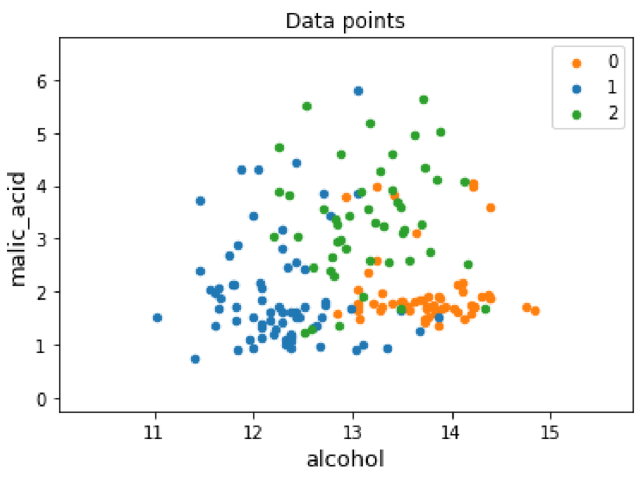
Homework 1 - Report

Francesco Montagna, S277596

0. COMMON STEPS

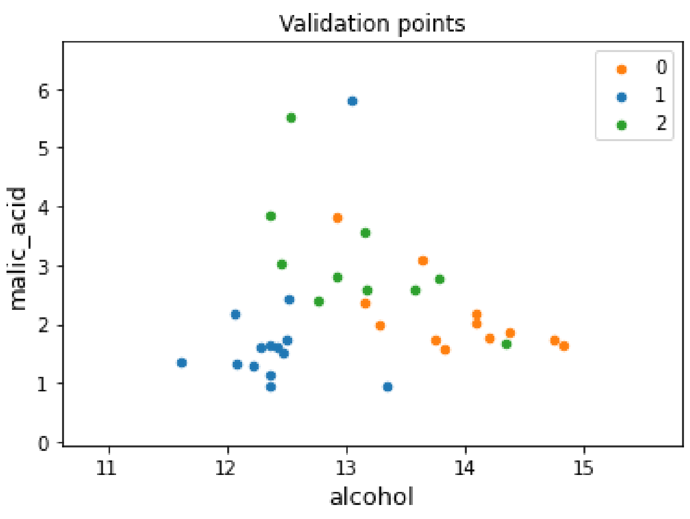
Some steps are common to all the codes implementing the 3 required classifiers.
First step has been importing Wine dataset from *sklearn.datasets*.
Then from the imported dataset, features has been separated from label column and a feature reduction has been performed, subsampling only *alcohol* and *malic_acid* columns (indices 0 and 1).

Having only 2 features on which to train the model allow to plot them on a graph: the following shows the data distribution per label



Then, the dataset has been splitted in training, validation and test set with a 5:2:3 proportion respectively.
In order to preserve the original data distribution in the new partitions, stratification has been used in the splitting phase.

The validation set resulted in



with label distribution: {0: 0.33, 1: 0.39, 2: 0.28} very similar to the distribution of Wine dataset {0: 0.33, 1: 0.40, 2: 0.27}

Note the the dataset is originally unbalanced resulting in quite less example for label 2: this partial lack of examples, also considering the small dimension of the dataset, might negatively influence the model accuracy for that particular label.

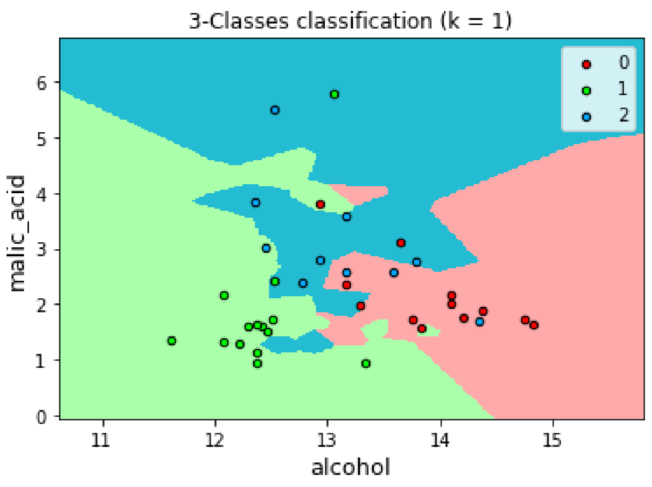
Note: due to the small size of the dataset, the results are biased and affected by the random component introduced by the dataset split. In a more realistic condition one should face this issue, maybe relying on different data sources. But considering that the main goal of the experience is to focus on the boundaries learning process with different classifiers, rather than building a model ready for the use, I decided to eliminate the random component at split time, in order to have comparable results, allowing me to focus on the learning process.

