Report of Homework #1

Machine Learning and Deep Learning course

Introduction

The first homework of "Machine learning and Deep learning" course is about 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[1].

This kind of problem is named as *Classification*, an instance of supervised learning[2] that consists of analyzing training data and producing an inferred function, which can be used for mapping new observations.

Steps followed to solve this issue are presented below.

1. Loading Wine dataset

The first step of the assignment is to load 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:

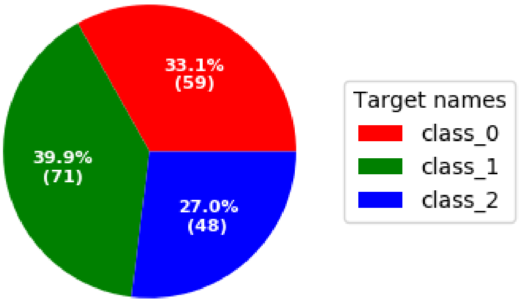
- Number of instances = **178**
- Number of attributes = **13**
- Number of classes = **3**
- Class names = **class_0**, **class_1**, **class_2**
- Number of missing values = **0**

[1] *Accuracy*: ratio of number of correct predictions to the total number of input samples

[2] *Supervised learning*: learning where a training set of correctly identified observations is available.

Another important thing to notice is the class distribution, that is showed in the pie chart below:

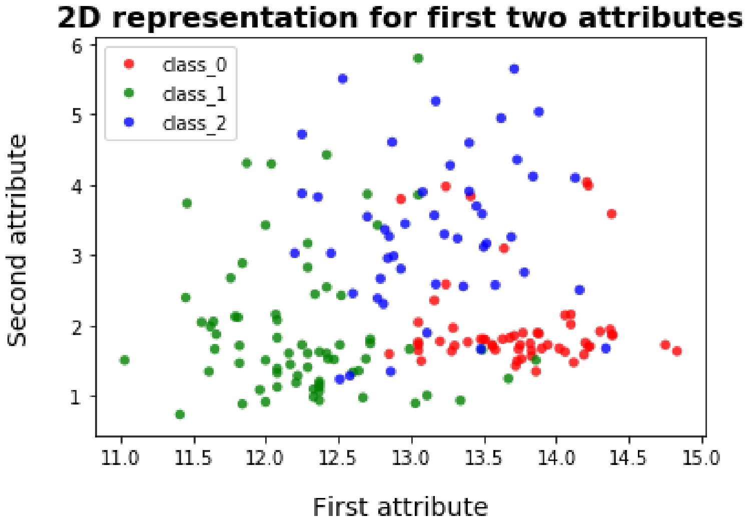
- **59** instances belong to class_0
- **71** instances belong to class_1
- **48** instances belong to class_2



As we can see from the graph, the classes are quietly balanced.

2. Representation of first two attributes

After data exploration phase, 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 from one order of magnitude, as showed in the following table.

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

So, 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.

3. Split 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, that shuffles data and divides it according to proportions provided by the assignment text (5:2:3). 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 invocation and a *random_state* = 32 for the second invocation.

