Exercise 4. ML classifiers examples in healthcare

In this exercise, we will work with the Pima Indians Diabetes Database[1] to build different Machine Learning (ML) classifiers that act as a clinical predictive models to classify whether the patient suffers from Diabetes.

The dataset, with 768 instances, consists of the following independent variables or features:

- `Pregnancies`: Number of times pregnant
- `Glucose`: Plasma glucose concentration at 2 hours in an oral glucose tolerance test
- `BloodPressure`: Diastolic blood pressure (mmHg)
- `SkinThickness`: Triceps skin fold thickness (mm)
- `Insulin`: 2-Hour serum insulin (mu U/ml)
- `BMI`: Body mass index ($weight \in kg/height \in m^2$)
- `DiabetesPedigreeFunction`: Diabetes pedigree function
- `Age`: Age in years
- `Outcome`: Target dependent variable. 0: Non diabetes; 1: Diabetes

To access the dataset, press the link to the following webpage: https://www.kaggle.com/uciml/pima-indians-diabetes-database and download the dataset in csv format. The file is also available in the moodle folder of this exercise.

For this practical exercise, the following libraries need to be installed: Numpy, Pandas and Matplotlib (you should have them installed from previous exercises), Seaborn and Scikit-learn. To install the latter two, use the Anaconda graphical environment, or the Anaconda prompt by typing:

- Seaborn: `conda install seaborn`
- SciKit Learn: `conda install scikit-learn`

Note: It is likely for installing scikit learn that you need to add the conda-forge channel

Note: If you are not using conda environment, open your command prompt and navigate to the python installation folder and substitute the command '`conda`' by '`pyhton.exe -m pip`'.

First of all, the following libraries need to be imported:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Exploratory Data Analysis (EDA)

In this section, a first exploratory data analysis will be carried out to get acquainted with dataset and explore the values of the features.

[1] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, 'Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus', *Proc Annu Symp Comput Appl Med Care*, pp. 261–265, Nov. 1988.

Question 1.1 Read the dataset csv file into a Pandas data frame and describe the statistics of the dataset by executing the code below. What kind of information (1 paragraph 3-4 lines) can you provide from this statistical description?

```
path='...'# Set the path where you have stored the dataset csv file
file_name='diabetes.csv'
df= pd.read_csv(path+file_name)
df.describe()
```

Question 1.2 Does any of the features have outliers if we consider the IQR method[2]? To explore the existence of outliers, we can use a graphical method by depicting the boxplot diagram of each feature. Run the following command:

```
features=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction','Age']
target_feature=['Outcome']
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
figure=sns.boxplot(data=df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'DiabetesPedigreeFunction','Age']])
```

Question 1.3 However, with the previous code it's difficult for some features to detect graphically the existence of outliers. Run the following code and compare to the previous obtained plot:

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
df_features_normalized=pd.DataFrame(scaler.fit_transform(df[features]),columns=features)
```

What would you add to your answer to the previous question about the outliers? What exactly the code has done and for what other purposes change to this code might be needed?

Question 1.4 How are the target feature's values distributed? Run the following command:

```
df['Outcome'].value_counts()
```

What would you say about the target outcome, are we dealing with a balanced dataset? Therefore, what would you do to consider such target/outcome feature distribution when training the ML classifier?

2. Building the classifier

In this section, we will build two diabetes prediction models by using two ML classifier algorithms. To that aim we will choose a Logistic Regression and Random Forest classifier. You can find documentation of both classifiers here: sklearn.linear_model.LogisticRegression — scikit-learn 1.3.1 documentation and sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.1 documentation.

[2] '3.2 - Identifying Outliers: IQR Method | STAT 200', *PennState: Statistics Online Courses*. https://online.stat.psu.edu/stat200/lesson/3/3.2 (accessed Nov. 05, 2021).

Firstly, let's execute the following code:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

logreg_clf=LogisticRegression(random_state=42, class_weight='balanced')
rf_clf=RandomForestClassifier(max_depth=5, n_estimators=100, min_samples_split=8,
min_samples_leaf=3, max_features= 'auto', bootstrap= True, random_state=42)
```

Question 2.1 Best practices in data sciences recommend splitting the dataset into a train and a test set. There are several strategies, as we are going to see in this section.

One strategy is called holdout when you split up your dataset into a 'train' and 'test' set. The training set is what the model is trained on, and the test set is used to see how well that model performs on unseen data[3]. Therefore, split the dataset by using the scikit-learn function `train_test_split` (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html). Please consider the following parameters when using the function:

- train/test split ratio: 70/30
- random_state:42
- Stratify on the target outcome.

We need to create the variables X_train, y_train, X_test, and y_test to train and test the classifier, thus, execute the following instructions:

```
from sklearn.model_selection import train_test_split
y=df['Outcome'].copy()
X=df.drop('Outcome',axis=1)
X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.3, random_state=42,
stratify=y)
```

Could you explain the outcome of applying stratification when splitting the dataset. (hint: employ `value_counts()` method)

Question 2.2

Now, it is time to fit the classifiers with the data and see how they perform by analysing different classification metrics results.

Thus, let's start by fitting the Logistic regression classifier with the data. For that purpose, it is helpful to ensure that all variables' values are within the same range, thus, a preprocessing step of standardization must be carried by using the scikit-learn function StandardScaler method (sklearn.preprocessing.StandardScaler — scikit-learn 1.3.1 documentation).