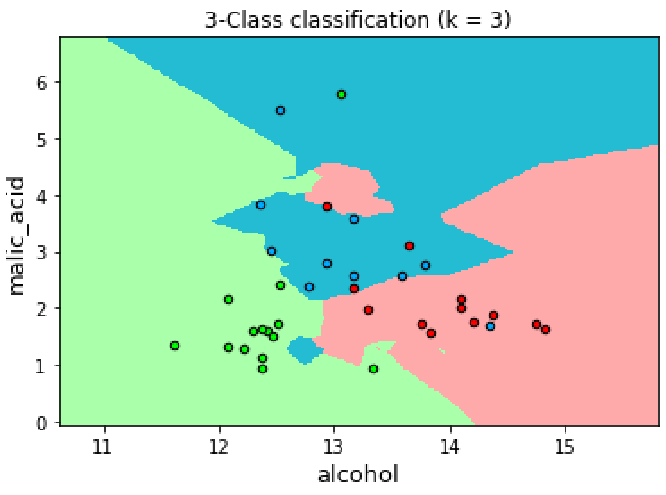
1. K-NEAREST NEIGHBORS

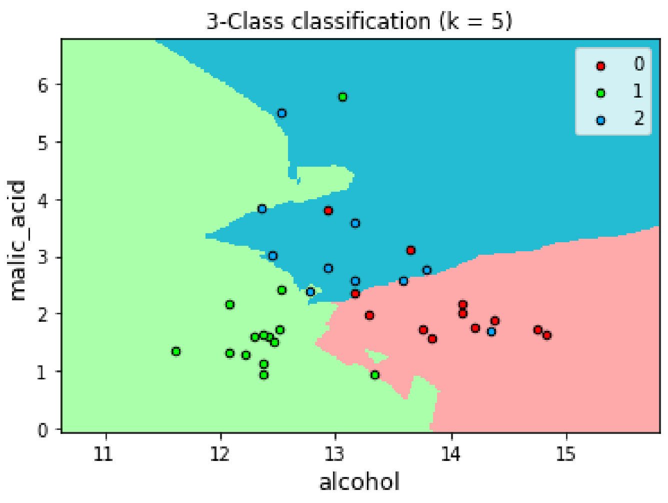
1.1 Training

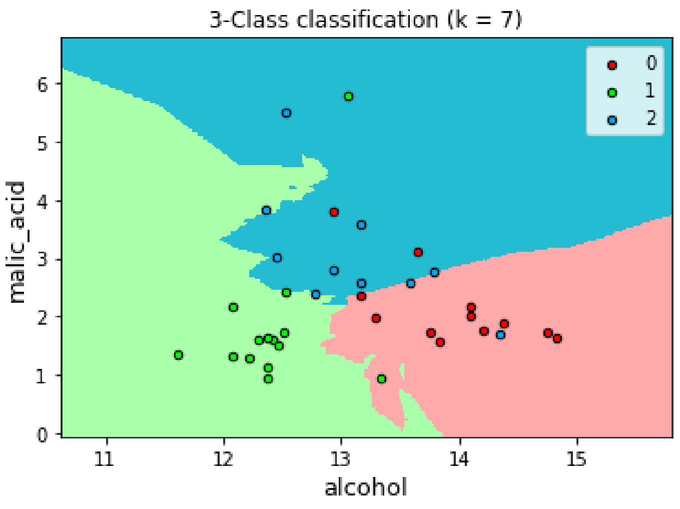
In order to train the KNN classifier I used the scikit learn class *KNeighborsClassifier*.
The only hyperparameter tuned in the training procedure is K, in particular comparing accuracy scores on the validation set for K = [1, 3, 5, 7].
For what concerns other parameteres available they have ben left to their default settings. In particular *distance* = Euclidean distance, and *weights* = 'uniform' for the majority voting when K > 1.

For each K value, these are the plots of the obtained decision boundaries in the feature space, along with validation points with the ground truth labels:





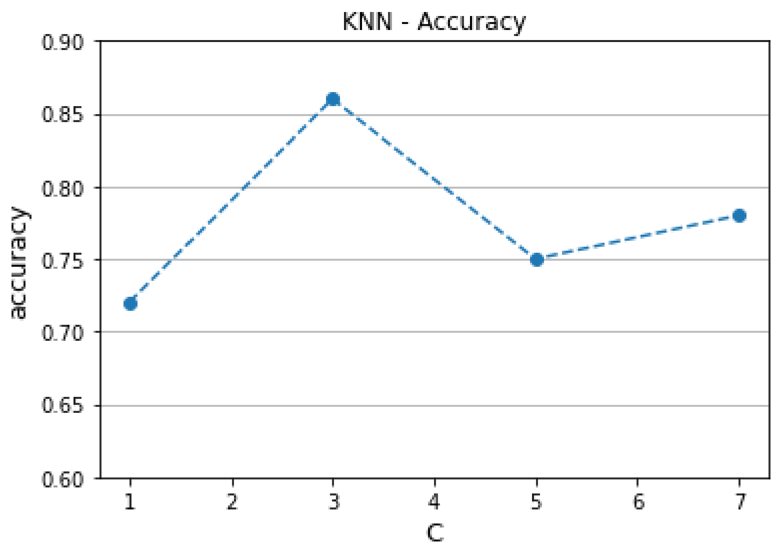




Decision boundaries change by changing K value: for K = [1, 3] boundaries don't seem well defined for label 2 especially: lowering K value on the one hand we build a simpler model, on the other the model finds harder to classify in region characterized by outliers. This is clearly observable in the up-center part of the graphs, where the blue region appears quite fragmented: this may be due to the fact that the learned boundaries overfit the training distribution.

This situation slightly improves with growing K = [5, 7], where it is possible to observe similar boundaries, with a region associated to label 2 classification which is more uniformly defined then for previous K values.

For each of the K value accuracy has been eveluated using *accuracy_score* class provided by scikit learn. Plotting the results we have



where the graph allows to choose best performing K on the validation set: in this case it is K = 3.

1.2 Test Prediction

Having learned best K in the tuning phase, now it is possble to train a KNN classifier with K = 3 and to have it performing on the test set.

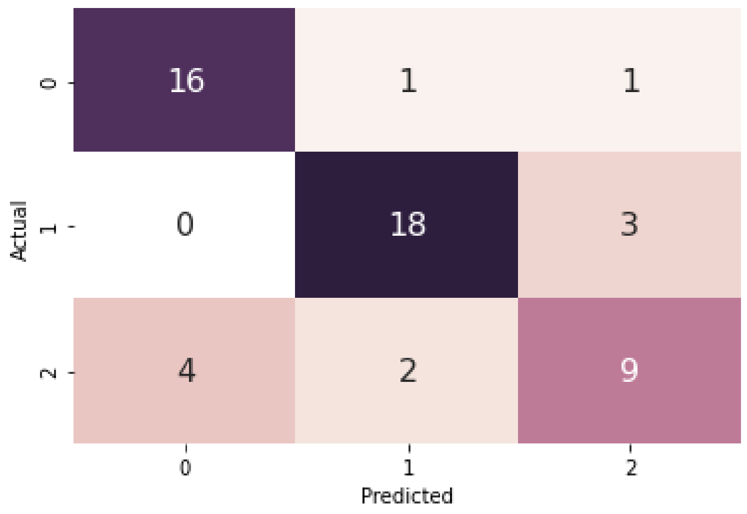
```
clf = KNeighborsClassifier(3) #define a  model with best parameter: K = 3
clf.fit(X_train, y_train)
y_test_pred = clf.predict(X_test) #predicttion on the test set

test_accuracy = accuracy_score(y_test, y_test_pred).round(decimals = 2) #prediction accuracy

test_accuracy
```

The above code returns an accuracy value of 0.8

Lastly, it is possible to plot a confusion matrix to see how well the KNN classifier performs on the test set.
As expected worst accuracy is achieved on label 2 points, where a partial explaination can be low number of label 2 training examples .