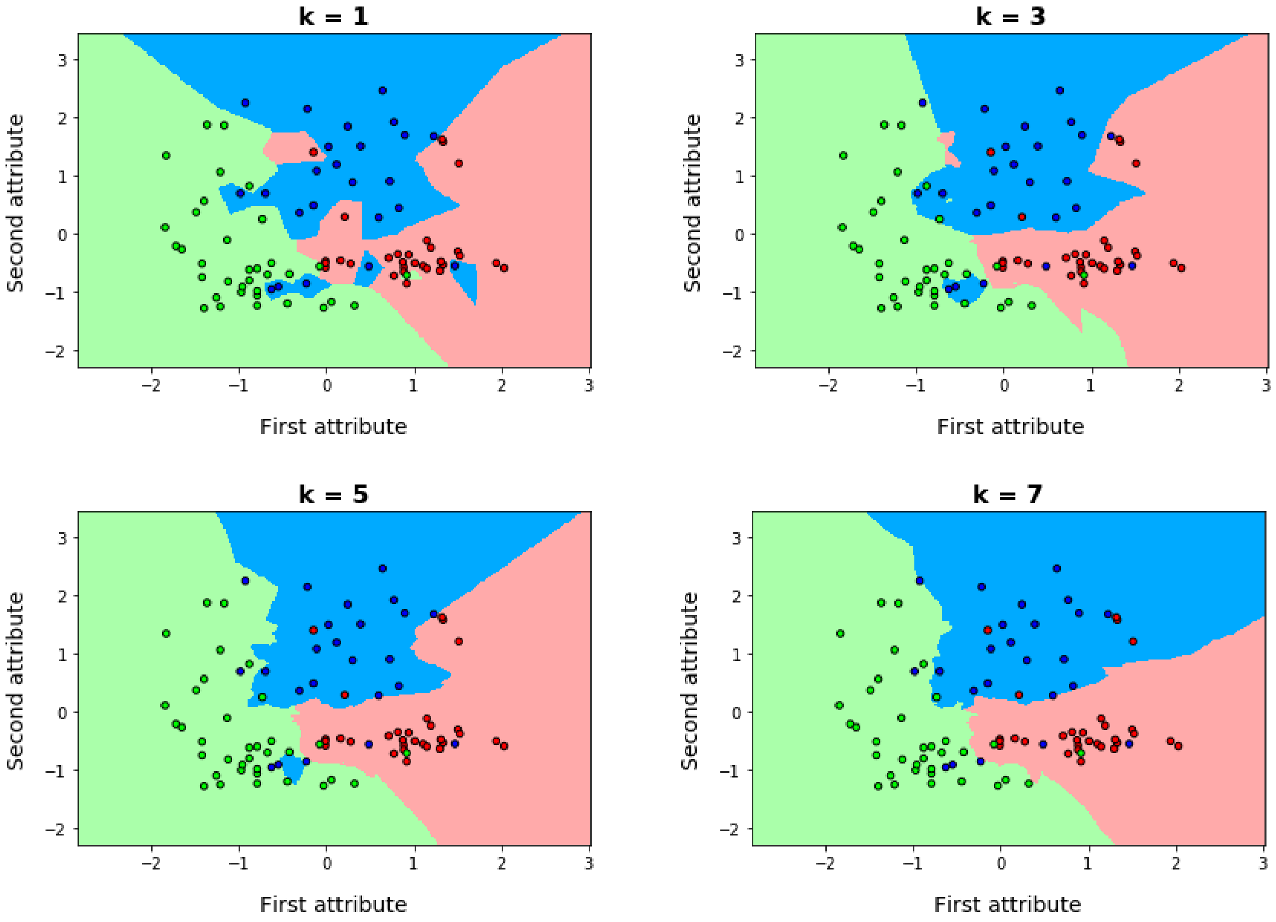
Class distribution among sets obtained by using these parameters is showed in the following table.

	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

4. Apply K-Nearest Neighbors

The first algorithm applied on data is *K-Nearest Neighbors* [1].

As hyperparameter *K* are considered the first four odd numbers (1, 3, 5, 7).



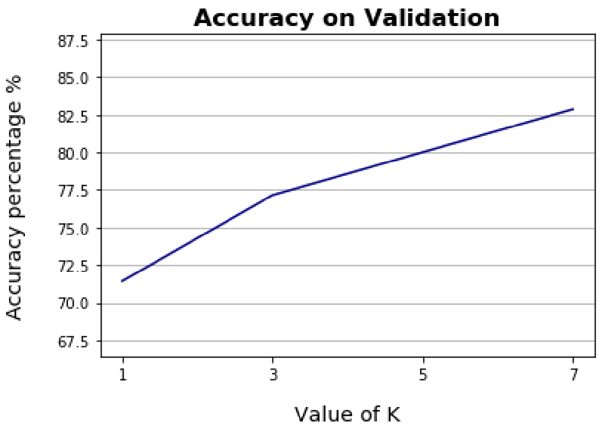
Four previous pictures show the distribution of Standardized data belonging to training set, highlighting decision boundaries that divide the three classes by different colors. As we can see, value of hyperparameter *K* is controlling the shape of the decision boundary: a small value for *K* provides 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* have smoother decision boundaries which means lower variance but increased bias.

This behavior is also verifiable using other training sets extracted from the dataset: indeed, even when there are more “outliers”, larger values of *K* reduce the effect of the noise on the classification making boundaries among classes less distinct.

Obviously, a different data extraction generates different graphs: in some cases it is possible to see more jaggedness and more “islands” different colored (as sign of variance) in the “sea” of one other classification area until *K* is equal to 7, although their presence is decreasing while increasing *K*.

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

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



As we can see, accuracy on validation set increases while *K* increasing its value.

In general, the accuracy of the *K-Nearest Neighbors* algorithm can be severely degraded by the presence of noisy, so the reason of such trend is probably linked to the presence of outliers in this training set extraction. Here, their negative effect is mitigated by the highest values of *K*.

But these results are not verified in all trials done: in some validation set extractions highest values of accuracy were corresponding to lowest values of K, while in other cases the trend of accuracy was not monotonous, then the highest accuracies were obtained when K was equal to 3 or 5.

