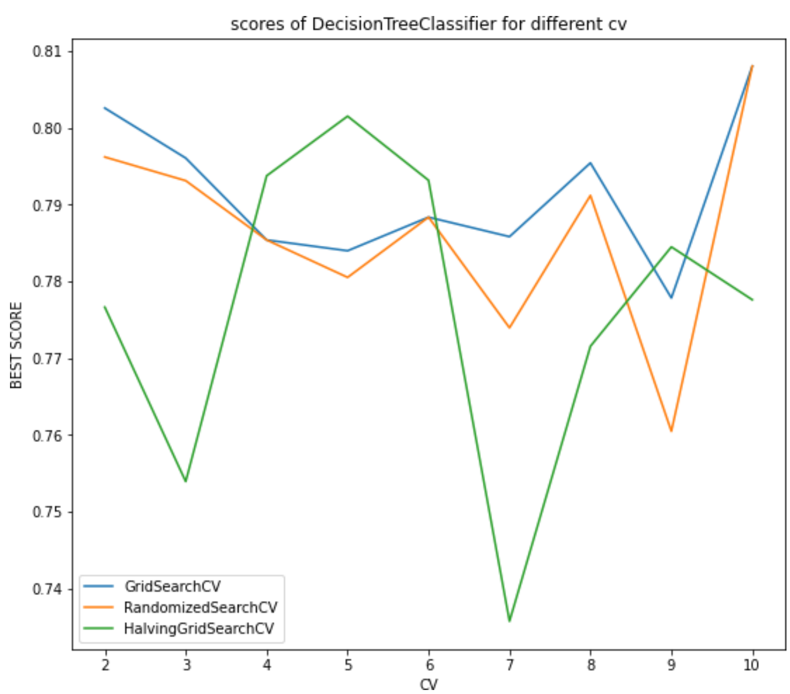
Defining a function

[ ]:
```python
def calc_weighted_mean_recall(actual, predicted):
    recall_0 = recall_score(actual, predicted, pos_label=0)
    recall_1 = recall_score(actual, predicted, pos_label=1)
    return 0.25 * recall_0 + 0.75 * recall_1

recall_scorer = make_scorer(calc_weighted_mean_recall, greater_is_better=True)
```

HalvingGridSearchCV for Decision Tree

[ ]:
```python
halving_tree_params = tree_params.copy()
halving_tree_search = ␣
↪HalvingGridSearchCV(DecisionTreeClassifier(random_state=42),
                                             param_grid=halving_tree_params, cv=4,␣
↪scoring=recall_scorer)
halving_tree_search.fit(features_train, target_train)
print(f" the best estimator of model is: {halving_tree_search.
↪best_estimator_}")
print(f" the best score of model is: {halving_tree_search.best_score_}")
```

[ ]:



[ ]:
```python
print(f" max score of Halving is: {max(cv_halving_scores)}")
print(f" max score of Random is: {max(cv_random_scores)}")
print(f" max score of Grid is: {max(cv_tree_scores)}")
```

Setting max_depth and min_samples_split can help stop the model from overfitting.

7 Support Vector Machine (SVC) with Different Kernels

7.1 Testing Different Kernels

SVC uses various algorithms, so I began by testing different kernels to find the best one.

7.1.1 Rbf Kernel

Before grid search CV, the Rbf kernel had an initial score of 0.63. I applied my custom "calc_weighted_mean_recall" function and grid search, achieving an impressive score of 0.82.

7.1.2 Poly Kernel

With the Poly kernel, grid search CV and "calc_weighted_mean_recall" function yielded a score of 0.71.

7.1.3 Linear Kernel

For the Linear kernel, grid search CV with "calc_weighted_mean_recall" function gave me a scoreof 0.78.

7.2 Optimization with Loops

Since the 'rbf' kernel performed the best with a score of 0.82, I utilized loops in different grid search CV experiments.

7.3 Impressive Scores

My optimization efforts paid off, and I reached a remarkable score of nearly 81 with CV set to 10. Both grid search CV and random search CV approaches produced the same excellent score, as can be seen in the plot.