2. LINEAR SVM

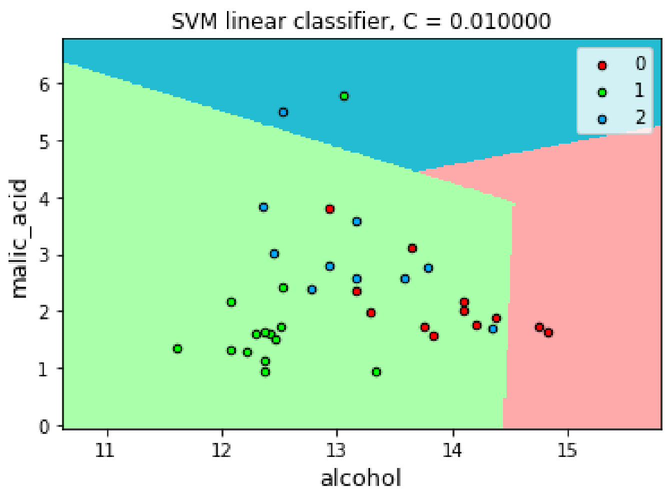
2.1 Training

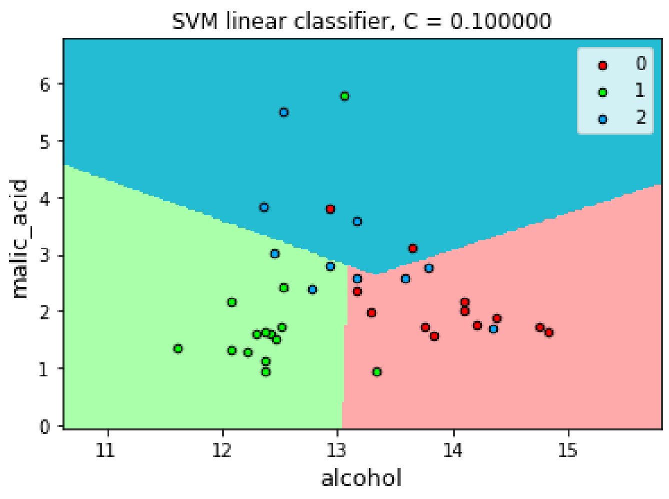
Linear SVMclassification on Wine dataset is done using *SVC* class from scikit-learn library.
The goal of the training phase in this case is to tune C giving best performance on the validation set.
The accuracy has been computed for C = [0.001, 0.01, 0.1, 1, 10, 100, 1000], for each of which has been trained the following classifier:

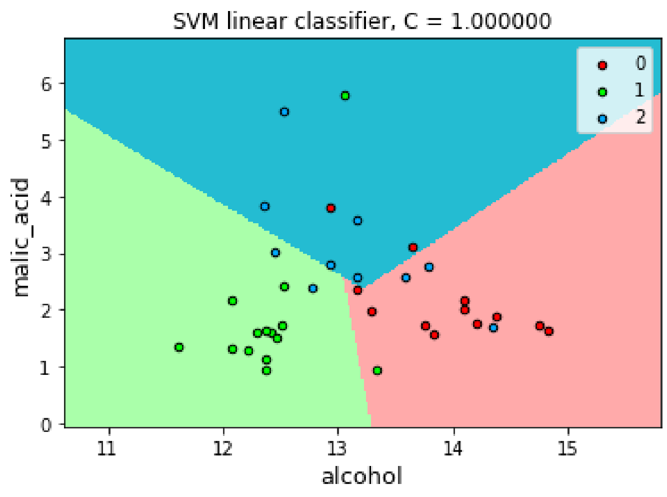
```
clf = SVC(C = c, kernel = 'linear')
clf.fit(X_train, y_train)
```

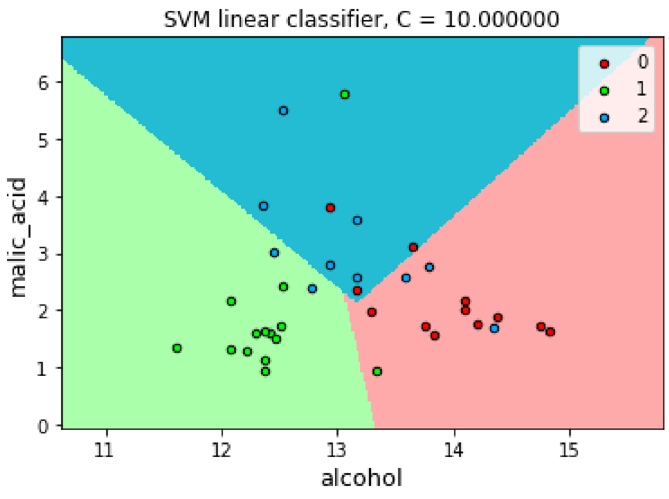
where a linear kernel is specified, and c refers to a value in the above C list.

As already done for KNN classifier, linear boundaries learned in the model are plotted ni a graph. I report here a meaningful sample of them, to see how boundaries changed. The whole list of graphs is avaliable running the provided code.



