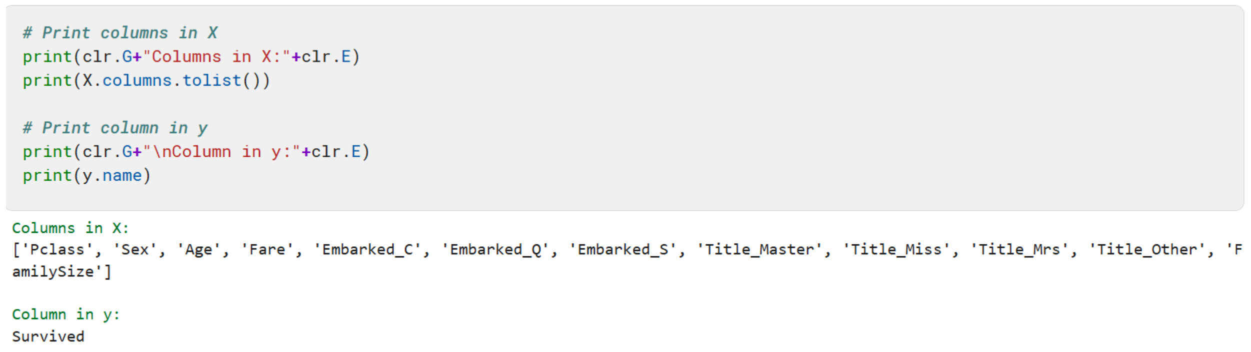
COS30082 - Applied Machine Learning

Week 4 - Lab - SVM

Huynh Trung Chien - 104848770

1. The columns selected for prediction

For this lab, the data preprocessing steps I have done are the same as the previous Logistic Regresison lab. Therefore, in this lab, I just summarized the final features after preprocessing and the chosen features for training.



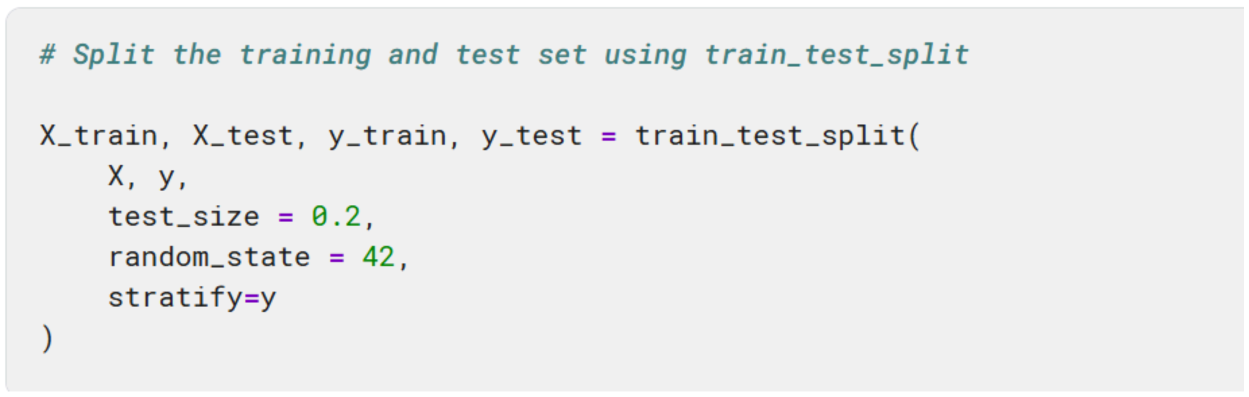
```
# Print columns in X
print(clr.G+"Columns in X:"+clr.E)
print(X.columns.tolist())

# Print column in y
print(clr.G+"\nColumn in y:"+clr.E)
print(y.name)
```

```
Columns in X:
['Pclass', 'Sex', 'Age', 'Fare', 'Embarked_C', 'Embarked_Q', 'Embarked_S', 'Title_Master', 'Title_Miss', 'Title_Mrs', 'Title_Other', 'F
amilySize']

Column in y:
Survived
```

Figure 1: Columns in X and y.