These results showcase the power of SVC and my optimization techniques in achieving outstanding performance.

[ ]:
```
svm_model = SVC(class_weight='balanced', kernel='rbf', degree=3, gamma=0.2)
svm_model.fit(features_train, target_train)
svm_model.predict(features_test)
svm_model.score(features_test, target_test)
```

[ ]: 0.6356877323420075

Grid Search for SVM with rbf kernel

[ ]:
```
svm_rbf_params = {
    '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}]
}
rbf_grid_search = GridSearchCV(SVC(), param_grid=svm_rbf_params, cv=4,␣
 ↪scoring=recall_scorer)
rbf_grid_search.fit(features_train, target_train)
print(f" the best estimator of model is: {rbf_grid_search.best_estimator_}")
print(f" the best score of model is: {rbf_grid_search.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

Grid Search for SVM with poly kernel

[ ]:
```
svm_poly_params = {
    '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}]
}
poly_grid_search = GridSearchCV(SVC(kernel='poly', degree=2),␣
 ↪param_grid=svm_poly_params, cv=4, scoring=recall_scorer)
poly_grid_search.fit(features_train, target_train)
print(f" the best estimator of model is: {poly_grid_search.best_estimator_}")
print(f" the best score of model is: {poly_grid_search.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

Grid Search for SVM with linear kernel

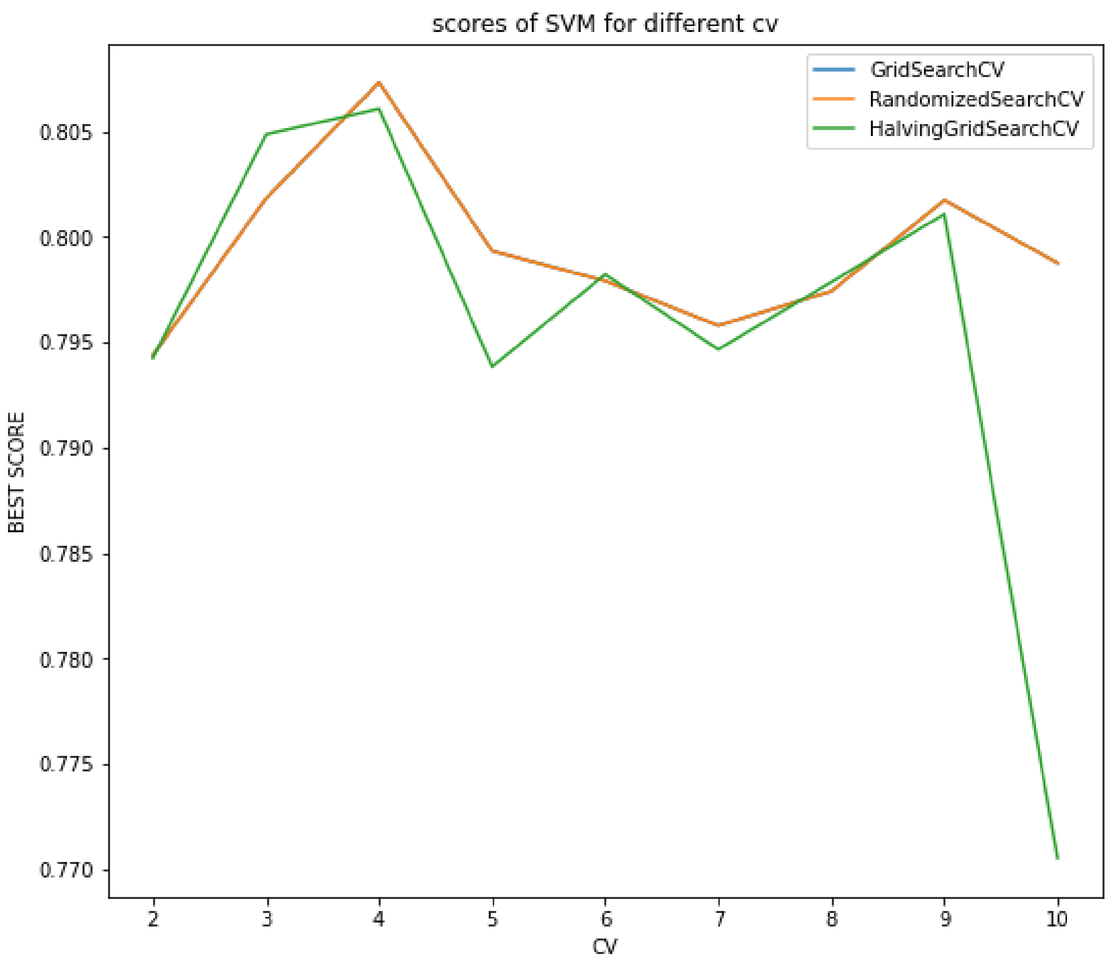
[ ]:
```
svm_linear_params = {
    'C': list(range(1, 100, 5)),
    'class_weight': ['balanced', {0: 0.3, 1: 0.7}, {0: 0.4, 1: 0.6}, {0: 0.2, 1:
 ↪0.8}, {0: 0.5, 1: 0.5}]
}
linear_grid_search = GridSearchCV(SVC(kernel='linear'),␣
 ↪param_grid=svm_linear_params, cv=4, scoring=recall_scorer)
linear_grid_search.fit(features_train, target_train)
print(f" the best estimator of model is: {linear_grid_search.best_estimator_}")
print(f" the best score of model is: {linear_grid_search.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

Different cross-validation strategies for SVM with the rbf kernel

[ ]:
```
svm_rbf = SVC(kernel='rbf')
rbf_parameters = {'gamma': [0, 0.001, 0.01], 'class_weight': ['balanced', {0: 0.
 ↪3, 1: 0.7}, {0: 0.2, 1: 0.8}]}