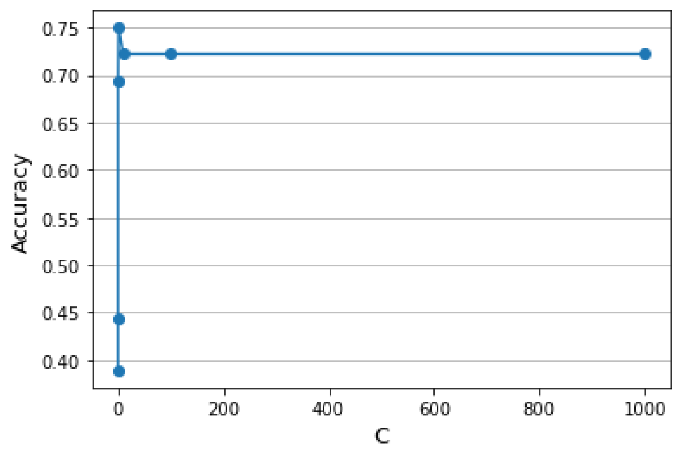




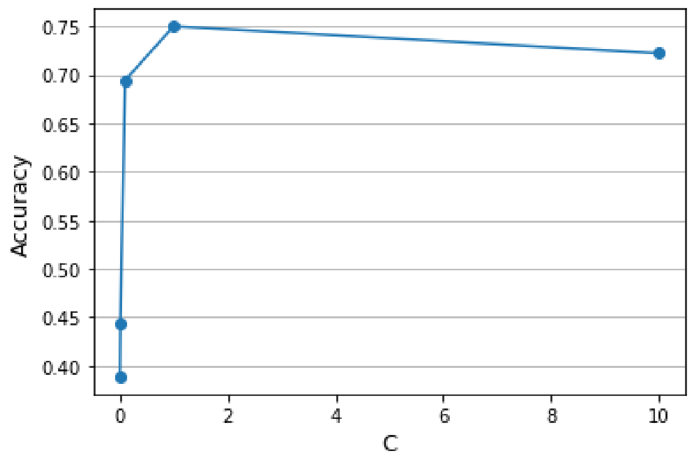


where for C larger than 1 the boundaries settles on same positions.
Boundaries changes with C since increasing C means to have softer marging, i.e. allow more errors. It must also be considered that larger C implies larger memory cost since more points have to be memorized in order to memorize the boundairs. Following this consideration for very similar boundaries, the choice must fall on lowest C value (this consideration are important when dealing with datasets larger that this one, but in principle must be done).

Accuracy has been then evaluated for each of the exploited C value



Since the above graph is hardly readable, I report a more meaningful graph which actually allows to choose best C value:



From the latter is clear that best performance on the validation test are reached with C = 1, which value will be used to train the model actually used on the test set

2.2 Test Prediction

Once best C value has been learned,the model is trained and tested on unseen data:

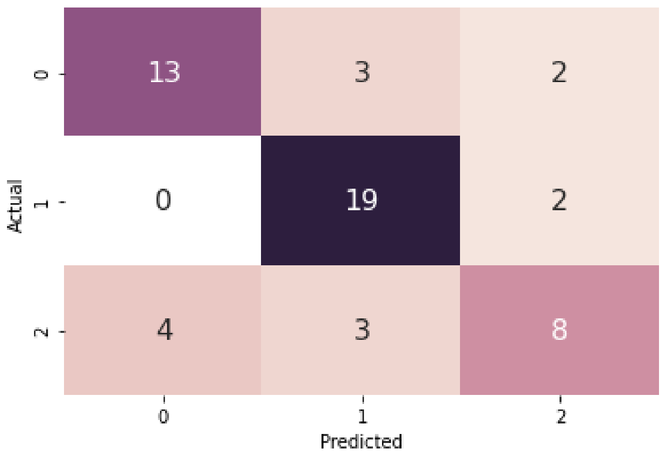
```
clf = SVC(C = 1)
clf.fit(X_train, y_train)
y_test_pred = clf.predict(X_test)

test_accuracy = accuracy_score(y_test, y_test_pred).round(decimals = 2)

test_accuracy
```

Which gives an accuracy score of 0.74

In order to have better insight on model performance on single labels also in this case a confusion matrix can be computed



3. RBF KERNEL

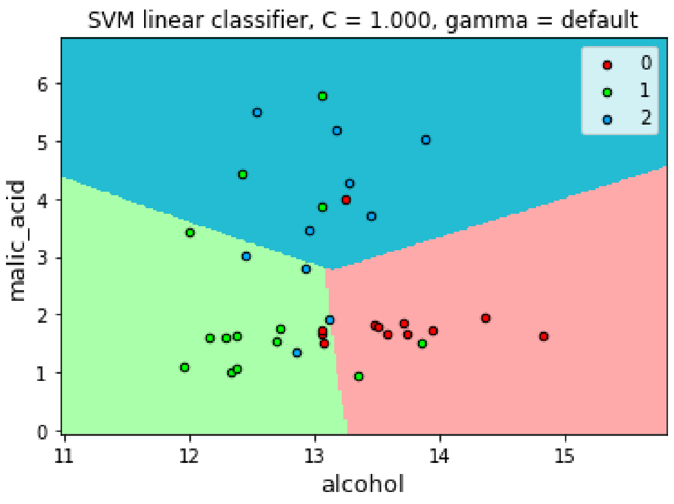
3.1 Training: gamma = default

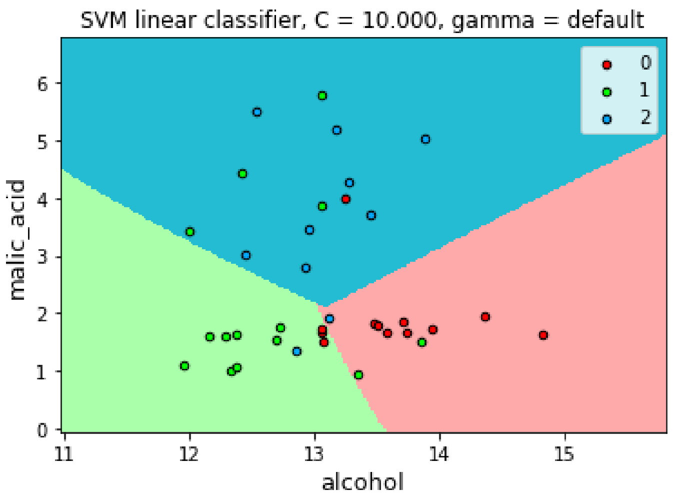
Before repeating the tuning pipeline on both gamma and C, the tuning can be performed only on C, but this time specifying as kernel value '*rbf*' (radial basis function). Therefore this time SVM is a non linear classifier.