In other dataset splits not only the trend, but also the percentage values were different: with these set compositions **71.43%** corresponds to the minimum, while **82.86%** is the maximum score, but using other training or validation sets, the lowest accuracy registered was about 60% while the highest got over the 90%.

After this analysis, *K-Nearest Neighbors* algorithm is applied on 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 accuracy score on test set is equal to **75.93%**, that is less than the value obtained from evaluation on validation one.

So, despite of it has been chosen the highest value of K from the list given from the assignment text, the number of neighbors considered is still too small to avoid overfitting on training set data.

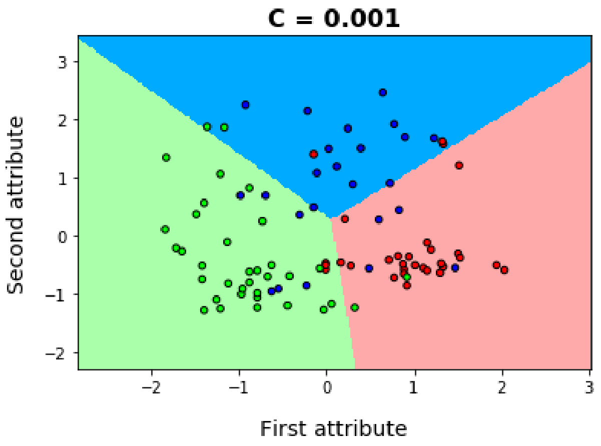
As conclusion, it is possible to say that the model is better at predicting validation data than test data because the model misclassifies many test data points.

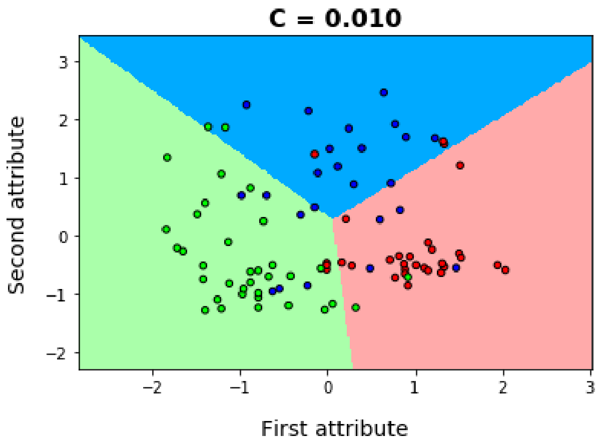
5. Apply Linear SVM

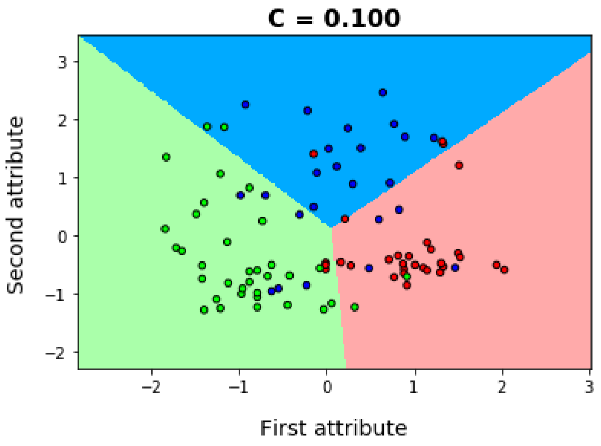
The second algorithm applied on data is the Linear Support-Vector Machine (*SVM*) [2].

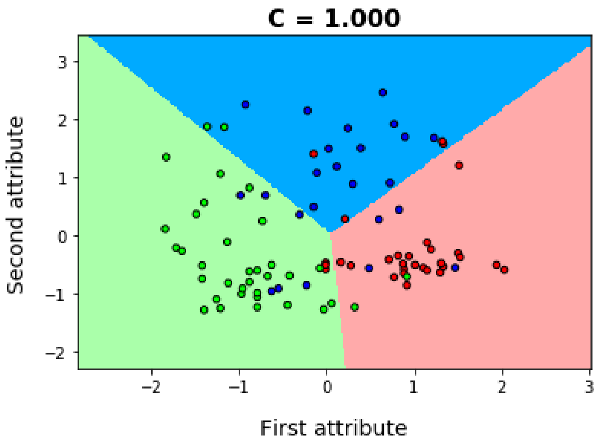
To perform this algorithm by python, *LinearSVC* [3] function from scikit-learn library is used, as it implements *"one-vs-the-rest"* multi-class strategy, thus training a single classifier per class.