2. The training and testing split

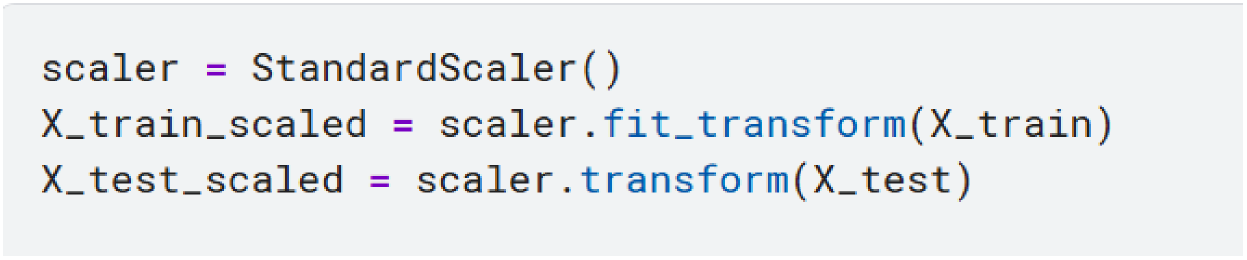
Then, I splited the train and test sets using ***train_test_split()*** with ***test_size = 0.2***, ***random_state = 42***, and ***stratify = y***.



```
# Split the training and test set using train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = 0.2,
    random_state = 42,
    stratify=y
)
```

Figure 2: Splitting the training and testing sets

After that, I used ***StandardScaler()*** for Normalization step.



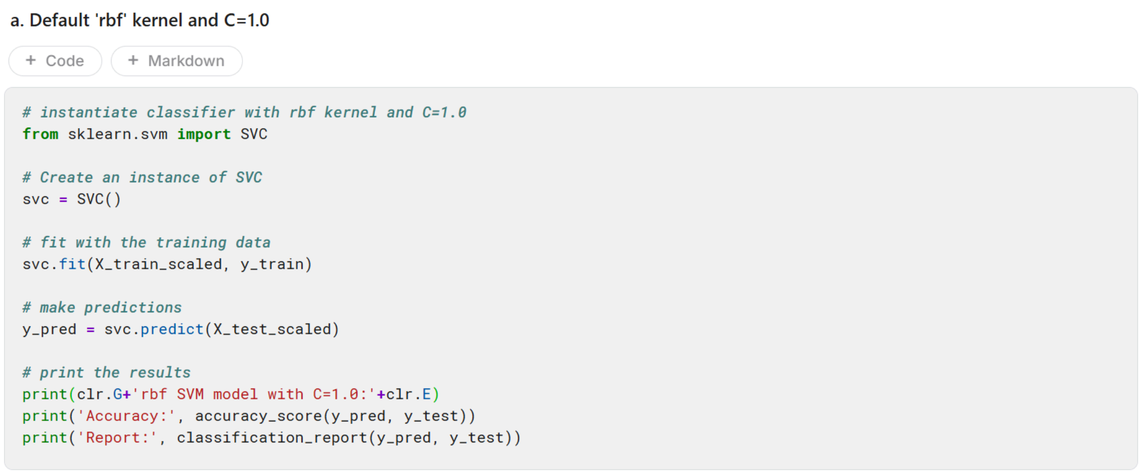
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 3: Normalized X_train and X_test

3. The SVM model building

For SVM model buildings, try 4 different ways:

- First, I built simple models with different kernels (rbf, linear, poly, sigmoid) and different hyperparameter 'C's manually, the figure below is an example, and I did the same for other kernels and hyperparameters:

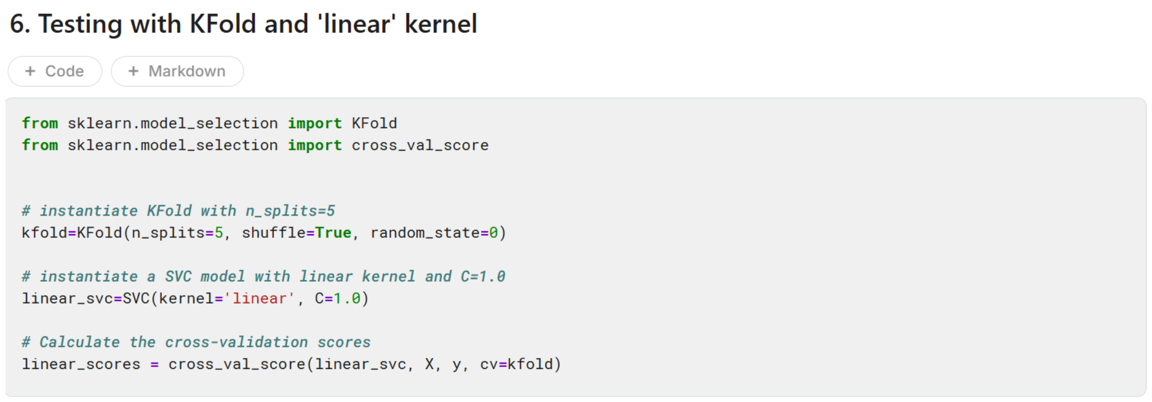


a. Default 'rbf' kernel and C=1.0

```
# instantiate classifier with rbf kernel and C=1.0
from sklearn.svm import SVC

# Create an instance of SVC
svc = SVC()

# fit with the training data
svc.fit(X_train_scaled, y_train)

# make predictions
y_pred = svc.predict(X_test_scaled)

# print the results
print(clr.G+'rbf SVM model with C=1.0:'+clr.E)
print('Accuracy:', accuracy_score(y_pred, y_test))
print('Report:', classification_report(y_pred, y_test))
```

Figure 4: Model for 'rbf' kernel with C=1.0

- Secondly, I observed that 'linear' kernel performed really well, so I tried to test with stratified Kfold to observe whether the performance will increase or not:

