MATH637 – Mathematical Techniques for Data Science

Project 2 Report: Classification analysis on wine dataset

Giuliamaria Menara, 702524795

Introduction

This report describes the application of different algorithms to create models able to predict the class label of input data. Such models are based on a small dataset and their goodness is evaluated in terms of accuracy.

Steps followed to solve this issue are presented below.

Load the dataset

We start by loading the Wine dataset from UCI Machine Learning Repository. This dataset contains the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. The analysis of each wine is characterized by thirteen different measurements taken for different constituents found in the three types of wine.

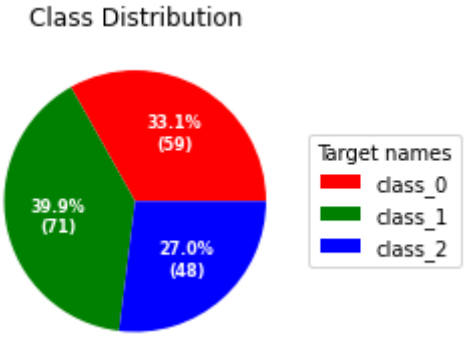
The constituents considered are: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines and Proline.

Main dataset characteristics are:

- number of instances: 178
- number of attributes: 13
- number of classes: 3
- class names: class_0, class_1, class_2
- number of missing values: 0.

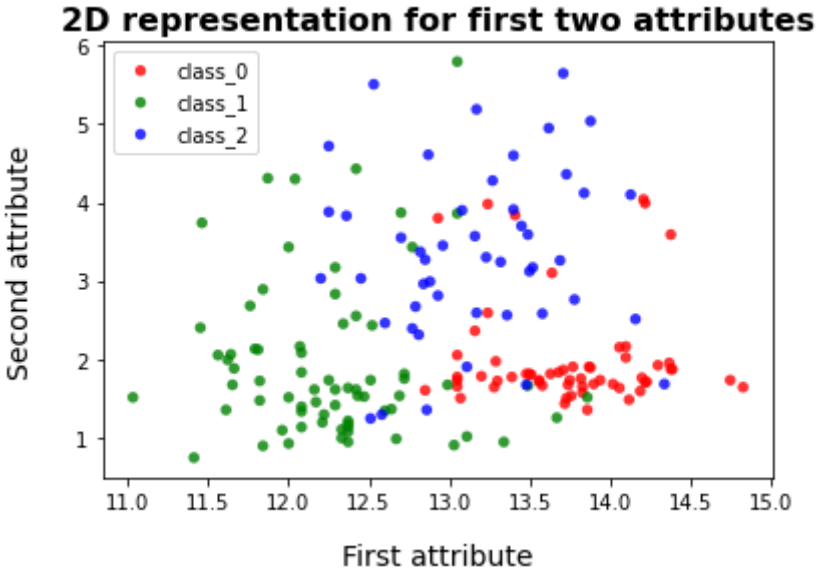
Another important thing to notice is the class distribution (quite balanced), showed below:

- 59 instances belong to class_0
- 71 instances belong to class_1
- 48 instances belong to class_2



Representation of first two attributes

After data exploration, each instance of dataset has been represented by the first two attributes (Alcohol and Malic acid measurements) through a 2-dimensional scatter plot.



As we can see from the graph, the three classes are not well-separated, so we cannot distinguish them clearly enough.

Analyzing better data belonging to the first two columns of dataset, it is possible to notice that the mean value of first attribute and the mean value of second attribute are different by one order of magnitude, as showed in the following table.

	Min value	Max value	Mean	Standard Dev
First attribute	11.0	14.8	13.0	0.8
Second attribute	0.74	5.80	2.34	1.12

Thus, it will be necessary to standardize data with the *StandardScaler* function before applying each algorithm. In this way data will be scaled to have mean and standard deviation respectively equal to 0 and 1.