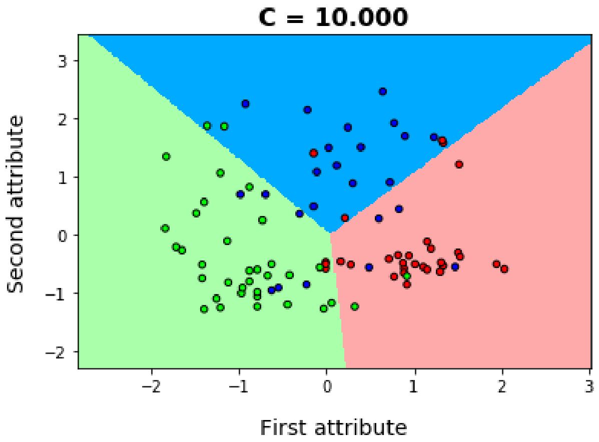
As hyperparameter *C*, values **0.001**, **0.01**, **0.1**, **1**, **10**, **100**, **1000** are considered.


C = 0.001

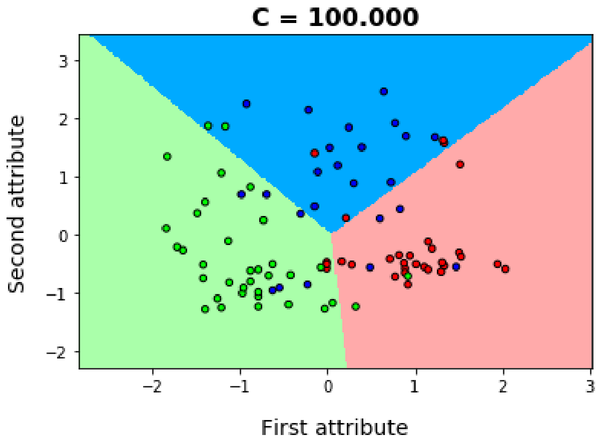

C = 0.010


C = 0.100

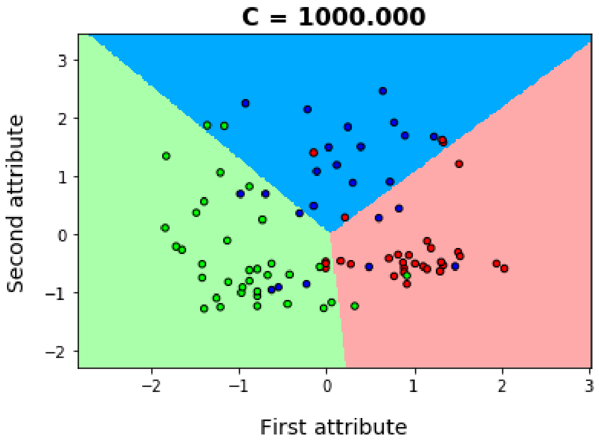

C = 1.000



C = 10.000



C = 100.000



C = 1000.000

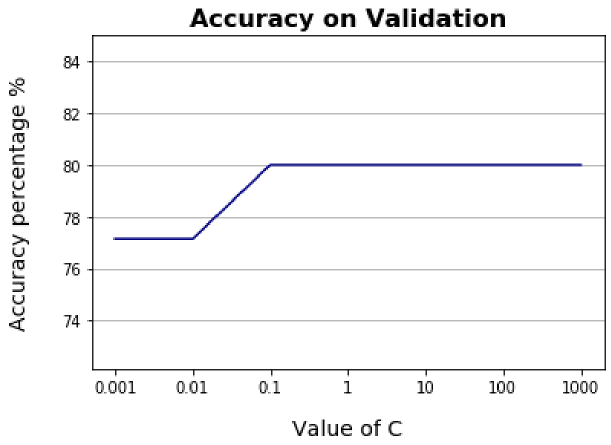
The pictures show the hyper-planes that separate the three different classes depending on the value of *C*. As we can see, training data is not linearly separable, so it is used a soft-margin approach that allows *SVM* to make a certain number of mistakes and keep margin as wide as possible, so that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

The number of mistakes is controlled by the hyperparameter *C*: when it is small, classification mistakes are given less importance and focus is more on maximizing the margins, whereas when it is large, the focus is more on avoiding misclassification at the expense of keeping the margins small.

So, decision boundaries change according to the number of points that it is possible to misclassify.

In this case, graphically there are not big differences between one plot and the other because 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



In this case, accuracy scores assume only two values: it is equal to **77.14 %** for the lowest *C*, while it is equal to **80.0 %** if *C* is 0.1 or greater.

So, accuracy score get best results at highest *C* values tried and this behavior has been noticed also for other training data extractions: in fact, it was possible to obtain the same trend with several different compositions of training set and also in these cases the scores were quietly similar to each other.