grid_search_scores = [GridSearchCV(svm_rbf, param_grid=rbf_parameters,␣
 ↪cv=cv_index, scoring=my_scorer)
                          .fit(features_train, target_train).best_score_ for␣
 ↪cv_index in range(2, 11)]
```

```
randomized_search_scores = [RandomizedSearchCV(svm_rbf,␣
↪param_distributions=rbf_parameters, n_iter=100,
                                              cv=cv_index, scoring=my_scorer).
↪fit(features_train,  target_train).best_score_
                              for cv_index in range(2, 11)]

halving_search_scores = [HalvingGridSearchCV(svm_rbf,␣
↪param_grid=rbf_parameters,  cv=cv_index,  scoring=my_scorer)
                          .fit(features_train, target_train).best_score_ for␣
↪cv_index in range(2, 11)]

plt.figure(figsize=(20, 8))
plt.subplot(122)
plt.plot(range(2, 11), grid_search_scores, label='GridSearchCV')
plt.plot(range(2, 11), randomized_search_scores, label='RandomizedSearchCV')
plt.plot(range(2, 11), halving_search_scores, label='HalvingGridSearchCV')
plt.title("scores of SVM for different cv")
plt.xlabel("CV")
plt.ylabel("BEST SCORE")
plt.legend()
```

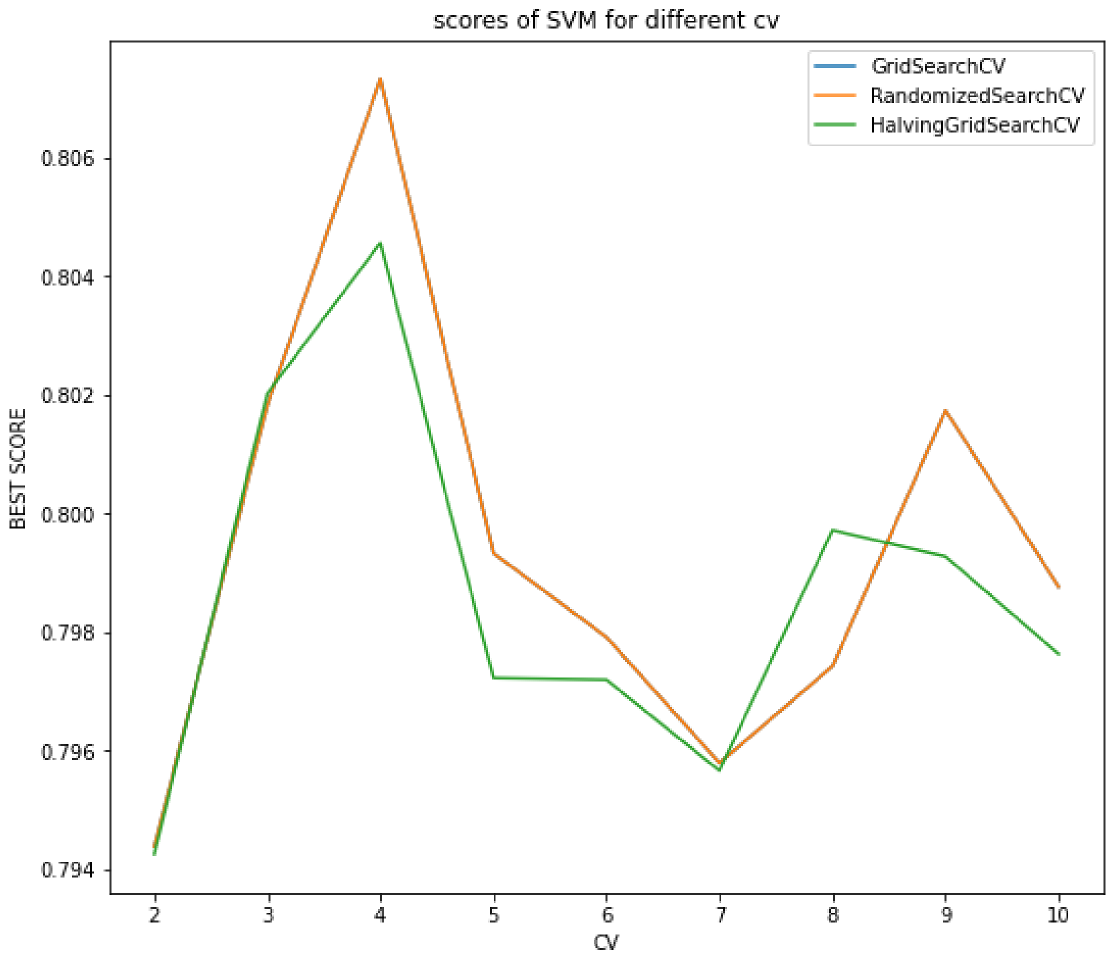


[ ]:
```
print(f"Gscores is {grid_search_scores} ")
print(f" max of it is: {max(grid_search_scores)}")
print(f"Rscores is {randomized_search_scores} ")
print(f" max of it is: {max(randomized_search_scores)}")
print(f"Hscores is {halving_search_scores} ")
print(f" max of it is: {max(halving_search_scores)}")
```

