Introduction to Machine Learning / Labwork 1

A first classification model: the k-NN

Maxime Ossonce maxime.ossonce@esme.fr

The purpose of this labwork is to learn a predictive model of wine quality using the k-nearest neighbors (k-NN) algorithm. The dataset has been downloaded as a homework (see labwork 0 on moodle). The needed packages have been installed as well.

Remark: for better readability of the code, imports are added when needed, even though this is not considered a good practice (imports are generally done at the beginning of the file).

Notation

The ith sample ($i \in 1,\dots,N$) is noted x^i. It is a vector of $\mathbb{R}^d$: $x^i = (x^i_1, \cdots, x^i_d)$. The matrix X (of size $N \times d$, see (1)) is a matrix with row i being the sample x^i: $X_{ij} = x^i_j$. The label attached to x^i is noted y^i.

$$X = \begin{pmatrix} x^1_1 & \cdots & x^1_j & \cdots & x^1_d \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x^i_1 & \cdots & x^i_j & \cdots & x^i_d \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x^N_j & \cdots & x^N_j & \cdots & x^N_d \end{pmatrix} \qquad y = \begin{pmatrix} y^1 \\ \vdots \\ y^i \\ \vdots \\ y^N \end{pmatrix}. \tag{1}$$

k-NN

The k-NN is a regression/classification algorithm. It is considered a *lazy* training algorithm: no parameter is learned. During the training phase, samples and labels $\{(x^i, y^i)\}_{i=1\cdots N}$ are simply stored. For the evaluation of a new sample x, the k nearest training samples are considered and the predicted label y for x is the one that has the majority amongst the ones attached to the k nearest neighbors.

Data analysis

1. Load the data and show its summary. How many samples do we have? How many features do we have? Name a few.

wine_db_knn.py

```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

# download the csv file first!
csv_file = 'winequality-white.csv'
df = pd.read_csv(csv_file, header='infer', delimiter=';')
print('=== Data summary ===')
df.info()

print('=== First samples ===')
print(df.head().to_string())
```

The loaded dataframe contains the samples and the labels, that we need to cut into the matrix $X \in \mathbb{R}^{N \times d}$, N being the number of samples and d being the number of variables (features) and the vector $y \in \mathcal{R}^N$. Moreover, the features *residual sugar* and *free sulfur dioxide* are known to not predict the wine quality: they are dropped to prevent overfitting.

2. What is the distribution of the wine qualities?

wine_db_knn.py

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

X = X.drop(['residual sugar', 'free sulfur dioxide', 'pH'], axis=1)
```

We aim to perform a binary classification (good wine *or* bad wine). We consider the good wines those with quality greater or equal to 6. The data is grouped between the good wines and the bad wines.

3. What would happen if the discriminatory value of wine quality (6) was, instead, chosen to be 5?

wine_db_knn.py

```
bins = [0, 5, 10]
labels = ['bad', 'good']
y = pd.cut(y, bins=bins, labels=labels)

print(y.value_counts())
```

4. Plot the correlation matrix and comment on the result.

wine_db_knn.py

```
# create a figure
fig1, ax1 = plt.subplots(1)
```

```
33
34 # draw the box plots
35 sns.boxplot(data=X, orient="v")
36 ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
37
38 # create a new figure
39 fig2, ax2 = plt.subplots(1)
40
41 # compute the matrix correlation
42 corr = X.corr()
43
44 # draw the correlation matrix
45 sns.heatmap(corr)
46
47 # show the figures
48 fig1.show()
49 fig2.show()
```

Classification

Prior to any model training, we have to split the dataset in three datasets:

- the training set, that is used to train the model,
- the validation set (generally a subset of the training set), to evaluate the performances of a given model and choose the proper hyperparameter (*e.g.* k, the numbers of neighbors in the k-NN alogrithm),
- the test set, to evaluate the model once the it has been selected.

The test set must be used only once. If it is used to tune an hyperparameter (or worse, to train the model), the given results will not be legit.

5. What does the argument `stratify` achieve?

wine_db_knn.py