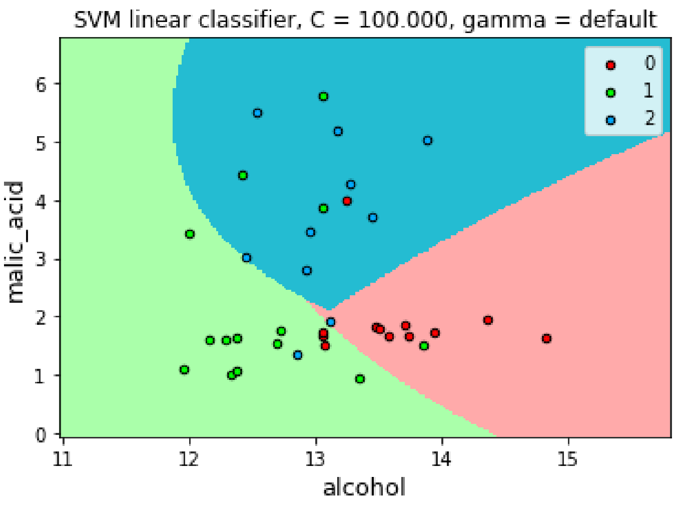
Since gamma is not tuned, it is left to its defaul value, which is '*scale*' : in this default setting gamma is computed as 1 / (n_features * X_train.var()) which is approximateveli equal to 1/60 = 0.017

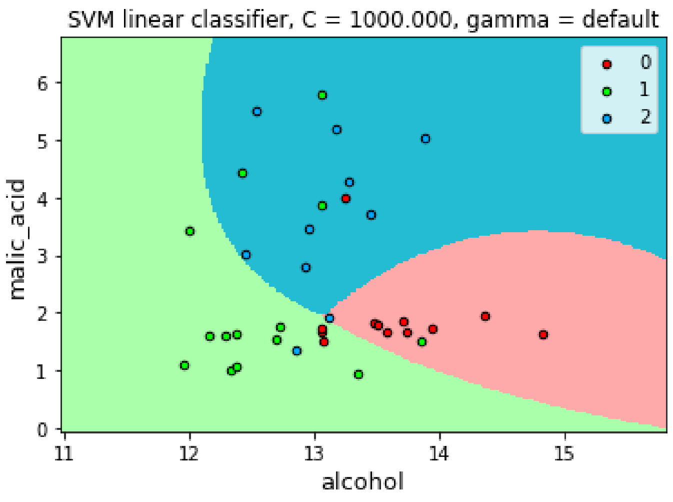
Now the already specified pipeline is used to find best C value performing predition on the validation dataset. Tested values will be C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

Some of the learned boundaries are polotted below. As in previous case the whole set of graphs can be obtained running the given code









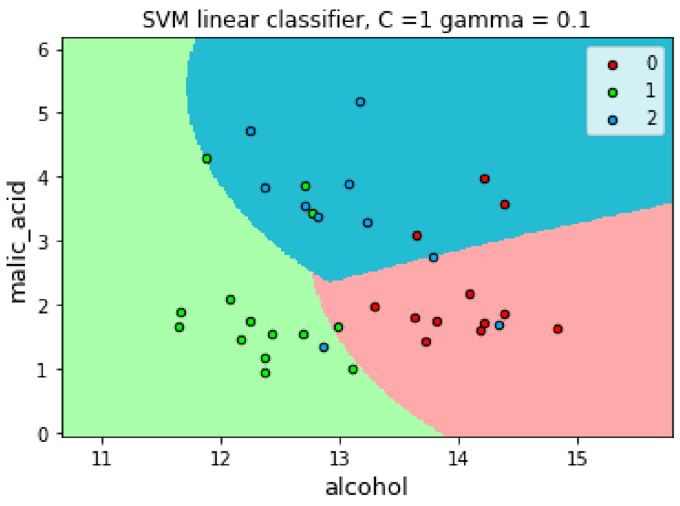
For C = [1, 10] the boundaries are almost linear: this because gamma value is quite small, therefore the model is learning a quasi-linear classifier. The boundaris are almost identical to the one obtained with kernel = ‘linear’.