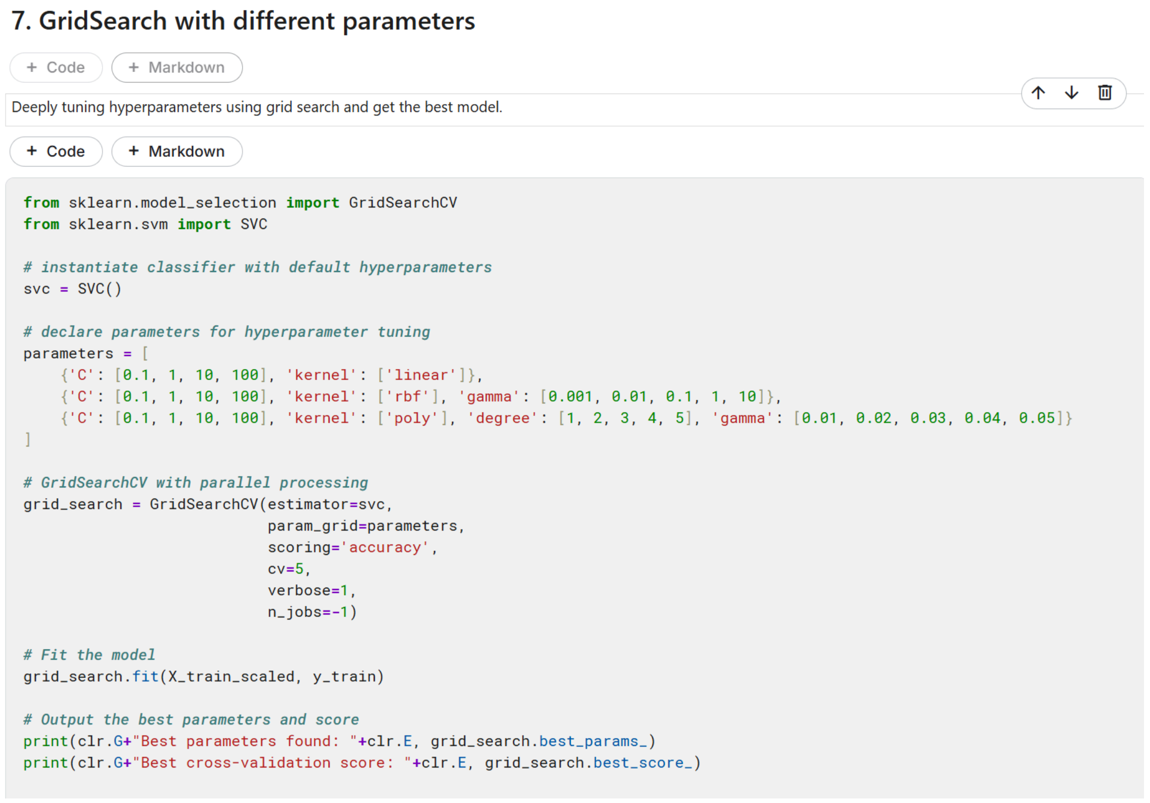


6. Testing with KFold and 'linear' kernel

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


# instantiate KFold with n_splits=5
kfold=KFold(n_splits=5, shuffle=True, random_state=0)

# instantiate a SVC model with linear kernel and C=1.0
linear_svc=SVC(kernel='linear', C=1.0)

# Calculate the cross-validation scores
linear_scores = cross_val_score(linear_svc, X, y, cv=kfold)
```

Figure 5: Stratified KFold for SVM model with linear kernel

- Thirdly, I used GridSearch to deeply tune hyperparameters including ‘C’, ‘degree’, and ‘gamma’, and print out the best result:



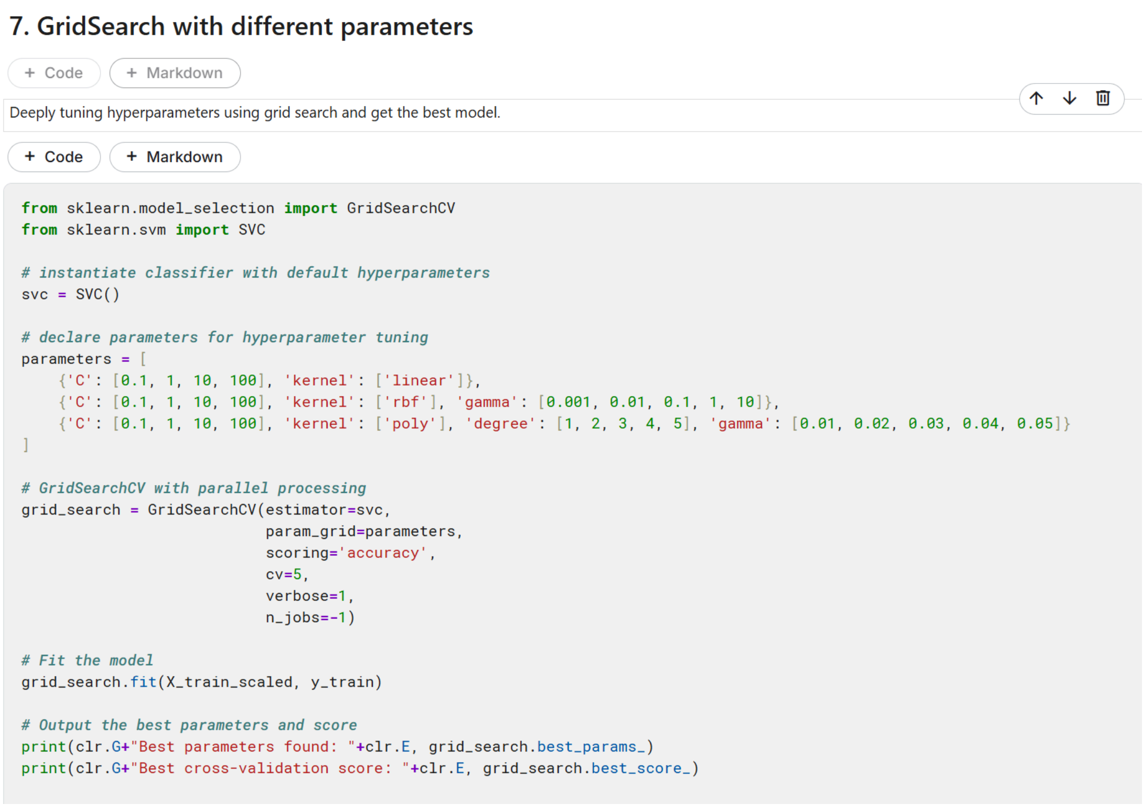
7. GridSearch with different parameters

Deeply tuning hyperparameters using grid search and get the best model.

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# instantiate classifier with default hyperparameters
svc = SVC()

# declare parameters for hyperparameter tuning
parameters = [
    {'C': [0.1, 1, 10, 100], 'kernel': ['linear']},
    {'C': [0.1, 1, 10, 100], 'kernel': ['rbf'], 'gamma': [0.001, 0.01, 0.1, 1, 10]},
    {'C': [0.1, 1, 10, 100], 'kernel': ['poly'], 'degree': [1, 2, 3, 4, 5], 'gamma': [0.01, 0.02, 0.03, 0.04, 0.05]}
]

# GridSearchCV with parallel processing
grid_search = GridSearchCV(estimator=svc,
                           param_grid=parameters,
                           scoring='accuracy',
                           cv=5,
                           verbose=1,
                           n_jobs=-1)

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Output the best parameters and score
print(clr.G+"Best parameters found: "+clr.E, grid_search.best_params_)
print(clr.G+"Best cross-validation score: "+clr.E, grid_search.best_score_)
```