Obviously during the trials of the code, also different trends and lower (or higher) accuracy scores are emerged, so similarly to the application of previous algorithm, results are strongly influenced by training set and validation set composition.

After this analysis, *Linear SVM* algorithm is applied on both training and validation data in order to evaluate the model so built on test set with the first best value of *C* (0.1) got from the preceding step: in this case the accuracy score is equal to **79.63%**, that is very similar to the value obtained from the evaluation on validation set. So, it reveals that *Linear SVM* can build a model with lower overfitting on training set than *KNN*.

5. Apply RBF Kernel

The third algorithm applied on dataset is the *SVM* trained with *Radial Basis Function* (*RBF*) kernel. To perform this algorithm by python, the *SVC* [4] function from scikit-learn library is chosen. Even in this case, the multiclass support is handled according to a "*one-vs-rest*" scheme.

As hyperparameter *C*, the same values as before (0.001, 0.01, 0.1, 1, 10, 100, 1000) are considered, while, in this first application, the hyperparameter gamma is set to the default value, that is '*scale*'.

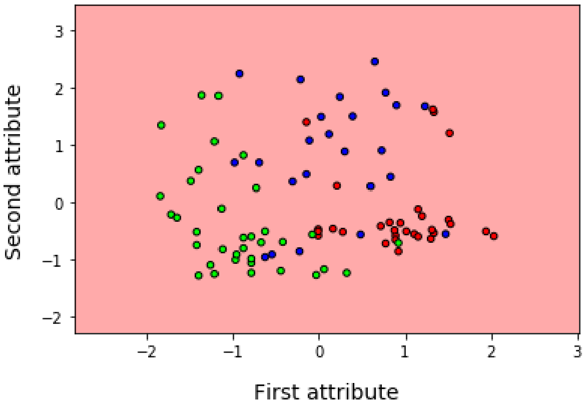
So:

$$\gamma = \frac{1}{n_{features} \times var(X_{train})} \approx 0.49$$

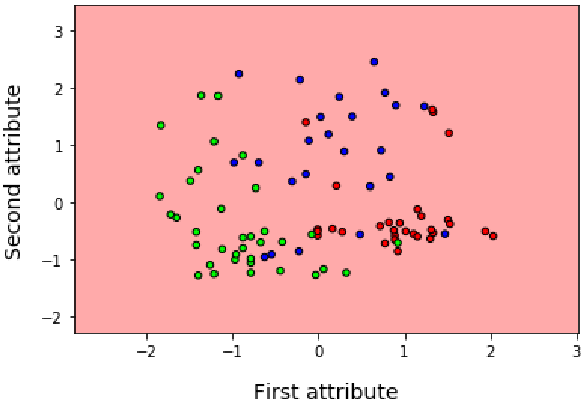
Parameter gamma is the inverse of the radius of influence of samples selected by the model as support vectors [5].

Applying *SVM* trained with *RBF Kernel*, following results are obtained.