Split the data into train, validation and test set randomly

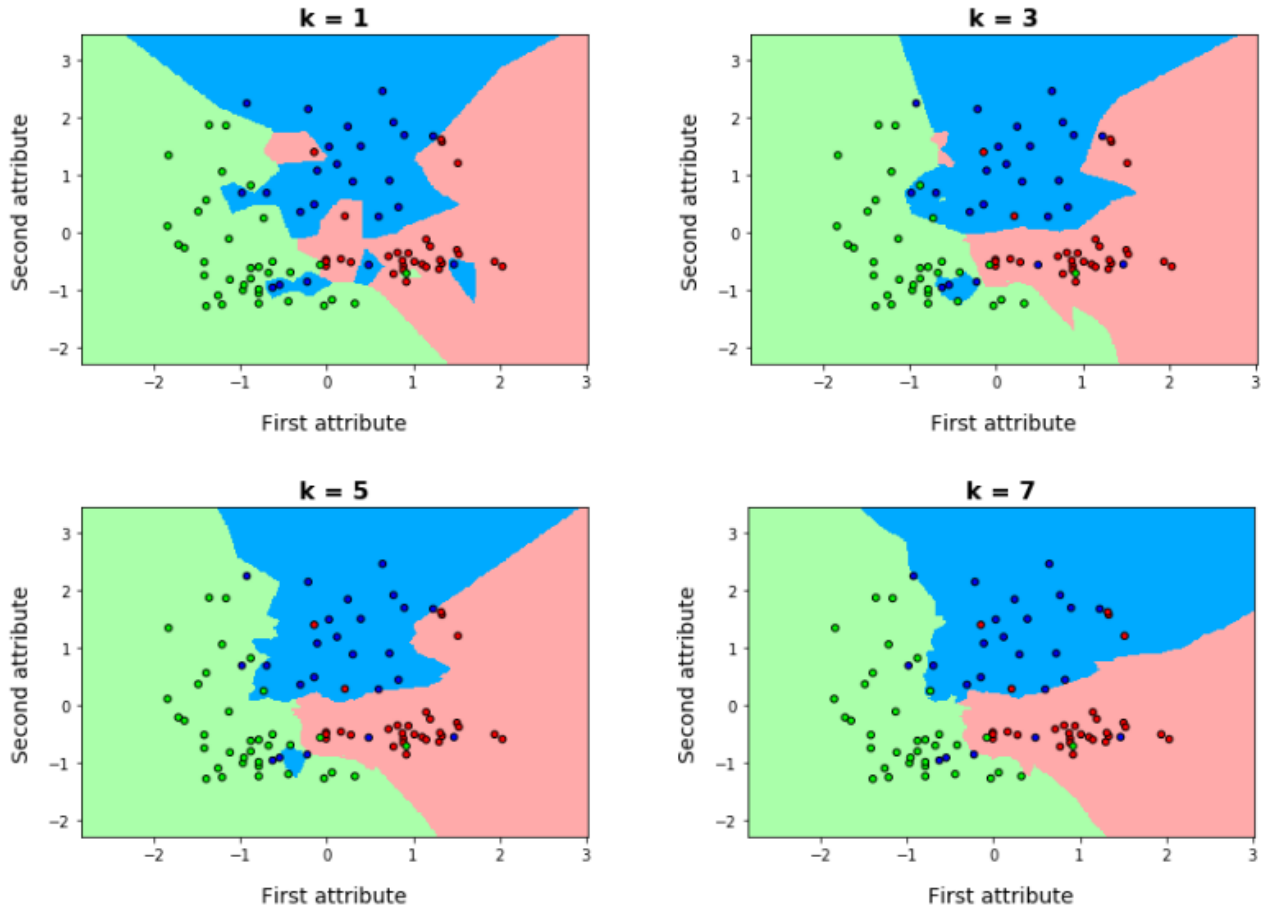
The split into train, validation and test set is made by using two times the function *train_test_split* from scikit library. To have a more equal partition of classes in each set, samples are stratified by targets and fixed by a *random_state* = 56 for the first call and a *random_state* = 32 for the second call.

Class distribution among sets obtained by using these parameters is showed in the following table.

	Class_0	Class_1	Class_2	Total
Training set	30	35	24	89
Validation set	11	14	10	35
Test set	18	22	14	54

Apply K-Nearest Neighbors

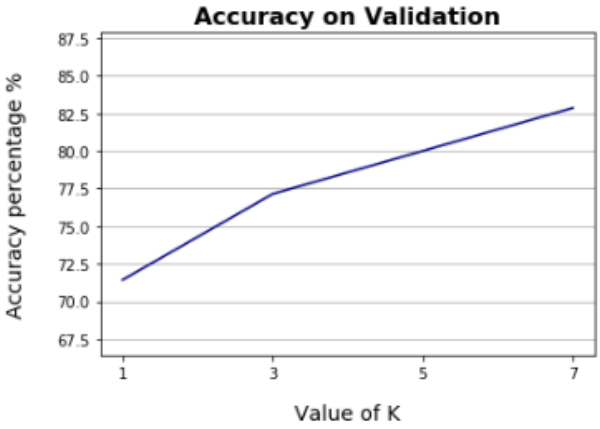
The first algorithm applied is k-nearest neighbors. As hyperparameter k we consider the first four odd numbers (1, 3, 5, 7).