Figure 6: Using GridSearch to deeply tune the models with different parameters and kernels.

- Finally, I did the same GridSearch with different test sizes:



7. GridSearch with different parameters

Deeply tuning hyperparameters using grid search and get the best model.

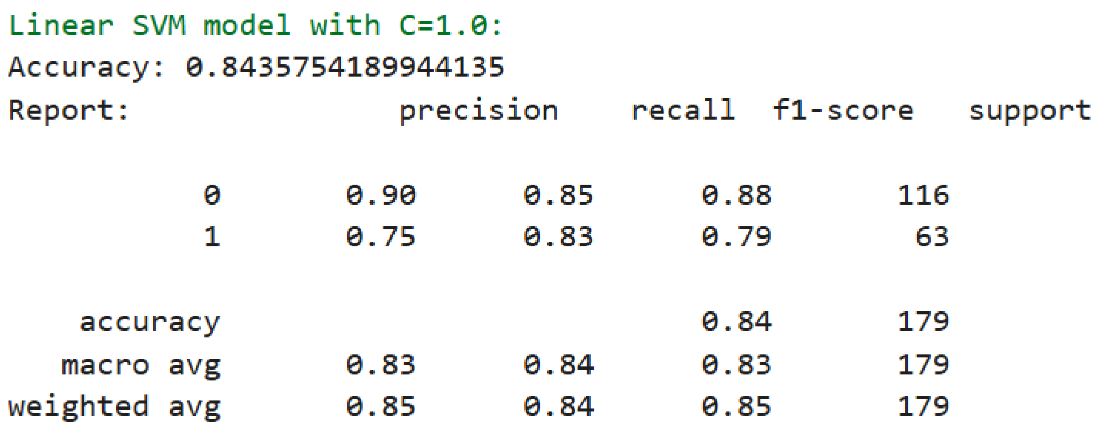
```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# instantiate classifier with default hyperparameters
svc = SVC()

# declare parameters for hyperparameter tuning
parameters = [
    {'C': [0.1, 1, 10, 100], 'kernel': ['linear']},
    {'C': [0.1, 1, 10, 100], 'kernel': ['rbf'], 'gamma': [0.001, 0.01, 0.1, 1, 10]},
    {'C': [0.1, 1, 10, 100], 'kernel': ['poly'], 'degree': [1, 2, 3, 4, 5], 'gamma': [0.01, 0.02, 0.03, 0.04, 0.05]}
]

# GridSearchCV with parallel processing
grid_search = GridSearchCV(estimator=svc,
                           param_grid=parameters,
                           scoring='accuracy',
                           cv=5,
                           verbose=1,
                           n_jobs=-1)

# Fit the model
grid_search.fit(X_train_scaled, y_train)

# Output the best parameters and score
print(clr.G+"Best parameters found: "+clr.E, grid_search.best_params_)
print(clr.G+"Best cross-validation score: "+clr.E, grid_search.best_score_)
```

Figure 7: GridSearch with different test sizes

4. The accuracy

For the accuracy of the models, I will show the best for each way of building the models, but generally the results are quite similar to the results in Logistic Regression lab.

