Programming assignment 1: introductory tour and decision trees

In this assignment, you will take a quick guided tour of the scikit-learn library, one of the most widely used machine learning libraries in Python. We will particularly focus on decision tree learning for classification and regression.

Work in groups of two or three and solve the tasks described below. Even though most of the code is given, try to understand what is going on in all the steps. Ask the lab assistant if you are confused about anything.

There are three compulsory tasks of the assignment, where the main focus is on using scikit-learn for training and evaluating machine learning models. **You need to solve these parts in order to pass this assignment.** There is a fourth part where the focus is on implementing a decision tree model for regression. A solution to the fourth task is not required for passing the assignment, but **to get a score higher than the minimal score for passing, you need to solve the fourth part at least partly.**

You should structure your solution as a **notebook**. You have two choices: either you work on your own machine using a Jupyter notebook. (You can work with Jupyter directly or use the notebook support of popular IDEs such as PyCharm or VS Code.) Alternatively, you use the Colab service, which works in a similar fashion. (Here is a document about Colab we prepared for another course. You can ignore the GPU-related parts for now.) In case you use Colab, you can submit your solution either by downloading the notebook (`.ipynb`) or by providing a link to the notebook. (In that case, make sure you allow access to the notebook.)

When you work with the notebook, you will enter your solution in the code cells. In addition, make sure that you add text cells that explain briefly what you are doing, and answer any questions you find in the assignment instructions. You should think of the notebook as a whole as a technical report.

This is a **group assignment** and your work should be carried out in a group of 2 or 3 people. If you don't have a group partner, please ask around in the Discussion tool in Canvas. Please include **the names of the group members** in your submission.

Submit your solution through the Canvas website. Remember to join a Canvas group before submitting.

Deadline: January 23

Didactic purpose of this assignment:

- getting a feel for the workflow of machine learning in Python;
- understanding decision tree learning algorithms for classification and regression;
- introducing the notions of overfitting and underfitting.

References

- Lecture 1 on the basic steps of machine learning, and decision tree learning.
- Extra reading material on decision trees.
- scikit-learn documentation.
- Pandas reference documentation.

Task 0: Making sure that you have the required libraries

(You can skip this part if you are using Colab, which comes with scikit-learn and Pandas enabled by default.)

If you are running on your own machine, the scikit-learn and Pandas libraries are **not** installed by default. You will have to install them on your own machine or in your home directory on the lab machine.

If you're new to Python, we recommend that you use the Anaconda Python distribution, which includes scikit-learn, Pandas, and several other useful libraries. If you don't use Anaconda, please refer to the official installation instructions for scikit-learn and Pandas.

After installing, verify your installation by starting `python` and typing `import sklearn`. You shouldn't see an error message at this point.

Task 1: A classification example: fetal heart condition diagnosis

The UCI Machine Learning Repository contains several datasets that can be used to investigate different machine learning algorithms. In this exercise, we'll use a dataset of fetal heart diagnosis. The dataset contains measurements from about 2,600 fetuses. This is a classification task, where our task is to predict a diagnosis type following the FIGO Intrapartum Fetal Monitoring Guidelines: *normal*, *suspicious*, or *pathological*.

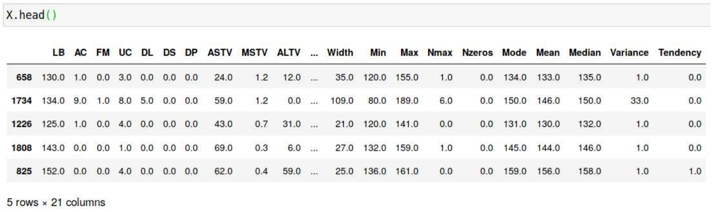
Step 1. Reading the data

This file contains the data that we will use. This file contains the same data as in the public distribution, except that we converted from Excel to CSV. Download the file and save it in a working directory.

