

# Introduction to classification

---

## Objective

Learn a predictive model of wines quality using measured physical-chemical characteristics. The dataset can be download online<sup>1</sup>. More details on the features are available at <http://archive.ics.uci.edu/ml/datasets/Wine+Quality>

The input variables are the physical-chemical features and the label (output, class) is the wine category (good or bad quality).

**Preamble** Activate Python virtual environment and launch your notebook (or spyder) by running in the terminal

```
>>> source /opt/venv/iti-data/bin/activate  
>>> jupyter-notebook&  
>>> spyder&
```

To get the help of a function, see an example below.

```
from sklearn.neighbors import KNeighborsClassifier  
# to get the online help, type:  
>>> ?KNeighborsClassifier
```

## 1 Data analysis

1. Load the data and show its summary. How many samples and inputs do we have?

```
import pandas as pd  
import numpy as np  
  
link = "http://archive.ics.uci.edu/ml/machine-learning-databases/wine-  
quality/winequality-white.csv"  
df = pd.read_csv(link, header="infer", delimiter=";")  
print("\n===== Dataset summary =====\n")  
df.info()  
print("\n===== A few first samples =====\n")  
print(df.head())
```

2. Form the arrays  $X \in \mathbb{R}^{N \times d}$  of the input variables and  $Y \in \mathbb{R}^N$  the output. What are the wine qualities and the related number of samples ?

```
X = df.drop("quality", axis=1) #we drop the column "quality"  
Y = df["quality"]  
print("\n===== Wine Qualities =====\n")  
print(Y.value_counts())
```

3. To form a binary classification problem, we group the data by quality level.

```
# bad wine (y=0) : quality <= 5 and good quality (y= 1) otherwise  
Y = [0 if val <=5 else 1 for val in Y]
```

Check the number of samples per class (we have now classes 0 and 1).

<sup>1</sup><http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv>

4. Perform a statistical analysis (mean, variance, correlation ...) of the input variables.  
Comments on the results.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure()
ax = plt.gca()
sns.boxplot(data=X, orient="v", palette="Set1", width=1.5, notch=True)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.figure()
corr = X.corr()
sns.heatmap(corr)
```

## 2 Classification

### 2.1 Data split

1. Randomly split-up  $(X, Y)$  (keep the proportion of the labels) in three subsets  $\mathcal{D}_a = (X_a, Y_a)$ ,  $\mathcal{D}_v = (X_v, Y_v)$  and  $\mathcal{D}_t = (X_t, Y_t)$ , respectively the training, validation and test set.

```
from sklearn.model_selection import train_test_split

Xa, Xt, Ya, Yt = train_test_split(X, Y, shuffle=True, test_size=1/3,
                                    stratify=Y)
Xa, Xv, Ya, Yv = train_test_split(Xa, Ya, shuffle=True, test_size=0.5,
                                    stratify=Ya)
```

2. Discuss why do we need to keep the class proportion and shuffle the data.

### 2.2 $k$ nearest neighbor ( $k$ -NN) classification

$k$ -NN uses the samples in  $(X_a, y_a)$  as the reference data. Given any sample  $x_i$ , the predicted label is the majority label of its  $k \in \mathbb{N}$  neighbors in  $(X_a, y_a)$ . The neighbors are the ones close to  $x_i$  in terms of the Euclidean distance.  $d(x_i - x_j)^2 = \|x_i - x_j\|^2 = (x_i - x_j)^\top (x_i - x_j)$ ,  $x_j \in \mathcal{D}_a$ .

1. As a starter let run  $k$ -NN with  $k = 3$  and evaluate its performance on  $\mathcal{D}_v$  by the error rate defined as

$$\text{error rate} = \frac{1}{N} \sum_{i \in \mathcal{D}} y_i \neq \hat{y}_i$$

with  $N$ , the size of set  $\mathcal{D}$ ,  $\hat{y}$  and  $y$  respectively the predicted and true label.

```
from sklearn.neighbors import KNeighborsClassifier

# Fit the model on (Xa, Ya)
k = 3
clf = KNeighborsClassifier(n_neighbors = k)
clf.fit(Xa, Ya)
```

```
# Predict the labels of samples in Xv
Ypred_v = clf.predict(Xv)

# evaluate classification error rate
from sklearn.metrics import accuracy_score
error_v = 1 - accuracy_score(Yv, Ypred_v)
```

- Inspiring from that, train the  $k$ -nn classifier for different values of  $k \in \mathbb{N}$  (in the range  $[1, 40]$ ) and evaluate its performance either on training set and validation set. Plot the training and validation error curves as a function of  $k$ . From the plots, point out the overfitting issue.

```
# some hints
k_vector = np.arange(1, 37, 2) #define a vector of k=1, 3, 5, ...
error_train = np.empty(k_vector.shape)
error_val = np.empty(k_vector.shape)

for ind, k in enumerate(k_vector):
    #fit with k
    clf = KNeighborsClassifier(n_neighbors = k)
    clf.fit(Xa, Ya)

    # predict and evaluate on training and validation sets
    Ypred_train = ...
    error_train[ind] = ...

    ...
```

- How to choose the appropriate value of  $k$ ? Select the best value  $k^*$  of  $k$  based on the validation error.

```
# some hints: get the min error and related k-value
err_min, ind_opt = error_val.min(), error_val.argmax()
k_star = k_vector[ind_opt]
```

- For the selected  $k^*$ , what is the error rate on the data test? Discuss the obtained results.

### 2.3 Normalize or not normalize the data ?

- So far, we have used the raw inputs without any normalization. To account for different input scales we normalize the data by removing the mean value of each feature and scale it by dividing by the standard deviation. Comment and explain the following codes. Is the applied normalization to the validation set sound?

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler(with_mean=True, with_std=True)
sc = sc.fit(Xa)

Xa_n = sc.transform(Xa)
Xv_n = sc.transform(Xv)
```

2. Replicate the experiments from section 2.2 with the normalized data and compare the achieved performances with the one of section 2.2. Drawn the conclusions ?
3. How to make the trained models less sensitive to the data split ?