Scikit-learn package allows to build pipelines to concatenate the pre-processing and classification steps in data science projects. In this exercise, we will use these pipelines thus, execute the following code:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

[3] Hold-out vs. Cross-validation in Machine Learning | by Jaz Allibhai | Medium

```
std_scaler=StandardScaler()
logreg_clf_pipe=Pipeline([(‘scaler’, std_scaler), (‘clf’, logreg_clf)])
logreg_clf_pipe.fit(X_train,np.ravel(y_train))
```

Now, that we have fitted the classifier with the train data, it’s time to evaluate its performance with the test set. This evaluation will consider different metrics and methods:

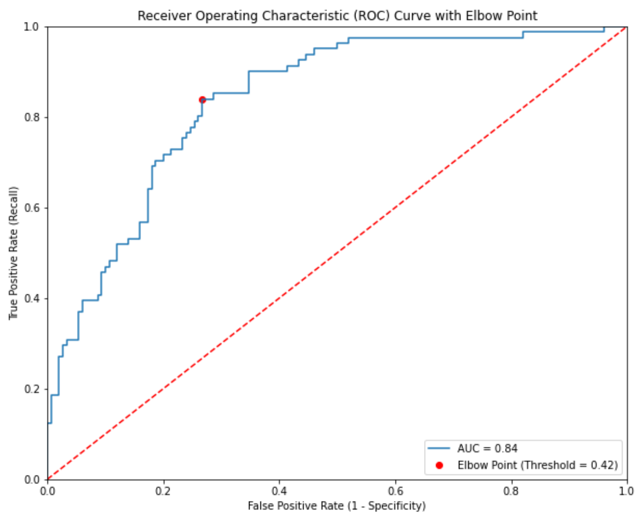
1. First, use the metrics as accuracy, balanced accuracy, sensitivity/recall and precision. What can you say about the results obtained?

```
from sklearn.metrics import accuracy_score, recall_score, precision_score
y_pred_test=logreg_clf_pipe.predict(X_test)
print('Test accuracy', accuracy_score(y_test, y_pred_test))
print('Test sensitivity/recall', recall_score(y_test, y_pred_test))
print('Test precision', precision_score(y_test, y_pred_test))
```

2. Second, let’s see visually through a confusion matrix how many classifications have been made correctly and incorrectly. How can you relate the results shown with the metrics figures obtained previously?

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, ypred_test)
df_cm = pd.DataFrame(cm, index= logreg_clf_pipe.classes_, columns= logreg_clf_pipe.classes_)
sns.heatmap(df_cm, annot=True, cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

3. Third, another way to assess the classification performance is the receiver operating characteristics curve and its area under the curve. How can you interpret the figure shown below? Describe some findings about the area under the curve and the elbow point.



3. Comparing several classifiers

Usually, data scientists use several ML classifiers to obtain the best algorithm that fits with the problem (classification/ regression) they are trying to solve. As said, in this exercise we will also use the Random Forest classifier. Use the code above to obtain the results concerning metrics, confusion matrix and auc_roc, this time, with the Random Forest.

Question 3.1 What classifier performs better? Why?

Question 3.2 You might notice when we defined the Random Forest classifier (at the start of Question 2), there was no parameter that handled the target feature imbalance. Create another random forest estimator by running the following code and evaluate their performance as you have done previously. What differences do you find compared to the other Random Forest?

```
rf_clf_balanced=RandomForestClassifier(max_depth=5, n_estimators=100, min_samples_split=8,
min_samples_leaf=3, max_features= 'auto', bootstrap= True, class_weight='balanced',
random_state=42)
```

4. Cross-validation approach

In addition to the holdout strategy, which involves splitting the dataset into a training set and a test set, there are alternative methods for training and evaluating a classifier such as cross-validation. Unlike the holdout method, which uses a fixed test set for evaluation, cross-validation offers a more robust way to assess model performance. It does this by creating multiple smaller sets, known as 'folds,' from the original training data. The model is then trained and evaluated multiple times, each time using a different fold as a validation set while the remaining folds serve as the training set.

In this section, we are going to evaluate our model Log Reg with this cross-validation approach. For that purpose, run the following code:

```
from sklearn.model_selection import cross_validate, StratifiedKFold
# Initialize StratifiedKFold
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# Define scoring metrics
scoring_metrics = ['accuracy', 'precision', 'recall']
# Perform 5-fold stratified cross-validation with multiple metrics
cv_results = cross_validate(logreg_clf_pipe, X_train, y_train, cv=stratified_kfold,
scoring=scoring_metrics)

# Output the cross-validation scores for each metric
for metric, scores in cv_results.items():
    if 'test_' in metric:  # We're interested in test scores
        print(f"{metric}: {scores}")
        print(f"Mean {metric}: {np.mean(scores)}")
        print(f"Standard Deviation of {metric}: {np.std(scores)}")
```

Compare the results obtained with this approach to what you obtained with the holdout approach. Which method gave a more reliable performance estimate? How did the performance metrics vary across different folds in cross-validation?

Note: If you are curious, you can also apply cross-validation to the Random Forest classifiers and check how the performance measure is affected.