The above pictures show the distribution of standardized data belonging to the training set, highlighting the decision boundaries that divide the three classes by different colors. As we can see, the value of *k* is controlling the shape of the decision boundary: a small value for *k* provides the most flexible fit, which has low bias but high variance (graphically, decision boundaries are more jagged); on the other hand, a higher *k* is more resilient to outliers (larger values of *k* result in smoother decision boundaries, which means lower variance but increased bias).

Evaluating this method on the validation set the following results are obtained:

K	*Accuracy (%)*
1	71.43
3	77.14
5	80.00
7	82.86



After this analysis, the k-nearest neighbors algorithm is applied to standardized samples coming from both training and validation set to build the model with the best value of *k* (7) obtained from the preceding step: in this way, the accuracy score on test set is equal to 75.93%, which is less than the value obtained from evaluation on validation set.

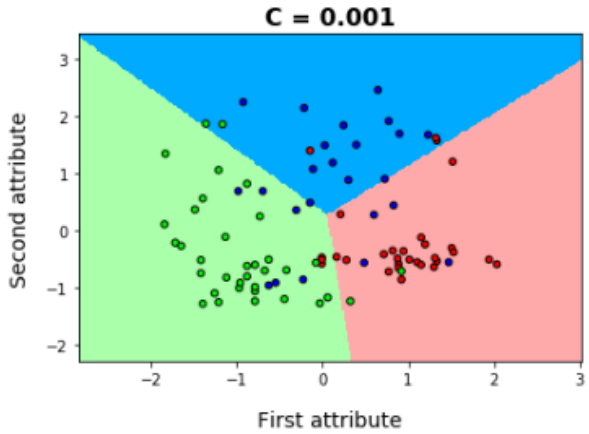
So, despite the fact that the highest value of *k* was chosen, the number of neighbors considered is still too small to avoid overfitting on training set data.