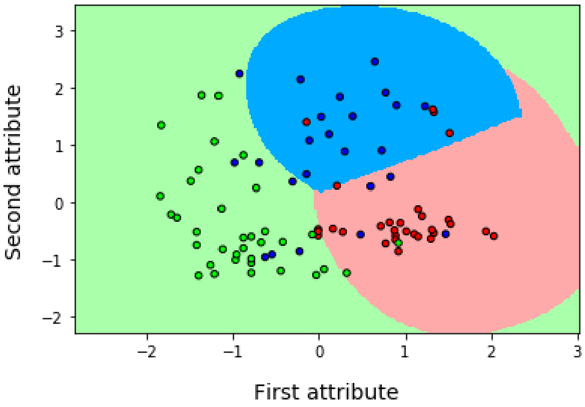
C = 0.001



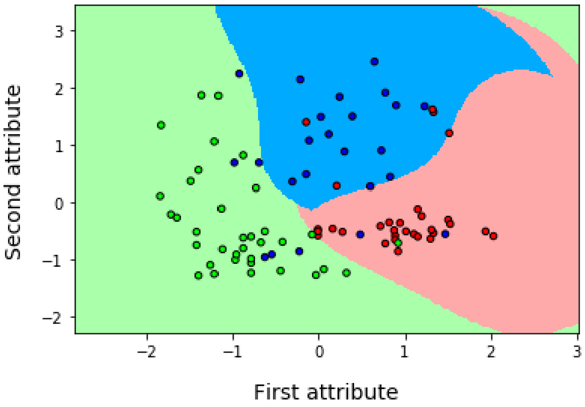
C = 0.010



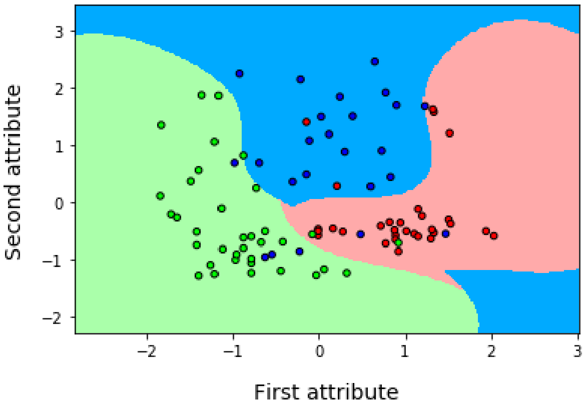
C = 0.100



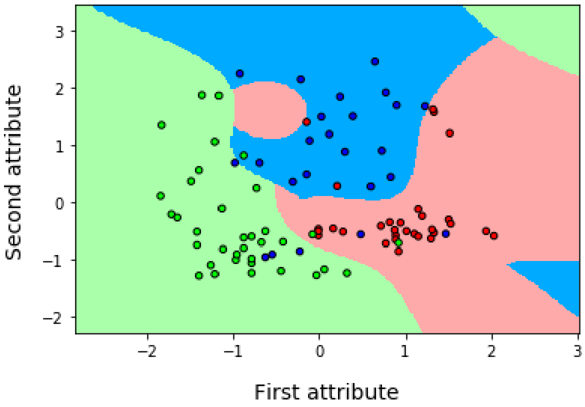
C = 1.000



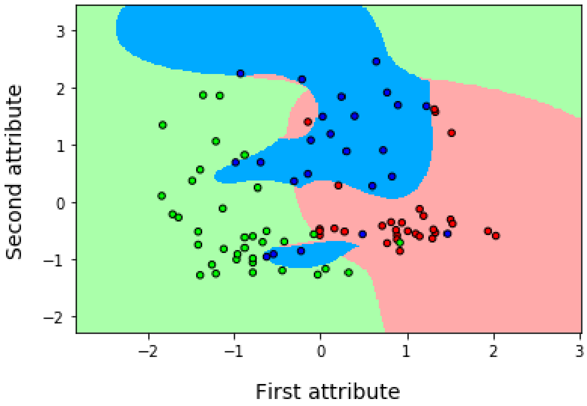
C = 10.000



C = 100.000



C = 1000.000

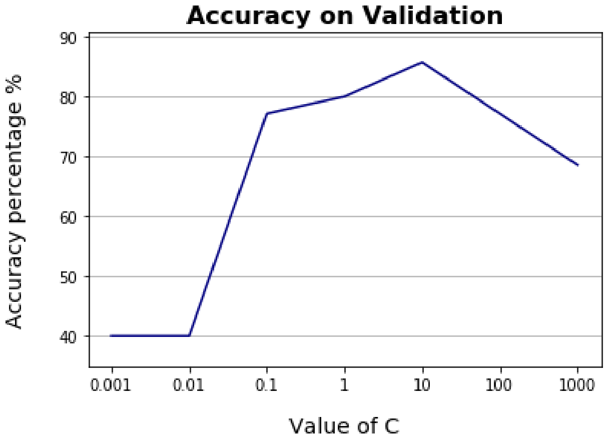


As seen before, parameter *C* trades off correct classification of training examples against maximization of the decision function's margin. For larger values of *C*, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower *C* encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

So, the reason due to we cannot see decision boundaries in the first two pictures (in which *C* has the lowest values) is that the classifier is too tolerant of misclassified data point, then the model is not able to separate classes and all new observations are classified as "*class_0*" points.

Evaluating this method on the validation set, it is possible to notice the following results:

C	*Accuracy (%)*
0.001	40.00
0.01	40.00
0.1	77.14
1	80.0
10	85.71
100	77.14
1000	68.57



As expected, lowest accuracy scores correspond to smallest *C* because in these cases the model recognize only one class, so the percentage corresponds to the relative presence of points belonging to that class in the validation set.

Increasing *C* value, the classifier becomes less tolerant to misclassified data points and therefore the decision boundaries are more severe.

Best accuracy is obtained when *C* is equal to 10: over this value, it is possible to see signs of overfitting because the classifier is starting to be penalized for any misclassified data points.

After this analysis, the *SVM* with *RBF Kernel* algorithm is applied on both training and validation data used before, so having a model to be evaluated on test set with the best value of *C* (10) obtained from the preceding step: in this case the accuracy score is equal to **77.78%**, that is lower than one obtained from the evaluation on validation set.

