Lab 04- Extended Exercises on Classification and Pipelines

```
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel,
mutual_info_classif
from sklearn import model_selection
from sklearn.metrics import roc_auc_score, balanced_accuracy_score,
confusion_matrix, ConfusionMatrixDisplay, silhouette_score
from sklearn.model_selection import train_test_split
```

You are the Senior Data Scientist in a learning platform called LernTime. You have realized that many users stop using the platform and want to increase user retention. For this purpose, you decide to build a model to predict whether a student will stop using the learning platform or not.

Your data science team built a data frame in which each row contains the aggregated features per student (calculated over the first 5 weeks of interactions) and the feature `dropout` indicates whether the student stopped using the platform (1) or not (0) before week 10.

The dataframe is in the file `lerntime.csv` and contains the following features:

- `video_time`: total video time (in minutes)
- `num_sessions` total number of sessions
- `num_quizzes`: total number of quizzes attempts
- `reading_time`: total theory reading time
- `previous_knowledge`: standardized previous knowledge
- `browser_speed`: standardized browser speed
- `device`: whether the student logged in using a smartphone (1) or a computer (-1)
- `topics`: the topics covered by the user
- `education`: current level of education (0: middle school, 1: high school, 2: bachelor, 3: master, 4: Ph.D.).
- `dropout`: whether the student stopped using the platform (1) or not (0) before week 5.

The newest data scientist created two models with an excellent performance. As a Senior Data Scientist, you are suspicious of the results and decide to revise the code.

Your task is to:

a) Identify the mistakes. In the first cell, add a comment above each line in which you identify an error and explain the error.

b) In the second cell, you must correct the code.

```
df = pd.read_csv('data/lerntime_dropout.csv')

y = df['dropout']
X = df[['video_time', 'num_sessions', 'num_quizzes', 'reading_time',
       'previous_knowledge', 'browser_speed']]
```

```
len(df)
```

```
300
```

Task A) Identify the mistakes in the code

In the following cell, add a comment above each line in which you identify an error and explain the why it is erroneous. Please start each of your comments with `#ERROR:`. For example:

```
#ERROR: the RMSE of the model is printed instead of the AUC
```

```
print("The AUC of the model is: {}".format(rmse))
```

You may assume that:

- all the features are continous and numerical.
- the features have already been cleaned and processed.

```
# ERROR: Train-test split should be done before preprocessing steps 1.
and 2. to avoid data leakage,
# fitting both scaler and selector only on X_train
## 1. Scale the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

## 2. Feature selection (Lasso)
print(X.shape)
lasso = Lasso(alpha=0.1, random_state=0).fit(X, y)
selector = SelectFromModel(lasso, prefit = True)
X = selector.transform(X)
print(X.shape)

## 3. Split the data
# ERROR: The split should be done before the feature selection or
transformation
# to avoid data leaking
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=0)

## Model 1
```

```
clf = RandomForestClassifier(n_estimators=10, max_depth=1,
random_state=0)
# ERROR: Fit should only be called on the train set
clf.fit(X,y)
preds = clf.predict(X_test)
# ERROR: The adjusted mutual information is not an appropriate score
for classification, since it would give
# a perfect score even if the predictions are the complete opposite of
y_test
print("Score model 1:
{}".format(np.round(adjusted_mutual_info_score(preds, y_test), 2)))

## Model 2
clf = RandomForestClassifier(n_estimators=1000, max_depth=None,
random_state=0)
# ERROR: Fit should only be called on the train set
clf.fit(X,y)
preds = clf.predict(X_test)
# ERROR: The adjusted mutual information is not an appropriate score
for classification, since it would give
# a perfect score even if the predictions are the complete opposite of
y_test
print("Score model 2:
{}".format(np.round(adjusted_mutual_info_score(preds, y_test), 2)))

# ERROR: The second model has just more complexity and can hence
better overfit to the test set, which leaked during training
## Discussion
# Our second model achieved perfect results with unseen data and
outperforms the first model.
## This is because we increased the number of estimators.
```

```
(300, 3)
(300, 3)
Score model 1: 0.05
Score model 2: 1.0
```

Task B) Correct the code

Correct all the erroneous code in the following cell.

```
## 1. Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=0)

## 2. Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

## 3. Feature selection (Lasso)
```

```python
print(X_train.shape)
lasso = Lasso(alpha=0.1, random_state=0).fit(X_train, y_train)
selector = SelectFromModel(lasso, prefit = True)
X_train = selector.transform(X_train)
X_test = selector.transform(X_test)
print(X_train.shape)

## Model 1
clf = RandomForestClassifier(n_estimators=10, max_depth=1,
random_state=0)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
print("Score model 1:
{}".format(np.round(balanced_accuracy_score(preds, y_test), 2)))

## Model 2
clf = RandomForestClassifier(n_estimators=1000, max_depth=None,
random_state=0)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
print("Score model 2:
{}".format(np.round(balanced_accuracy_score(preds, y_test), 2)))

## Discussion
# Our first model outperformed the second model.
# However, it is not clear why because we change the number of
estimators and the maximum depth at the same time
```

```
(240, 3)
(240, 3)
Score model 1: 0.9
Score model 2: 0.81
```

Task C) Re-write your code using pipelines.