```
50 # split the dataset into train and test
51 shuffle = True
52
53 Xa, Xt, ya, yt = train_test_split(X, y, test_size=0.3, stratify=y, shuffle=shuffle)
54
55 # split the trainset into train and validation
56 Xa, Xv, ya, yv = train_test_split(Xa, ya, test_size=0.5, stratify=ya, shuffle=shuffle)
```

The k-NN model keeps in memory all samples x^i from $\mathcal{X}_a$, the training set and their corresponding labels y^i. Nothing more is performed during the training phase (it is called a *lazy* training). During the evaluation phase, given a sample x of which we want to estimate the label y, all samples from $\mathcal{X}_a$ are parsed, and the k closest to x one are kept. The distance used is the euclidean distance:

$$\|x - x^i\|^2 = \sum_{j=1}^{d} (x_j^{\cdot} - x_j^i)^2.$$

The predicted label is the one that has majority amongst the selected *neighbors*:

$$y = \arg\max_y \left| \left\{ i : i \in \mathcal{J}_k(x), y = y^i \right\} \right|$$

where $\mathcal{J}_k(x)$ is the set of the indices of the k training samples that are the closest to x and $\left|\left\{i : i \in \mathcal{J}_k(x), y = y^i\right\}\right|$ is the number of samples amongst the k neighbors with the label y.

6. What is the error rate achieved on the validation set with $k = 3$?
7. What would be the error rate on the *training* set with $k = 1$? Explain.

wine_db_knn.py

```
57 # Fit the model on (Xa, Ya)
58 k = 3
59 clf = KNeighborsClassifier(n_neighbors=k)
60 clf.fit(Xa, ya)
61
62 # Predict the labels of samples in Xv
63 yv_ = clf.predict(Xv)
64
65 # evaluate classification error rate
66 valid_error = 1 - accuracy_score(yv, yv_)
67 print('k={:2} Predict error on validation set: {:.1%}'.format(k, valid_error))
```

The choice of $k = 3$ is arbitrary. We want to choose a better value of k. For this purpose, we fit the model with different values of k and select the optimal value based on the performances achieved on the validation set.

8. Why only odd values of k are tested (line 70 of the code below)?
9. Complete the code below and comment on the results. For which value of k is there a greater generalization gap?

wine_db_knn.py

```
68 k_max = 60
69 step = 2
70 k_ = np.arange(1, k_max, step)
71 valid_error_ = np.ndarray(len(k_))
72 train_error_ = np.ndarray(len(k_))
73
74 for i, k in enumerate(k_):
75
76     clf = KNeighborsClassifier(n_neighbors=k)
77     # fit the model
78     ...
79
80     # Predict the labels of samples in Xv, Xa
81     ya_ = ...
82     yv_ = ...
83
84     # evaluate classification error rate
85     valid_error = 1 - accuracy_score(yv, yv_)
86     train_error = 1 - accuracy_score(ya, ya_)
87
88     res_str = 'k={:2} Predict error on validation set: {:.1%} ({:.1%} on train)'
89     print(res_str.format(k, valid_error, train_error))
```

```
90      valid_error_[i] = ...
91      train_error_[i] = ...
92
93  fig3, ax3 = plt.subplots(1)
94
95  ax3.plot(k_, valid_error_, label='valid')
96  ax3.plot(k_, train_error_, label='train')
97
98  ax3.set_xlabel('k')
99  ax3.set_ylabel('Error')
100 ax3.legend()
101
102 fig3.show()
```

10. What is the value k^* of k minimizing the error rate on the validation state? Compare that value with the heuristics that states that the optimal value of k is approximately $\sqrt{N}$, where N is the number of training samples.

wine_db_knn.py

```
104 i_star = valid_error_.argmin()
105 k_star = k_[i_star]
106 print('k*={}'.format(k_star))
```

11. What is the error rate achieved on the test set with the selected model (*i.e.* $k = k^*$)?