Comparing plots got from the application of *Linear SVM* and *SVM* with *RBF Kernel*, it is possible to see that the forms of boundaries are different from one algorithm to each other: in the first case boundaries are represented by an intersection of straight lines that sharply separate one class from each other, while in the second case borders are represented by curved lines that are the results of a linear separation in an higher dimension.

Indeed, in the case of *RBF Kernel*, *SVM* classifier exploits a technique called “Kernel trick”, that maps each point in the training set to a higher dimensional space according to the following kernel function:

$$k(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$$

However, lower accuracy score got by using *RBF Kernel* points out that *SVM* based on a linear kernel performs slightly better on this considered partitions from the original dataset.

6. Grid search

In order to find the best hyper-parameters for *SVM* classifier trained with *RBF Kernel*, a *Grid Search* [6] operation is performed on validation set, looking for best accuracy score.

As *C* 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3} are tuned, while as *gamma* 10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}. 10^{1}, 10^{3} are tried.

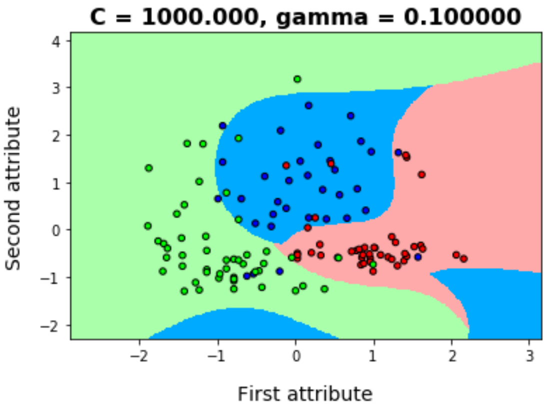
Results are summarized in the following table:

Gamma \ C	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
10^{-7}	40.0%	40.0%	40.0%	40.0%	40.0%	40.0%
10^{-5}	40.0%	40.0%	40.0%	40.0%	40.0%	68.6%
10^{-3}	40.0%	40.0%	40.0%	68.6%	80.0%	80.0%
10^{-1}	40.0%	68.6%	77.1%	80.0%	82.8%	85.7%
10^{1}	40.0%	42.9%	77.1%	71.4%	65.7%	65.7%
10^{3}	40.0%	40.0%	40.0%	45.7%	45.7%	45.7%

As we can see from the table, best configuration consists of:

- *C* = 1000
- *Gamma* = 0.1

So, applying *SVM* on test set, accuracy turns out to be equal to **81.48%** (quietly similar to the value obtained on validation phase and better than one got using default gamma). Moreover, this deeper research for gamma value has improved accuracy on test set with respect to one got by using linear kernel too. Decision boundaries and training data considered are represented in the picture showed in the next page.



7. Apply K-Fold

Applying a grid search on *C* and *gamma* values has allowed us to find a more general model able to getting better results on test set.

Now, after training and validation splits merging operation, *Grid Search* is repeated on data, but now performing a *K-fold* cross validation *[7]* by *GridSearchCV* *[8]* function from scikit-learn library.

Cross-validation is a straightforward technique used to avoid overfitting: it consists of dividing the data into K folds. Out of the K folds, K-1 sets are used for training while the remaining set is used for testing.

SVM algorithm is trained and tested K times and then the result of the *K-Fold* cross validation is the average of the results obtained on each test set.

In this case, *K-Fold* is applied only on data belonging to training and validation set and the value of K is equal to 5, so it is applied a 5-fold cross validation, obtaining the following scores.

Gamma \ C	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}
10^{-7}	39.5%	39.5%	39.5%	39.5%	39.5%	39.5%
10^{-5}	39.5%	39.5%	39.5%	39.5%	56.5%	96.8%
10^{-3}	39.5%	39.5%	54.0%	96.8%	96.0%	97.6%
10^{-1}	39.5%	95.2%	98.4%	98.4%	98.4%	98.4%
10^{1}	39.5%	39.5%	39.5%	39.5%	39.5%	39.5%
10^{3}	39.5%	39.5%	39.5%	39.5%	39.5%	39.5%

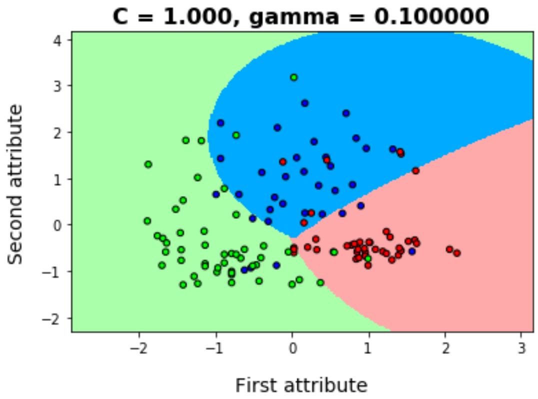
As we can see from the table, grid search made performing cross validation provides higher accuracies in many cases. Best score is now very good on validation phase and it is equal to **98.4%.**

GridSearchCV function chooses the first configuration that gives best average score on folds, then hyperparameters selected are equal to:

- *C* = 1
- *Gamma* = 0.1

So, applying *SVM* on test set, accuracy results equal to **79.63%**, that is a bit less than one obtained during grid search without cross validation.