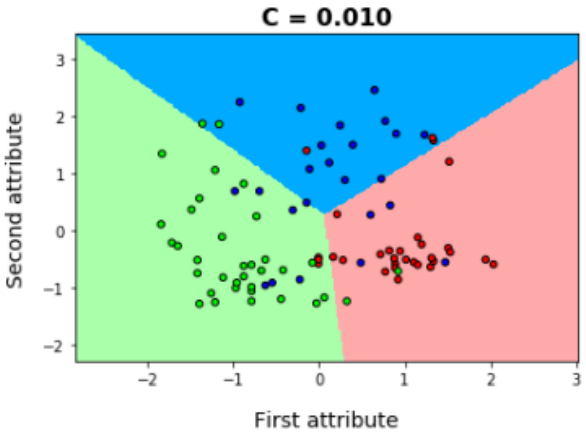
Apply Linear SVM

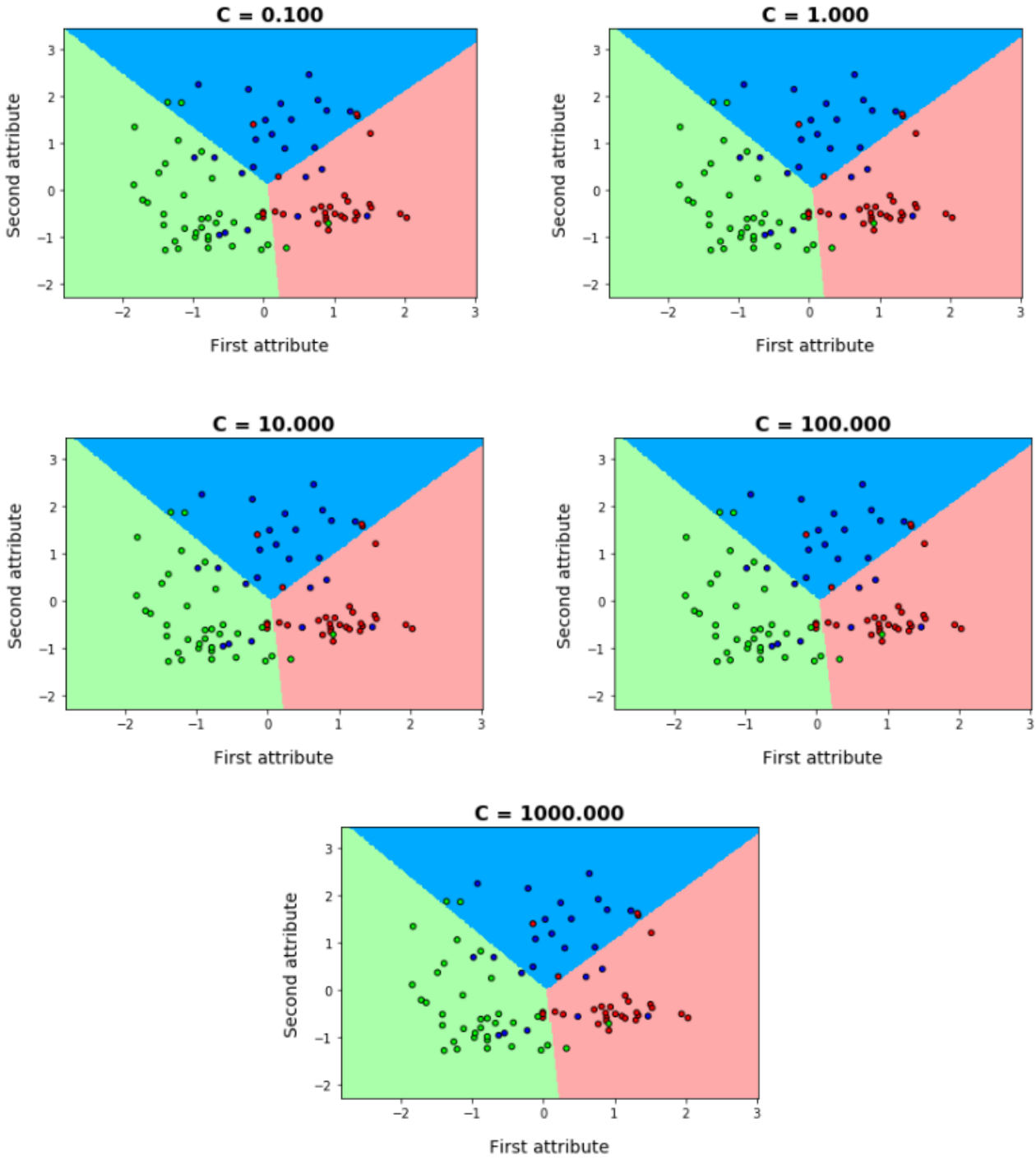
The second algorithm applied on data is the Linear Support-Vector Machine (SVM).

To perform this algorithm in python, the *LinearSVC* function from scikit-learn library is used, as it implements "*one-vs-the-rest*" multi-class strategy, thus training a single classifier per class.

As hyperparameter C, values 0.001, 0.01, 0.1, 1, 10, 100, 1000 are considered.





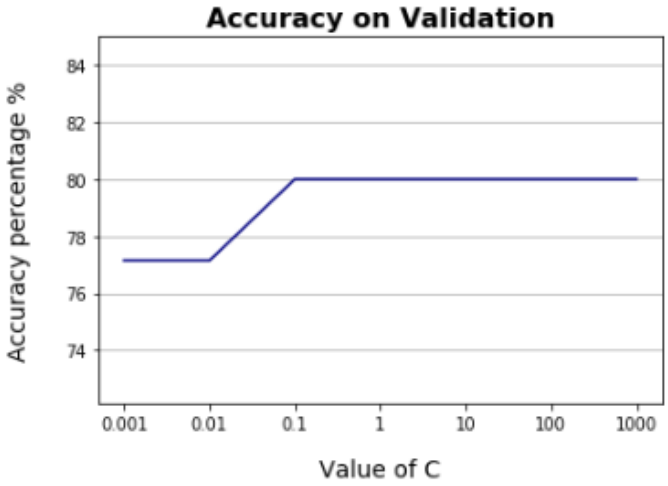


The pictures show the hyper-planes that separate the three different classes depending on the value of C. As we can see, the training set is not linearly separable, so we need a soft-margin approach that allows SVM to make a certain number of mistakes and keep margin as wide as possible, so that other points can still be classified correctly.