Open your favorite editor or a Jupyter notebook. To read the CSV file, it is probably easiest to use the Pandas library. Here is a code snippet that carries out the relevant steps:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the CSV file.
data = pd.read_csv(LOCATION_OF_THE_FILE, skiprows=1)

# Select the relevant numerical columns.
selected_cols = ['LB', 'AC', 'FM', 'UC', 'DL', 'DS', 'DP', 'ASTV', 'MSTV', 'ALTV',
                 'MLTV', 'Width', 'Min', 'Max', 'Nmax', 'Nzeros', 'Mode', 'Mean',
                 'Median', 'Variance', 'Tendency', 'NSP']
data = data[selected_cols].dropna()

# Shuffle the dataset.
data_shuffled = data.sample(frac=1.0, random_state=0)

# Split into input part X and output part Y.
X = data_shuffled.drop('NSP', axis=1)

# Map the diagnosis code to a human-readable label.
def to_label(y):
    return [None, 'normal', 'suspect', 'pathologic'][(int(y))]

Y = data_shuffled['NSP'].apply(to_label)

# Partition the data into training and test sets.
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.2, random_state=0)
```

You can call `X.head()` to take a peek at the data. The input part consists of 21 numerical features. The official page for the dataset includes brief descriptions of all the features.



```
X.head()
```

```
        LB   AC   FM   UC   DL   DS   DP  ASTV  MSTV  ALTV  ...  Width    Min    Max  Nmax  Nzeros   Mode   Mean  Median  Variance  Tendency
 658 130.0  1.0  0.0  3.0  0.0  0.0  0.0  24.0   1.2  12.0  ...   35.0  120.0  155.0   1.0     0.0  134.0  133.0   135.0       1.0       0.0
1734 134.0  9.0  1.0  8.0  5.0  0.0  0.0  59.0   1.2   0.0  ...  109.0   80.0  189.0   6.0     0.0  150.0  146.0   150.0      33.0       0.0
1226 125.0  1.0  0.0  4.0  0.0  0.0  0.0  43.0   0.7  31.0  ...   21.0  120.0  141.0   0.0     0.0  131.0  130.0   132.0       1.0       0.0
1808 143.0  0.0  0.0  1.0  0.0  0.0  0.0  69.0   0.3   6.0  ...   27.0  132.0  159.0   1.0     0.0  145.0  144.0   146.0       1.0       0.0
 825 152.0  0.0  0.0  4.0  0.0  0.0  0.0  62.0   0.4  59.0  ...   25.0  136.0  161.0   0.0     0.0  159.0  156.0   158.0       1.0       1.0

5 rows × 21 columns
```

Step 2. Training the baseline classifier

We can now start to investigate different classifiers.

The `DummyClassifier` is a simple classifier that does not make use of the features: it just returns the most common label in the training set, in this case `Spondylolisthesis`. The purpose of using such a stupid classifier is as a *baseline*: a simple classifier that we can try before we move on to more complex classifiers.

```
from sklearn.dummy import DummyClassifier

clf = DummyClassifier(strategy=’most_frequent’)
```

To get an idea of how well our simple classifier works, we carry out a **cross-validation** over the training set and compute the classification accuracy on each fold.

```
from sklearn.model_selection import cross_val_score

cross_val_score(clf, Xtrain, Ytrain)
```

The result is a NumPy array that contains the accuracies on the different folds in the cross-validation. Aggregate these scores so that you get a single score that you can use to compare different classifiers.

Step 3. Trying out some different classifiers

Replace the `DummyClassifier` with some more meaningful classifier and run the cross-validation again. Try out a few classifiers and see how much you can improve the cross-validation accuracy. Remember, the accuracy is defined as the proportion of correctly classified instances, and we want this value to be **high**.

Here are some possible options:

Tree-based classifiers:

- `sklearn.tree.DecisionTreeClassifier`
- `sklearn.ensemble.RandomForestClassifier`
- `sklearn.ensemble.GradientBoostingClassifier`

Linear classifiers:

- `sklearn.linear_model.Perceptron`
- `sklearn.linear_model.LogisticRegression`
- `sklearn.svm.LinearSVC`

Neural network classifier (will take longer time to train):

- `sklearn.neural_network.MLPClassifier`

You may also try to tune the **hyperparameters** of the various classifiers to improve the performance. For instance, the decision tree classifier has a parameter that sets the maximum depth, and in the neural network classifier you can control the number of layers and the number of neurons in each layer.

Step 4. Final evaluation

When you have found a classifier that gives a high accuracy in the cross-validation evaluation, train it on the whole training set and evaluate it on the held-out test set.