Hint: Go over sklearn-pipeline-introduction.

```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedGroupKFold

scalers =  [
    StandardScaler(),
    'passthrough'] # none

feature_selectors = [
    SelectFromModel(Lasso(alpha=0.1, random_state=0)),
    'passthrough'
]

steps = [('scaler', StandardScaler()), # preprocessing steps
```

```python
         ('feature_selector', SelectFromModel(Lasso(alpha=0.1,
random_state=0))), # Feature selection
         ('clf', RandomForestClassifier(random_state=0))]
# Model

param_grid = {
    'scaler': scalers,
    'feature_selector':feature_selectors,
    'clf__n_estimators': [10,1000],
    'clf__max_depth':[1, None]
}

pipeline = Pipeline(steps)

search = GridSearchCV(pipeline, param_grid, n_jobs=-1, cv = 5, scoring
= "balanced_accuracy")
search.fit(X, y)
print("Best parameter (CV score=%0.2f):" % search.best_score_)
print(search.best_params_)
```

```
Best parameter (CV score=0.81):
{'clf__max_depth': None, 'clf__n_estimators': 1000,
'feature_selector': SelectFromModel(estimator=Lasso(alpha=0.1,
random_state=0)), 'scaler': StandardScaler()}
```

```python
results = pd.DataFrame(search.cv_results_)
results.sort_values('rank_test_score')[[
       'param_clf__max_depth', 'param_clf__n_estimators',
       'param_feature_selector', 'param_scaler', 'params',
'mean_test_score', 'std_test_score',
       'rank_test_score']]
```

```
   param_clf__max_depth param_clf__n_estimators  \
12                 None                    1000
14                 None                    1000
13                 None                    1000
15                 None                    1000
8                  None                      10
9                  None                      10
10                 None                      10
11                 None                      10
0                     1                      10
1                     1                      10
2                     1                      10
3                     1                      10
4                     1                    1000
5                     1                    1000
6                     1                    1000
7                     1                    1000
```

```
                              param_feature_selector  
param_scaler  \
12  SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
StandardScaler()   
14                                        passthrough  
StandardScaler()   
13  SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
passthrough   
15                                        passthrough  
passthrough   
8   SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
StandardScaler()   
9   SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
passthrough   
10                                        passthrough  
StandardScaler()   
11                                        passthrough  
passthrough   
0   SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
StandardScaler()   
1   SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
passthrough   
2                                         passthrough  
StandardScaler()   
3                                         passthrough  
passthrough   
4   SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
StandardScaler()   
5   SelectFromModel(estimator=Lasso(alpha=0.1, ran...  
passthrough   
6                                         passthrough  
StandardScaler()   
7                                         passthrough  
passthrough   

                                               params  mean_test_score  
\
12  {'clf__max_depth': None, 'clf__n_estimators': ...         0.806935   

14  {'clf__max_depth': None, 'clf__n_estimators': ...         0.806935   

13  {'clf__max_depth': None, 'clf__n_estimators': ...         0.801380   

15  {'clf__max_depth': None, 'clf__n_estimators': ...         0.801380   

8   {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950   

9   {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950   
```

```
10  {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950

11  {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950

0   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

1   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

2   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

3   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

4   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164

5   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164

6   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164

7   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164


    std_test_score  rank_test_score
12        0.045989                1
14        0.045989                1
13        0.043601                3
15        0.043601                3
8         0.049943                5
9         0.049943                5
10        0.049943                5
11        0.049943                5
0         0.040118                9
1         0.040118                9
2         0.040118                9
3         0.040118                9
4         0.055740               13
5         0.055740               13
6         0.055740               13
7         0.055740               13
```

```
results[['split0_test_score',
       'split1_test_score', 'split2_test_score', 'split3_test_score',
       'split4_test_score', 'mean_test_score',
'std_test_score',]].sort_values('std_test_score')
```

```
    split0_test_score  split1_test_score  split2_test_score  \
0            0.676471           0.583333           0.666667
1            0.676471           0.583333           0.666667
2            0.676471           0.583333           0.666667
3            0.676471           0.583333           0.666667
```

13	0.788646	0.809524	0.730159
15	0.788646	0.809524	0.730159
12	0.788646	0.837302	0.730159
14	0.788646	0.837302	0.730159
8	0.747606	0.718254	0.753968
9	0.747606	0.718254	0.753968
10	0.747606	0.718254	0.753968
11	0.747606	0.718254	0.753968
4	0.617647	0.527778	0.658730
5	0.617647	0.527778	0.658730
6	0.617647	0.527778	0.658730
7	0.617647	0.527778	0.658730

	split3_test_score	split4_test_score	mean_test_score	std_test_score
0	0.583333	0.611111	0.624183	0.040118
1	0.583333	0.611111	0.624183	0.040118
2	0.583333	0.611111	0.624183	0.040118
3	0.583333	0.611111	0.624183	0.040118
13	0.813492	0.865079	0.801380	0.043601
15	0.813492	0.865079	0.801380	0.043601
12	0.813492	0.865079	0.806935	0.045989
14	0.813492	0.865079	0.806935	0.045989
8	0.769841	0.865079	0.770950	0.049943
9	0.769841	0.865079	0.770950	0.049943
10	0.769841	0.865079	0.770950	0.049943
11	0.769841	0.865079	0.770950	0.049943
4	0.527778	0.638889	0.594164	0.055740
5	0.527778	0.638889	0.594164	0.055740
6	0.527778	0.638889	0.594164	0.055740
7	0.527778	0.638889	0.594164	0.055740