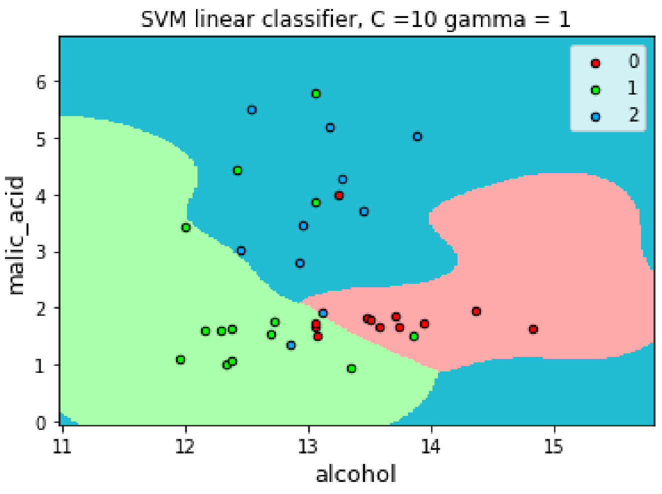
For larger C values the combined effect of C and gamma allow the model to learn non linear boundaries: we can already observe the effect of the non linear Kernenl. Next step is to perform a gridsearch in order to compare accuracy for all possible C-gamma pairs

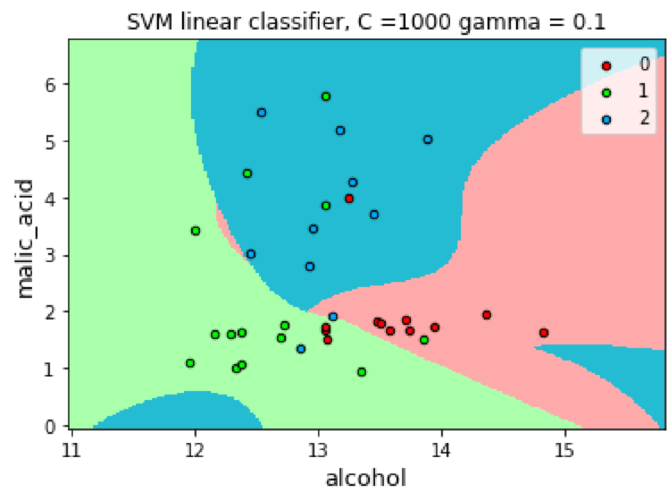
3.2 Training: C and gamma tuning

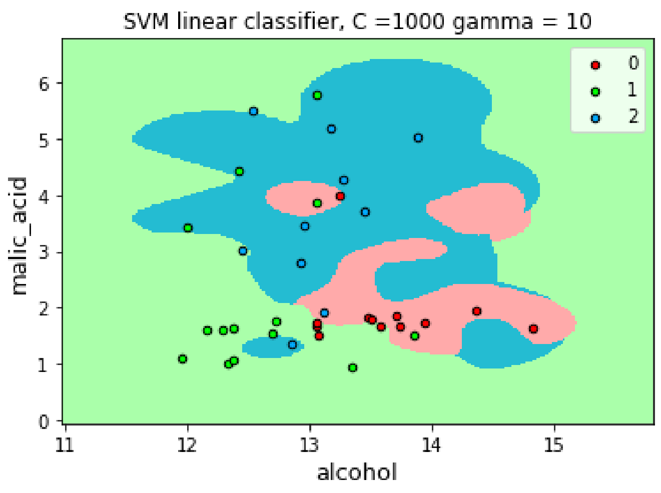
As mentioned, this section describes the training phase over which both C and gamma of the non linear classifier are tuned: this is done by performing a grid search over C = [0.1, 1, 10, 100, 1000] and gamma = [0.01, 0.1, 1, 10]

Let’s first plot some of the boundaries learned just for illustrative purpose, since it is not possible to exctract a meaningful subset of images between all the possible C and gamma combinations



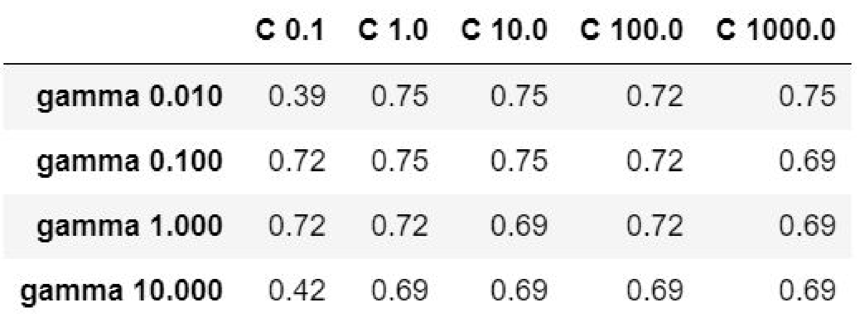






It is now possible to observe highly non linear boundaries.
Especially in the last graph, to observe overfitting of the training distribution which leads to fragmented classification regions.