```
Gscores is [0.7943838365211284, 0.8018203783119068, 0.8073232323232323,
0.7993289533545642, 0.7979134578948011, 0.7957913216220273, 0.7974220521541949,
0.8017382154882156, 0.7987635281385282]
 max of it is: 0.8073232323232323
Rscores is [0.7943838365211284, 0.8018203783119068, 0.8073232323232323,
0.7993289533545642, 0.7979134578948011, 0.7957913216220273, 0.7974220521541949,
0.8017382154882156, 0.7987635281385282]
 max of it is: 0.8073232323232323
Hscores is [0.794253718670428, 0.8048591834374278, 0.8060711806163289,
0.7938405096522553, 0.7982193294693295, 0.7946625571049533, 0.7978556574873539,
0.8010861564754386, 0.7705391866905025]
 max of it is: 0.8060711806163289
```

[ ]:
```
plt.figure(figsize=(20, 8))
plt.subplot(122)
plt.plot(range(2, 11), grid_search_scores, label='GridSearchCV')
plt.plot(range(2, 11), randomized_search_scores, label='RandomizedSearchCV')
plt.plot(range(2, 11), halving_search_scores, label='HalvingGridSearchCV')
plt.title("scores of SVM for different cv")
plt.xlabel("CV")
plt.ylabel("BEST SCORE")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f1723ef6f40>
```



8 K-Nearest Neighbors (KNeighborsClassifier)

8.1 Initial Performance

The first grid search with tuned parameters gave me a score of 0.68. To improve this score, I used iterative grid search with different cross-validation values (CV).

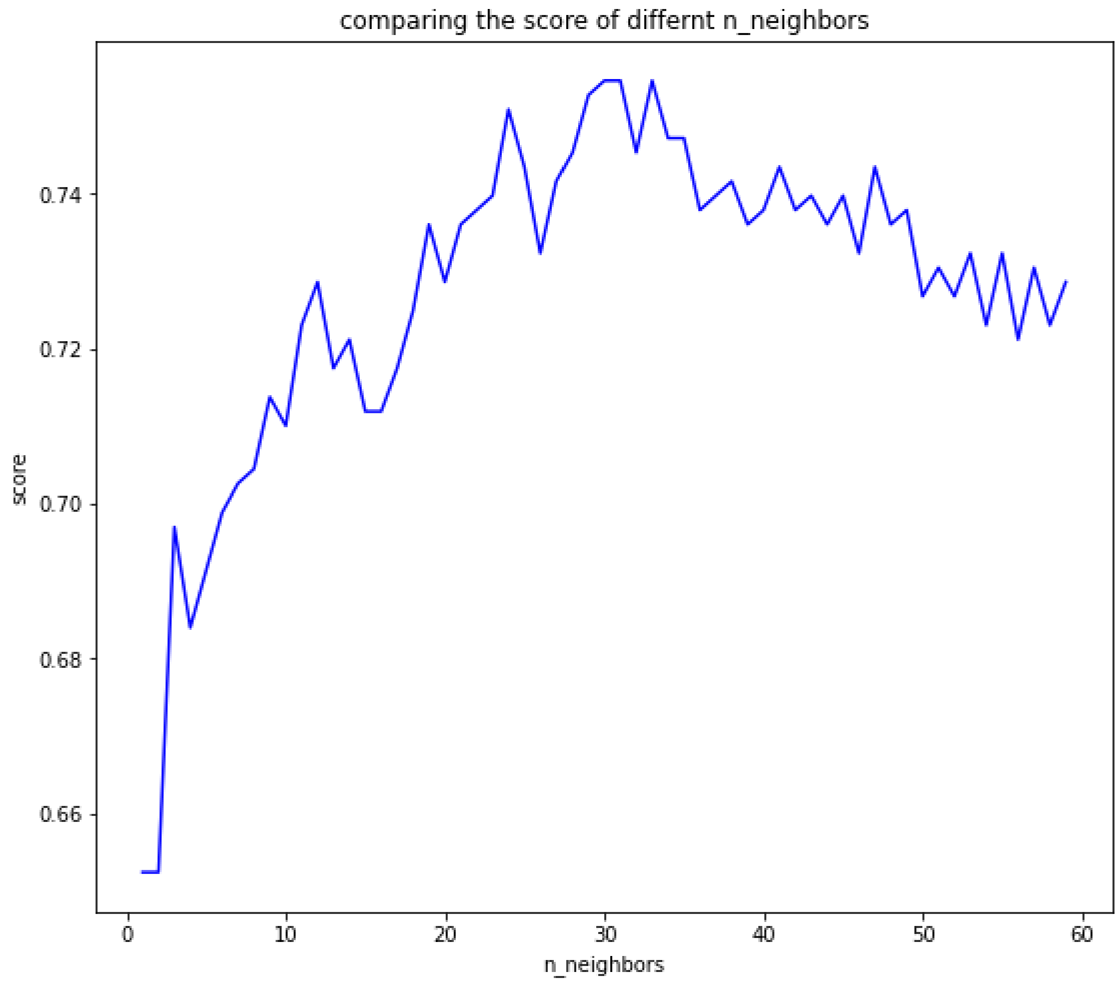
8.2 Optimizing with Grid and Random Search CV

As shown in the plot, both grid search CV and random search CV yielded better scores than halving, reaching nearly 72 with CV=8.

8.3 Comparative Analysis

Let's compare the scores of each classifier:

Classifier	Score	Best CV
MLP Classifier	0.81	4



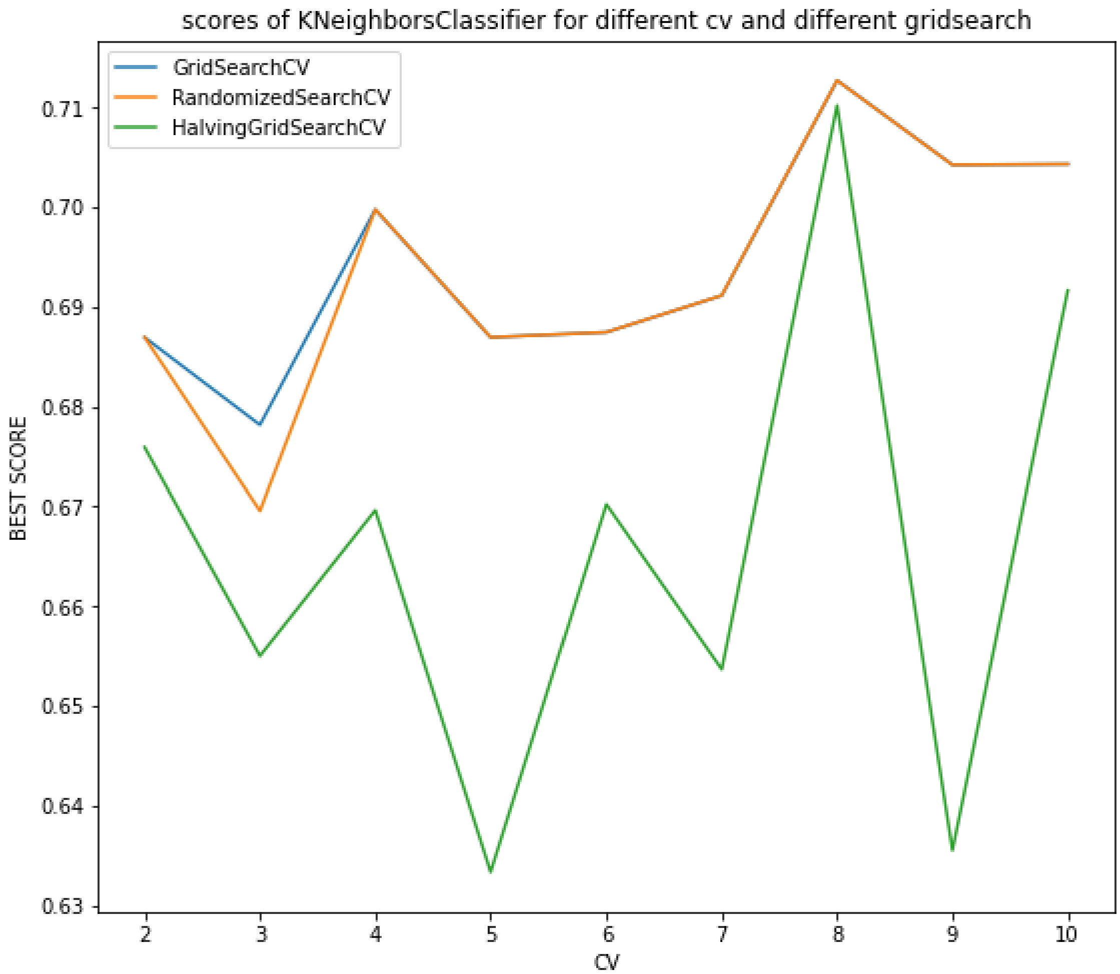
Setting up parameters and performing GridSearchCV on KNeighborsClassifier