```
from sklearn.metrics import accuracy_score

clf.fit(Xtrain, Ytrain)
Yguess = clf.predict(Xtest)
print(accuracy_score(Ytest, Yguess))
```

For the report. In your submitted report, please include a description of the classifier you selected and report its accuracy. (At this point, we are of course not asking you to describe internal workings of various machine learning models that we will cover in detail at later points during the course, but you are of course free to read about them if you're interested.)

Task 2: Decision trees for classification

Download the code that was shown during the lecture and use the defined class `TreeClassifier` as your classifier in an experiment similar to those in Task 1, using the same dataset. (Alternatively, you can create a new notebook and copy the code from this page.) Tune the hyperparameter `max_depth` to get the best cross-validation performance, and then evaluate the classifier on the test set.

For the report. In your submitted report, please mention what value of `max_depth` you selected and what accuracy you got.

For illustration, let's also draw a tree. Set `max_depth` to a reasonably small value (not necessarily the one you selected above) and then call `draw_tree` to visualize the learned decision tree. Include this tree in your report.

Task 3: A regression example: predicting apartment prices

Here is another dataset. This dataset was created by Sberbank and contains some statistics from the Russian real estate market. Here is the Kaggle page where you can find the original data.

Since we will just be able to handle numerical features and not symbolic ones, we'll need with a simplified version of the dataset. So we'll just select 9 of the columns in the dataset. The goal is to predict the price of an apartment, given numerical information such as the number of rooms, the size of the apartment in square meters, the floor, etc. Our approach will be similar to what we did in the classification example: load the data, find a suitable model using cross-validation over the training set, and finally evaluate on the held-out test data.

The following code snippet will carry out the basic reading and preprocessing of the data.

```
# Read the CSV file using Pandas.
alldata = pd.read_csv(LOCATION_OF_YOUR_FILE)

# Convert the timestamp string to an integer representing the year.
def get_year(timestamp):
    return int(timestamp[:4])
alldata['year'] = alldata.timestamp.apply(get_year)

# Select the 9 input columns and the output column.
selected_columns = ['price_doc', 'year', 'full_sq', 'life_sq', 'floor', 'num_room', 'kitch_sq', 'full_all']
alldata = alldata[selected_columns]
alldata = alldata.dropna()

# Shuffle.
alldata_shuffled = alldata.sample(frac=1.0, random_state=0)

# Separate the input and output columns.
X = alldata_shuffled.drop('price_doc', axis=1)
# For the output, we'll use the log of the sales price.
Y = alldata_shuffled['price_doc'].apply(np.log)

# Split into training and test sets.
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.2, random_state=0)
```

We train a baseline dummy regressor (which always predicts the same value) and evaluate it in a cross-validation setting.

This example looks quite similar to the classification example above. The main differences are (a) that we are predicting numerical values, not symbolic values; (b) that we are evaluating using the *mean squared error* metric, not the accuracy metric that we used to evaluate the classifiers.

```
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import cross_validate
m1 = DummyRegressor()
cross_validate(m1, Xtrain, Ytrain, scoring='neg_mean_squared_error')
```

Replace the dummy regressor with something more meaningful and iterate until you cannot improve the performance. Please note that the `cross_validate` function returns the *negative* mean squared error.

Some possible regression models that you can try:

- sklearn.linear_model.LinearRegression
- sklearn.linear_model.Ridge
- sklearn.linear_model.Lasso
- sklearn.tree.DecisionTreeRegressor
- sklearn.ensemble.RandomForestRegressor
- sklearn.ensemble.GradientBoostingRegressor
- sklearn.neural_network.MLPRegressor

Finally, train on the full training set and evaluate on the held-out test set:

```
from sklearn.metrics import mean_squared_error

regr.fit(Xtrain, Ytrain)
mean_squared_error(Ytest, regr.predict(Xtest))
```

For the report. In your submitted report, please include a description of the regression model you selected and report its evaluation score.

Task 4: Decision trees for regression (required only if you're aiming for a high grade)

For the final task, we'll implement a class `TreeRegressor` for regression with decision trees.

Before you start, make sure that you understand the implementation of the class `TreeClassifier`.

Step 1. Implementing the regression model

It is probably a good idea to structure this in a similar way to the `TreeClassifier`, just adapted for regression instead. This means that you should probably write a new subclass that inherits from the same abstract superclass `DecisionTree` that the classifier used. Probably it's a good idea to replace `ClassifierMixin` with `RegressorMixin` (this just replaces the default type of evaluation metric).

In your regressor subclass, implement the same methods that you can see in `TreeClassifier`. In particular, you'll need to define:

- what it means for a set of output values to be "homogeneous": instead of checking whether all values are identical, you should probably compare the variance to some threshold. (This threshold can either be a small value that you hard-code, or a user-specified hyperparameter.) The NumPy function `np.var` is probably useful here.
- how to compute the default output value: you should return the mean instead of the most common value. Here, you can use `np.mean`.
- how to select split that leads to the most homogeneous subsets.

The third of these items is the most challenging. In the cases of regression, we use the variance of a set of values to measure its homogeneity. The homogeneity criterion most typically used in decision tree regression (including scikit-learn's implementation) is the following, called "variance reduction":

$$\text{variance reduction} = V(S) - \frac{n_H}{n}V(S_H) - \frac{n_L}{n}V(S_L)$$

Here, *V(S)* means the variance of the full set of values, *V(S_H)* means the variance of the higher part, n_H means the size of the higher part, etc.

To implement this splitting function, you can get some inspiration from the corresponding function for the classifier. The basic structure will be very similar. But instead of counting the number of instances of each class (using the `Counter`), you'll have to compute variances instead.

Hint. Computing the variances from scratch at each possible threshold, for instance by calling `np.var`, will be time-consuming if the dataset is large. (What is the time complexity?) It's better to rely on the formula

$$V([x_1, \ldots, x_n]) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2$$

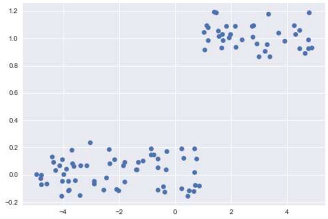
When you go through the possible thresholds, keep track of the sum and sum of squares for the lower and upper partitions.

Step 2. Sanity check

The following function will generate a small number of training examples for a simple regression task with one input variable.

```
def make_some_data(n):
    x = np.random.uniform(-5, 5, size=n)
    Y = (x > 1) + 0.1*np.random.normal(size=n)
    X = x.reshape(n, 1) # X needs to be a 2-dimensional matrix
    return X, Y
```

If we generate such a dataset and plot it, the result will look something like the figure below. The *x* axis represents the input and the *y* axis the output.



For the report. If you consider the data-generating function, what kind of decision tree would we want to describe this data?

Train your decision tree regressor algorithm on a small dataset generated by the function above, and then draw the tree. Select the tree depth according to your common sense. Does the result make sense? What happens if we allow the tree depth to be a large number?

Step 3. Predicting apartment prices using decision tree regression

Train and evaluate a decision tree regression model for the Russian apartment price prediction example.

For the report. In your submitted report, please describe what tree depth you used and the evaluation score you got on the test set.

Step 4. Underfitting and overfitting

For the apartment price prediction task, draw a plot that shows the evaluation score on the *training set* and on the *test set* for different values of `max_depth`, ranging from 0 to 12. (It's probably easiest if you implement a function to draw this plot, but it's also OK if you draw the plot by hand.)

Clarification. When computing the evaluation scores for the training set, do not use cross-validation this time! We are investigating overfitting now.

For the report. Please include this plot in the report, and comment on the differences between the two curves.