# Theoretical foundations of Machine Learning

TP 3

## 1 Perceptron and K-NN

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

We will work with a dataset about wines, which can be downloaded from pamplemousse. Be careful not to open the CSV file with Excel and save it afterward, as Excel may alter the file. We load it as a Pandas DataFrame and display its first five rows as follows:

```
data = pd.read_csv('wine_dataset.csv')
data.head()
```

**Question 1.** Examine the `DataFrame data`. How many examples and how many variables does the dataset contain?

We aim to predict the type of wine (red or white) based on its chemical properties. The variable `quality` is not a chemical property but a subjective rating given by wine tasters; it is not relevant for our problem, so we will remove it. Pandas allows us to manipulate (in this case, delete) columns by referring to their names.

```
data = data.drop(columns=['quality'])
```

Before building predictors, we want to select a small number of explanatory variables (say four): those that seem most promising for predicting wine type. For example, for the variable `fixed_acidity`, we can visualize its distribution for each wine type as follows:

```
g = sns.FacetGrid(data=data, hue='style')
g.map(sns.histplot, 'fixed_acidity').add_legend()
plt.show()
```

**Question 2.** Produce a figure that contains the visualizations for each explanatory variable. You can use a `for` loop, as well as `data.columns`, which provides the list of variable names.

**Question 3.** Based on the generated figures, choose four variables that appear to be the most promising for predicting the type of wine. Create a `DataFrame data_reduced` containing only these selected variables as well as the target variable. You can use the following syntax.

```
data_reduced = data[['column_name1','column_name2']]
```

We will now split the dataset into training and test samples. For this, we can use a dedicated function provided by `scikit-learn`, which also takes care of shuffling the order of the examples.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_reduced.drop(columns='style'),
    data_reduced['style'])
```

**Question 4.** Observe the obtained samples and determine the ratio of their sizes.

We will now train a logistic regression model.

```
from sklearn.linear_model import LogisticRegression

f = LogisticRegression()
f.fit(X_train, y_train)
```

**Question 5.** Display the training and test errors of the constructed predictor. Remember to use the `f.score()` function.

**Question 6.** Also train a Perceptron predictor, which can be loaded using the following command

```
from sklearn.linear_model import Perceptron
```

as well as $k$-NN predictors (for different values of $k$), which can be loaded using

```
from sklearn.neighbors import KNeighborsClassifier
```

Compare the results.

We will now look at, for example for predictor `f`, the *confusion matrix* computed on the test sample.

```
pd.crosstab(y_test, f.predict(X_test))
```

**Question 7.** Explain what the confusion matrix represents.

By examining the confusion matrix, we observe that the test sample contains significantly more white wines than red wines. We can therefore deduce that, when evaluating the performance of the predictors on the test sample, the accuracy on white wines has a greater influence on the overall score than the accuracy on red wines.

**Question 8.** From `X_test` and `y_test`, construct a test subsample `X_test_` and `y_test_` that contains an equal number of red and white wines. You may use the resample function, whose documentation you can consult, and which can be imported using the following command:

```
from sklearn.utils import resample
```

**Question 9.** Evaluate the performance of the constructed predictors on this subsample. Comment on the results.

**Question 10.** Similarly, create a balanced training subsample `X_train_` and `y_train_`. Train the predictors on this subsample and observe their scores. Comment.

**Question 11.** Train the estimators using the full set of initial explanatory variables. Comment on the results.

## 2 $k$-NN and cross-validation

```
import numpy as np
import matplotlib.pyplot as plt
```

We consider a character recognition problem. The dataset contains images of handwritten digits, along with their corresponding labels. Our goal is to build a predictor that can recognize the digit from the image  this is therefore a classification problem. We will now load the dataset.

```
from sklearn import datasets
digits = datasets.load_digits()
X = digits.data
y = digits.target

print(X[1])
```

We can see that each entry, for example `X[1]` above, is a 1-dimensional `array` of size 64. It represents an image of size 8 x 8 pixels. Each component is an integer between 0 and 16 and represents the grayscale intensity of the corresponding pixel. For each entry, we can use the `.reshape()` function to convert it into a 2-dimensional array of size 8 x 8.