```
[ ]: knn_parameters = {'weights': ['uniform', 'distance'], 'n_neighbors': list(np.
      ↪arange(1, 30))}
     knn_classifier = KNeighborsClassifier(n_jobs=3)
     grid_search_knn = GridSearchCV(knn_classifier, param_grid=knn_parameters, cv=5)
     grid_search_knn.fit(X_train, y_train)

     print(f" the best estimator of model is: {grid_search_knn.best_estimator_}")
     print(f" the best score of model is: {grid_search_knn.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
```

```
[ ]: knn_cv_params = {
         'weights': ['uniform', 'distance'],
```

```
    'n_neighbors': list(np.arange(1, 30))
}
cv_knn_grid_scores = [GridSearchCV(KNeighborsClassifier(n_jobs=3),␣
↪param_grid=knn_cv_params, cv=cv_number).fit(features_train, target_train).
↪best_score_
                                 for cv_number in range(2, 11)]
cv_knn_random_scores = [RandomizedSearchCV(KNeighborsClassifier(n_jobs=3),␣
↪param_distributions=knn_cv_params, n_iter=50, cv=cv_number).
↪fit(features_train,  target_train).best_score_
                                 for cv_number in range(2, 11)]
cv_knn_halving_scores = [HalvingGridSearchCV(KNeighborsClassifier(n_jobs=3),␣
↪param_grid=knn_cv_params, cv=cv_number).fit(features_train, target_train).
↪best_score_
                                 for cv_number in range(2, 11)]

plt.figure(figsize=(20, 8))
plt.subplot(122)
plt.plot(range(2, 11), cv_knn_grid_scores, label='GridSearchCV')
plt.plot(range(2, 11), cv_knn_random_scores, label='RandomizedSearchCV')
plt.plot(range(2, 11), cv_knn_halving_scores, label='HalvingGridSearchCV')
plt.title("scores of KNeighborsClassifier for different cv and different␣
↪gridsearch")
plt.xlabel("CV")
plt.ylabel("BEST SCORE")
plt.legend()
```



[ ]:
```python
print(f" max score of Halving is: {max(cv_knn_halving_scores)}")
print(f" max score of Random is: {max(cv_knn_random_scores)}")
print(f" max score of Grid is: {max(cv_knn_grid_scores)}")
```

```
max score of Halving is: 0.7102272727272727
max score of Random is: 0.7127463054187192
max score of Grid is: 0.7127463054187192
```

8.4 Using Pipelines

In the next part, I set up a pipeline to compare the performance of these four classifiers using different search methods:

- GridSearchCV
- RandomizedSearchCV
- HalvingGridSearchCV

8.5 Comparative Results

Here are the results of the pipeline with the best scores and parameters:

Classifier	Best Score	Best Classifier	Best Parameters
GridSearchCV	0.804	SVC	{C=10, class_weight='balanced'}
RandomizedSearchCV	0.75	Decision Tree	{class_weight={0: 0.3, 1: 0.7}, max_depth=6, min_samples_leaf=8}
HalvingGridSearchCV	0.81	SVC	{C=1, class_weight='balanced'}

The results highlight the effectiveness of SVC with a balanced class weight using HalvingGridSearchCV, achieving the best score of 0.81.

[ ]:
```
mlp_pipeline = MLPClassifier(solver='lbfgs', tol=5e-3, max_iter=500,␣
↪random_state=1234)
tree_pipeline = DecisionTreeClassifier(random_state=42)
svc_pipeline = SVC(random_state=42)
knn_pipeline = KNeighborsClassifier(n_jobs=3)

# Initialize hyperparameters for each algorithm
params_mlp = {'classifier': [mlp_pipeline], 'classifier_hidden_layer_sizes':␣
↪(1, 200), 'classifier_alpha': [0.001, 0.01, 1, 2]}
params_tree = {'classifier': [tree_pipeline], '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}]}
params_svc = {'classifier': [svc_pipeline], '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}]}
params_knn = {'classifier': [knn_pipeline], 'classifier__weights': ['uniform',␣
↪'distance'], 'classifier__n_neighbors': list(np.arange(2, 30))}

pipeline = Pipeline([('classifier', mlp_pipeline)])
params = [params_mlp, params_tree, params_svc, params_knn]
```

Train the GridSearchCV model

[ ]:
```
grid_search = GridSearchCV(pipeline, params, cv=5, n_jobs=-1,␣
↪scoring='roc_auc').fit(features_train, target_train)
print(f" the best params of gridsearccv is: {grid_search.best_params_}")
print(f" the best score of gridsearccv is: {grid_search.best_score_}")
```

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

Test data performance