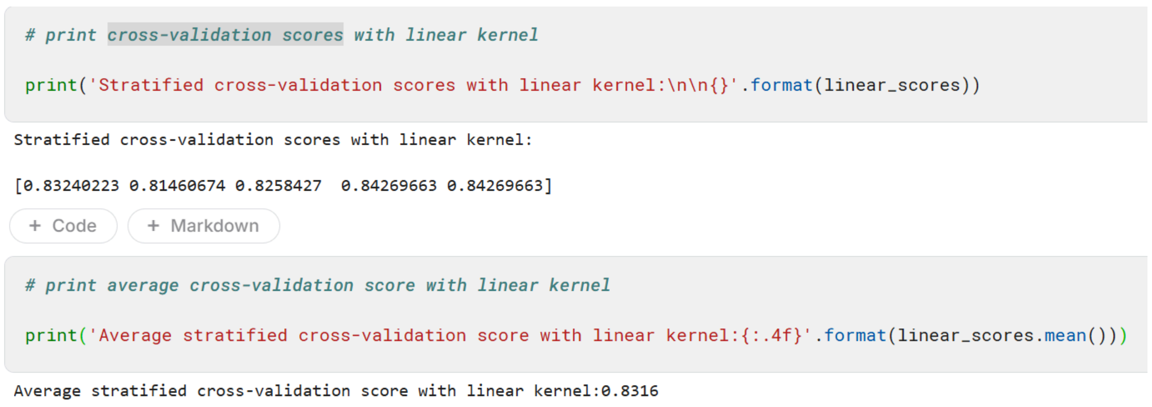
- For simple model, SVM with linear kernel performs the best, the accuracy are similar with different values of 'C':



```
Linear SVM model with C=1.0:
Accuracy:  0.8435754189944135
Report:               precision    recall  f1-score   support

           0       0.90      0.85      0.88       116
           1       0.75      0.83      0.79        63

    accuracy                           0.84       179
   macro avg       0.83      0.84      0.83       179
weighted avg       0.85      0.84      0.85       179
```

Figure 8: Accuracy for simple linear SVM model

- For stratified KFold with linear kernel and C=1.0, here is the result:



```
# print cross-validation scores with linear kernel

print('Stratified cross-validation scores with linear kernel:\n\n{}'.format(linear_scores))
```

```
Stratified cross-validation scores with linear kernel:

[0.83240223 0.81460674 0.8258427  0.84269663 0.84269663]
```

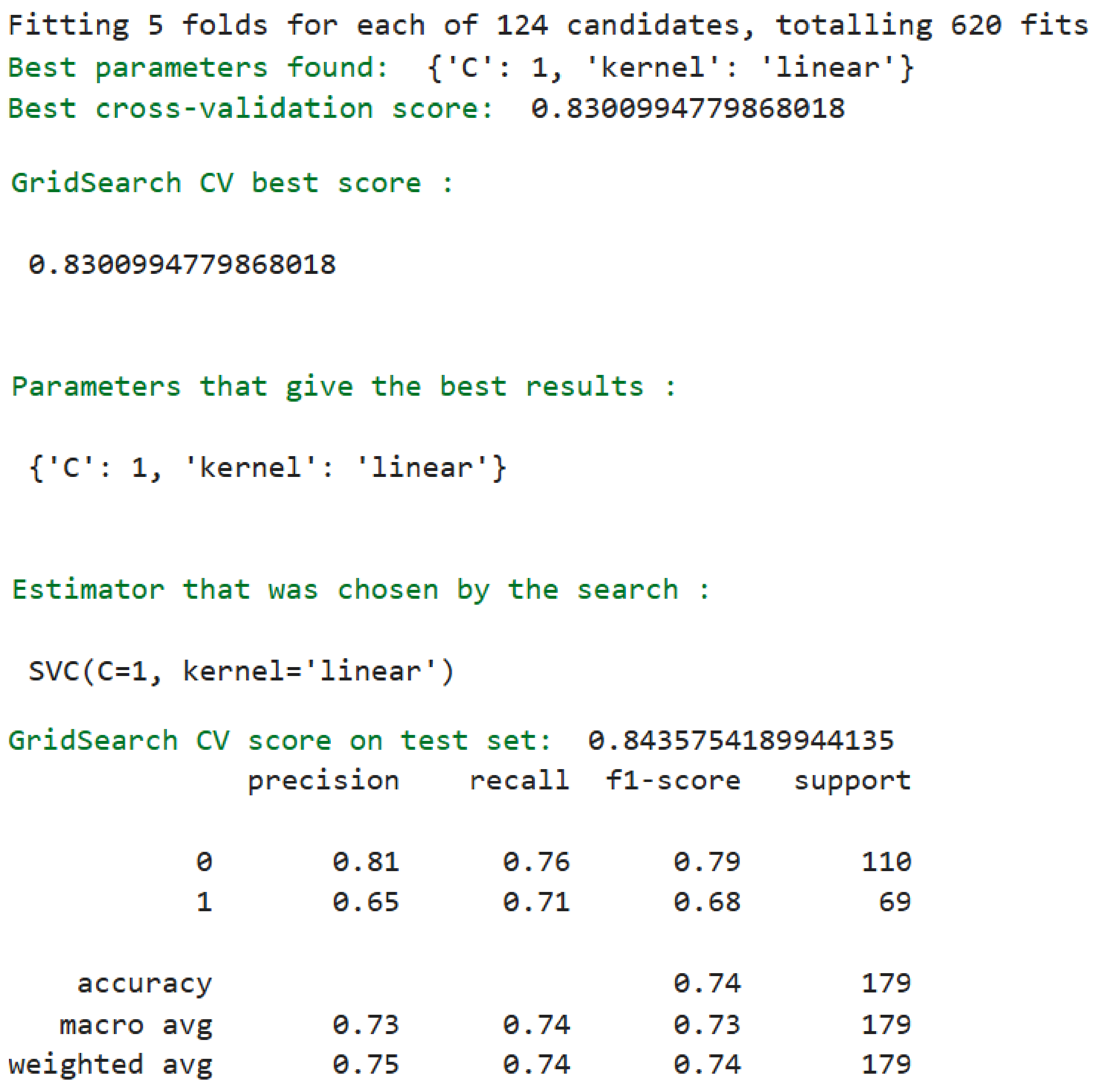
+ Code + Markdown

```
# print average cross-validation score with linear kernel

print('Average stratified cross-validation score with linear kernel:{:.4f}'.format(linear_scores.mean()))
```

```
Average stratified cross-validation score with linear kernel:0.8316
```