The number of mistakes is controlled by the hyperparameter C: when it is small the focus is on maximizing the margins, whereas when it is large the focus is more on avoiding misclassification. Therefore, decision boundaries change according to the number of points that it is possible to misclassify.

In the case of Linear SVM, there are not big graphical differences between one plot and the other, being the position and the slope of hyper-planes are quietly similar. This is also reflected on the validation accuracy trend.

C	*Accuracy (%)*
0.001	77.14
0.01	77.14
0.1	80.00
1	80.00
10	80.00
100	80.00
1000	80.00



After this analysis, *Linear SVM* is applied on both training and validation data: in this case the accuracy score is equal to 79.63%, which is very similar to the value obtained from the evaluation on validation set. So, this reveals that *Linear SVM* can build a model with lower overfitting on training set than *KNN*.

Differences between KNN and SVM

K-Nearest Neighbors is one of the simplest of classification algorithm for supervised learning. It is a non-parametric and lazy learning method: non-parametric means there is no assumption for underlying data distribution and so the model structure is determined from the dataset; lazy algorithm means it does not need any training data points for model generation, because all training data are used in the testing phase. This makes training faster and testing phase slower.

So, KNN does not attempt to construct a general internal model, but simply stores instances of the training data, and the classification is based on the distance metric and is computed from a simple majority vote of the nearest neighbors of each point.

On the other hand, Support Vector Machine is a discriminative classifier formally defined by separating hyperplanes. In other words, given labeled training data, the algorithm outputs optimal hyperplanes which categorize new examples. In 2-dimensional space these hyperplanes are lines dividing a plane into as many parts as there are classes.

It is originally designed for binary classification, but it can also work fine with multi-class problems, as proved by the previous analysis, adopting one-against-all strategy, a heuristic method that involves splitting the multi-class dataset into multiple binary classification problems.

Differently from KNN, SVM is slower during the training phase because model creation requires to evaluate all training data, but then test phase is faster.

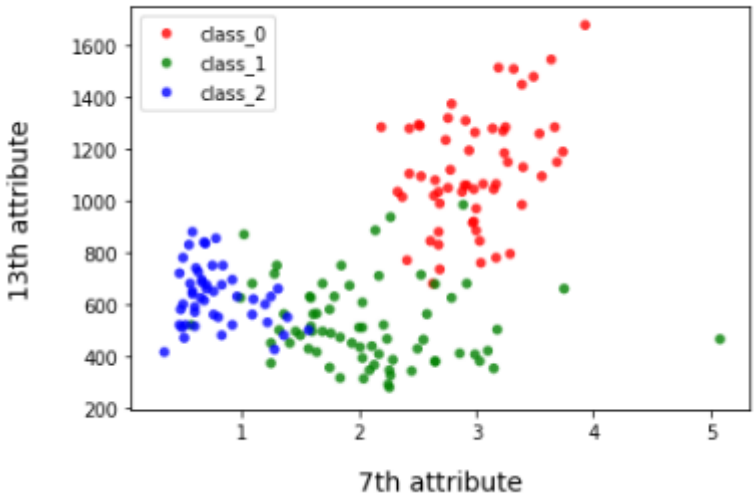
Trial with a different pair of attributes

After performing KNN and SVM on the first two attributes, another pair of features is selected. To make this choice, the *SelectKBest* function from scikit-learn library is used, which provides best features of dataset according to ANOVA F-value.

The most representative features extracted are the 7th and the 13th. Accuracy scores obtained evaluating models on test set with best hyperparameters found on validation set are summarized in the following table.

	KNN (K = 3)	Linear SVM (C = 0.1)
Accuracy	90.74%	92.59%,

As we can see, choosing these two features the accuracy percentage increases a lot for both classifiers. Hence it is possible to say that the three wine classes are identified better by these other two features. This is also confirmed by the following graphical representation.



References

[1] K-Nearest Neighbors (KNN).
https://scikit-learn.org/stable/modules/neighbors.html

[2] Support Vector Machines (SVM).
https://scikit-learn.org/stable/modules/svm.html

[3] LinearSVC function from Scikit-learn.
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[4] SVC function from Scikit-learn.
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[5] SelectKBest function from Scikit-learn
https://scikit-learn.org/stable/modules/generated/
sklearn.feature_selection.SelectKBest.html