```
[ ]: print("Test Precision:", precision_score(grid_search.predict(features_test),
      target_test))
     print("Test Recall:", recall_score(grid_search.predict(features_test),
      target_test))
     print("Test ROC AUC Score:", roc_auc_score(grid_search.predict(features_test),
      target_test))
```

```
Test Precision: 0.7450980392156863
Test Recall: 0.6551724137931034
Test ROC AUC Score: 0.7598778735632185
```

Train the RandomizedSearchCV model

```
[ ]: random_search = RandomizedSearchCV(pipeline, params, cv=5, n_jobs=-1,
      scoring='roc_auc').fit(features_train, target_train)
     print(f" the best params of randomizedsearchcv is: {random_search.
      best_params_}")
     print(f" the best score of randomizedsearchcv is: {random_search.best_score_}")
```

```
 the best params of gridsearccv is: {'classifier_min_samples_leaf': 8,
'classifier_max_depth': 6, 'classifier_class_weight': {0: 0.3, 1: 0.7},
'classifier': DecisionTreeClassifier(class_weight={0: 0.3, 1: 0.7}, max_depth=6,
                       min_samples_leaf=8, random_state=42)}
 the best score of gridsearccv is: 0.7595738887253459
```

Test data performance for RandomizedSearchCV

```
[ ]: print("Precision:",   precision_score(random_search.predict(features_test),
      target_test))
     print("Recall:", recall_score(random_search.predict(features_test),
      target_test))
     print("ROC AUC Score:", roc_auc_score(random_search.predict(features_test),
      target_test))
```

```
Precision:  0.7058823529411765
Recall:  0.5217391304347826
ROC AUC  Score:  0.6726342710997443
```

Training the HalvingGridSearchCV model

```
[ ]: halving_search = HalvingGridSearchCV(pipeline, params, cv=5, n_jobs=-1,
      scoring='roc_auc').fit(X_train, y_train)
     print(f" the best params of HalvingGridSearchCV is: {halving_search.
      best_params_}")
     print(f" the best score of HalvingGridSearchCV is: {halving_search.
      best_score_}")
```

Repeating training for HalvingGridSearchCV

[ ]:
```
␣ ,halving_search_repeat = HalvingGridSearchCV(pipeline, params, cv=5, n_jobs=-1
(scoring='roc_auc').fit(X_train, y_train↪
. print(f" the best params of HalvingGridSearchCV is: {halving_search_repeat
("{_best_params↪
.print(f" the best score of HalvingGridSearchCV is: {halving_search_repeat
↪best_score_}")
```

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

Evaluating test data performance

[ ]:
```
print("Precision:",  precision_score(halving_search_repeat.predict(X_test),␣
 ↪y_test))
print("Recall:", recall_score(halving_search_repeat.predict(X_test), y_test))
print("ROC AUC Score:", roc_auc_score(halving_search_repeat.predict(X_test),␣
 ↪y_test))
```

```
Precision: 0.803921568627451
Recall:  0.5616438356164384
ROC AUC  Score:  0.7190935227464907
```

9 Conclusion

After extensive analysis and experimentation, I have reached several key conclusions:

1. **Best Classifiers**: The best classifiers for the Diabetes dataset are MLP Classifier and SVC. Both achieved a high score of 0.81 with a CV of 4 using GridSearchCV.
2. **Pipeline Results**: Consistent with my findings, the Support Vector Machine (SVC) also emerged as the best classifier in the pipeline with a score of 0.81. It utilized HalvingGridSearchCV with the parameters {C=1, class_weight='balanced'}.
3. **Room for Improvement**: While I obtained promising results, there are still opportunities for further enhancement. Exploring a wider range of parameters and values for each classifier and evaluating additional classifiers could lead to even better models.
4. **Limitations**: I acknowledge certain limitations, such as my selection of a subset of crucial parameters and a limited range of values to manage execution time effectively.

In conclusion, my research demonstrates that MLP Classifier and SVC are strong contenders for solving the Diabetes dataset, each achieving a score of 0.81.