```
print(X[1].reshape(8,8))
```

This allows us to visualize the corresponding image using the `plt.imshow()` function.

```
plt.figure()
plt.imshow(X[1].reshape(8,8),cmap=plt.cm.gray_r)
plt.show()
```

**Question 12.** Are the different classes $(0, 1, ..., 8, 9)$ present in approximately equal quantities in the dataset?

We will build $k$-NN predictors. The predictions made by these algorithms are slow to compute when the training set is large. Therefore, we will try to limit the size of the training set.

**Question 13.** Partition the dataset into two samples: a sample $(X_{\text{cv}}, y_{\text{cv}})$ of size 200, which will be used for cross-validation, and a test sample $(X_{\text{test}}, y_{\text{test}})$, which will be used for the final evaluation. You may use the `train_test_split` function seen in the previous lab session, in which the optional argument `train_size` can be specified.
We can now perform cross-validation on the sample $(X_{\text{cv}}, y_{\text{cv}})$. For example, we will consider the $k$-NN algorithm with $k = 5$, and perform 7-fold cross-validation. We will use the `cross_val_score` function provided by `scikit-learn`.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
knn = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(knn,X_cv,y_cv,cv=7)
```

**Question 14.** What does the variable `scores` defined above contain? Deduce from it the validation score of the 7-fold cross-validation for the 5-NN algorithm.
For a range of values of the hyperparameter $k$ (for example $k \in \{1, \dots, 20\}$), the training and validation scores from 7-fold cross-validation can be computed as follows:

```
from sklearn.model_selection import validation_curve
k_range = range(1, 21)
train_scores, valid_scores = validation_curve(
    KNeighborsClassifier(), X_cv, y_cv, "n_neighbors", k_range, cv=7)
```

**Question 15.** Plot the corresponding validation curves. Is it necessary to try other values of $k$? Define a variable `k_best` equal to the best value of the hyperparameter $k$ based on the above results.

We now want to plot the learning curve for $k = k_{\text{best}}$, corresponding to training set sizes from 10 to 150, in increments of 5. The following code gives the corresponding 7-fold cross-validation scores:

```
from sklearn.model_selection import learning_curve
train_size_range = range(10,151,5)
train_sizes, train_scores, valid_scores = learning_curve(KNeighborsClassifier(n_neighbors=k_best),
X_cv, y_cv,train_sizes=train_size_range,cv=7)
```

**Question 16.** Plot the corresponding learning curve. Would it be interesting to use larger training samples? If so, return to Question 2 using a larger training set.

**Question 17.** Plot a learning curve where the score is obtained using simple validation (i.e., without cross-validation). Explain the advantage of using cross-validation in this context.

**Question 18.** Train the $k$-NN predictor (with $k = k_{\text{best}}$) using $(X_{\text{cv}}, y_{\text{cv}})$ as the training set. Compute its test score using the sample $(X_{\text{test}}, y_{\text{test}})$. Also display the confusion matrix on the test set. What is the most frequent classification error?

We now wish to add new examples to the dataset by modifying existing ones, in the hope of obtaining better results.

**Question 19.** Write a function `shift_image` that takes as arguments: (1) a 2-dimensional array named `image` of size $8 \times 8$ (representing an image), and (2) a 1-dimensional array named `direction` of size 2, and returns the image translated by the vector of coordinates given by `direction`. The image content should be cropped accordingly on the sides. Verify that the function works correctly by visualizing a few translated images.

**Question 20.** Augment the dataset by translating all examples with each of the following vectors:

$$(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1).$$

Then, build a $k$-NN predictor by choosing the hyperparameter $k$ through cross-validation, and evaluate its score on a test sample. Compare the resulting predictor with the one from Question 7. You may initially use a cross-validation sample of size 400, and then decide whether it is worthwhile to increase the sample size.