Figure 9: Results for stratified KFold with linear kernel and C=1.0

- For GridSearch with split_size=0.2, here is the results:



```
Fitting 5 folds for each of 124 candidates, totalling 620 fits
Best parameters found:  {'C': 1, 'kernel': 'linear'}
Best cross-validation score:  0.8300994779868018

GridSearch CV best score :

 0.8300994779868018


Parameters that give the best results :

 {'C': 1, 'kernel': 'linear'}


Estimator that was chosen by the search :

 SVC(C=1, kernel='linear')

GridSearch CV score on test set:  0.8435754189944135
              precision    recall  f1-score   support

           0       0.81      0.76      0.79       110
           1       0.65      0.71      0.68        69

    accuracy                           0.74       179
   macro avg       0.73      0.74      0.73       179
weighted avg       0.75      0.74      0.74       179
```

Figure 10: Results for GridSearch with split_size = 0.2

- For GridSearch with different split sizes, the results are as below:

```
========================================
Test size: 0.1
========================================
Best Parameters: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Best CV Accuracy: 0.8377
Test Set Accuracy: 0.8111
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.85        55
           1       0.82      0.66      0.73        35

    accuracy                           0.81        90
   macro avg       0.81      0.78      0.79        90
weighted avg       0.81      0.81      0.81        90
```

```
========================================
Test size: 0.2
========================================
Best Parameters: {'C': 1, 'kernel': 'linear'}
Best CV Accuracy: 0.8301
Test Set Accuracy: 0.8436
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.90      0.88       110
           1       0.83      0.75      0.79        69

    accuracy                           0.84       179
   macro avg       0.84      0.83      0.83       179
weighted avg       0.84      0.84      0.84       179
```

```
========================================
Test size: 0.3
========================================
Best Parameters: {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
Best CV Accuracy: 0.833
Test Set Accuracy: 0.8358
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       165
           1       0.82      0.74      0.78       103

    accuracy                           0.84       268
   macro avg       0.83      0.82      0.82       268
weighted avg       0.83      0.84      0.83       268
```

```
========================================
Test size: 0.4
========================================
Best Parameters: {'C': 1, 'degree': 3, 'gamma': 0.05, 'kernel': 'poly'}
Best CV Accuracy: 0.8184
Test Set Accuracy: 0.8291
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.95      0.87       220
           1       0.88      0.64      0.74       137

    accuracy                           0.83       357
   macro avg       0.84      0.79      0.81       357
weighted avg       0.84      0.83      0.82       357
```

Figure 11: Results for GridSearch with different split sizes