Decision boundaries and training data considered are represented in the picture below.



Performing the script with different distributions of data into training and test set, it is possible to see that accuracy scores obtained with *K-Fold* on test set is often very similar to results got without *K-Fold* and in some cases they are equal to each other because hyperparameters chosen are the same.

So, due to available data shortage, K-Fold cross validation do not allow us to find a hyperparameter configuration able to make *SVM* based on *RBF Kernel* better on classifying samples from the extracted test set, although now the model seems to be less affected by overfitting (as it is possible to notice from boundaries plot).

8. 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 and costlier.

So, *KNN* does not attempt to construct a general internal model, but simply stores instances of the training data and classification is based on the distance metric and is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned to data class which has the most representatives within the nearest neighbors of the 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, where each class lay in one side.

It is originally designed for binary classification, but it can also work finely 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.

Summarizing the results obtained by the application of each model on test set, we can see that *KNN* has got the worst accuracy result on this dataset, while *SVM* trained with *RBF Kernel* and tuned by optimal hyper-parameters from the grid search, has got the best one.

	KNN	Linear SVM	RBF Kernel SVM	RBF Kernel SVM	RBF Kernel SVM
				(Grid Search)	(K-Fold)
	(K = 7)	(C = 0.1)	(C = 10)	(C=1000, γ=0.01)	(C=1, γ=0.1)
Accuracy	75.93%	79.63%,	77.78%	81.48%	79.63%,

The success of *RBF Kernel* is probably due to the nature of dataset: indeed, as it was noticeable in pictures before, the dataset is not linearly separable in 2-dimensional space, but projecting data points into an higher dimension, the training set becomes linearly separable.

Anyway, as said previously, results are strongly influenced by data distribution into training and test set, especially in case of small datasets like this.

9. Trial with a different pair of attributes

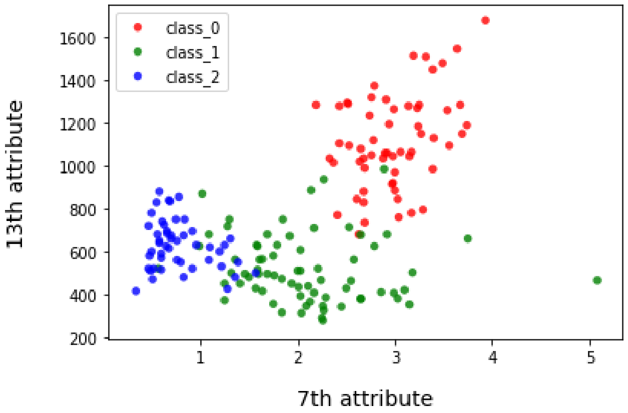
After performing *KNN* and *SVM* on the first two attributes, another couple of features to apply preceding classifiers is selected. To make this choice, *SelectKBest* [9] function from scikit-learn library is exploited. It provides best features of dataset according to ANOVA F-value. Most representative features extracted are the 7th and the 13th feature of wine dataset.

Accuracy scores obtained evaluating models on test set with best hyperparameters found on validation set are summarized in the following table.

	KNN	Linear SVM	RBF Kernel SVM	RBF Kernel SVM	RBF Kernel SVM
				(Grid Search)	(K-Fold)
	(K = 3)	(C = 0.1)	(C = 1)	(C=100, γ=0.01)	(C=1, γ=0.1)
Accuracy	90.74%	92.59%,	92.59%	92.59%	92.59%,

As we can see, choosing these two features, accuracy percentage increases a lot its value for every classifier.

So, it is possible to say that the three wine classes are identified better by these other two features, that allow to distinguish categories more clearly, as we can see on 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] *RBF SVM* parameters
https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[6] Tuning the hyper-parameters of an estimator
https://scikit-learn.org/stable/modules/grid_search.html

[7] Cross-validation: evaluating estimator performance
https://scikit-learn.org/stable/modules/cross_validation.html

[8] *GridSearchCV* function from Scikit-learn
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

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