The accuracy score table on the validation set is given by



	C 0.1	C 1.0	C 10.0	C 100.0	C 1000.0
gamma 0.010	0.39	0.75	0.75	0.72	0.75
gamma 0.100	0.72	0.75	0.75	0.72	0.69
gamma 1.000	0.72	0.72	0.69	0.72	0.69
gamma 10.000	0.42	0.69	0.69	0.69	0.69

One can also observe that for some intermediate values of gamma we get equally performing models when C becomes very large: it is not necessary to regularize by enforcing a larger margin. The radius of the RBF kernel alone acts as a good structural regularizer. In practice though it might still be interesting to simplify the decision function with a lower value of C so as to favor models that use less memory and that are faster to predict. [1]
According to this logic a plausible choice is C = 1 and gamma = 0.1

3.3 Test Prediction

With C= 10 and gamma = 0.1 an SVM classifier is fitted on the training set. Then it is possible to perform prediction on the test set of unseen data, proceeding as done with previous classifiers: in this case it is obtained a test accuracy equal to 0.83

3.4 KFold cross validation

Another way to proceed in the validation step is to perform cross validation. For this reason validation training set previously defined are merged together to form a new, larger training set.

Using the KFold class built in the scikit-learn library, is possible to set *n_splits*, i.e. the number of folds: in this case n_splits = 5

KFold validation is for sure more expenssive then the hold-out technique used until now, but it is also more reliable: training is performed K times (K = 5) and the returned accuracy is an average of the K accuracy obtained on the different splits. It is therefore also possible to compute standard deviation of the accuracy on the K splits, which is a meaningful insight on the stability of the model.

Re-using the previously built skeleton to grid search over C and gamma best values, the following accuracy score are obtained. They are reported along with the standard deviation charachterizing the average value

	C 0.0	C 0.1	C 1.0	C 10.0	C 100.0	C 1000.0
gamma 10.000	(0.299, +/-0.141)	(0.310, +/-0.132)	(0.773, +/-0.072)	(0.750, +/-0.058)	(0.750, +/-0.058)	(0.750, +/-0.058)
gamma 1.000	(0.299, +/-0.141)	(0.601, +/-0.131)	(0.817, +/-0.078)	(0.829, +/-0.039)	(0.817, +/-0.047)	(0.727, +/-0.040)
gamma 0.100	(0.299, +/-0.141)	(0.356, +/-0.156)	(0.782, +/-0.072)	(0.783, +/-0.059)	(0.784, +/-0.045)	(0.784, +/-0.045)
gamma 0.010	(0.299, +/-0.141)	(0.299, +/-0.141)	(0.445, +/-0.145)	(0.738, +/-0.060)	(0.782, +/-0.103)	(0.782, +/-0.062)

from which we see that the best parameters according to the cross validation performed are C =10 and gamma = 1.

Fitting the classifier with the chosen values, it can be used on the test set. Computing the accuracy the reached score is 0.81

Notice the difference between the accuracy score reported in previous paragraph, i.e. 0.83, and the current 0.81: the 2 values are of course different, since the 2 models are trained in 2 different training set. This leads to different classifiers and therefore to different performances.

4. EXTRA

4.1 KNN vs SVM

In this experience 2 classifiers have been used: KNN and SVM.
Generally speaing, since KNN doesn't need an actual training phase, it is computationally less intensive and very easy to implement. In fact KNN classification of unseen data is simply based on distance from the training set records. Therefore on the one hand this approach is very immediate to implement, on the other it scales badly with training set dimension: memory cost can be unsustainable for too many records, as well as distance computation can result in very time expensive task.

If you compare this characteristics with an SVM classifier, in the latter case we need to train the model, which construction is therefore more expensive. The time consumption for the training increases for multi class problems: in a one vs all approach, one SVM must be trained for each class; if instead one vs one scheme is used, given N classes then *N*(N-1)/2* classifiers are constructed. This is even more expensive and actually is the scheme implemented in scikit learn SVC for multi label classfication.
At the same time since SVM needs to memorize only those vectors used to define decision boundaries, it has a lower memory usage than KNN, which makes it feasible in situations where KNN is not due to dataset dimension.

4.2 Different pairs of attributes

All classification tasks until this point have been led with an arbitrarly chosen couple of features pair. This last section is used to exploit the research of the most meaningful pairs of attribute in the classification task.
It is important to notice that due to the fact that the 13 features varies in very different ranges of values (several orer of magnitude differences) in this case is necessary to normalize the data: this has been done with the MinMaxScaler class implemented in scikit-learn, with the range setted to default value = (0, 1)

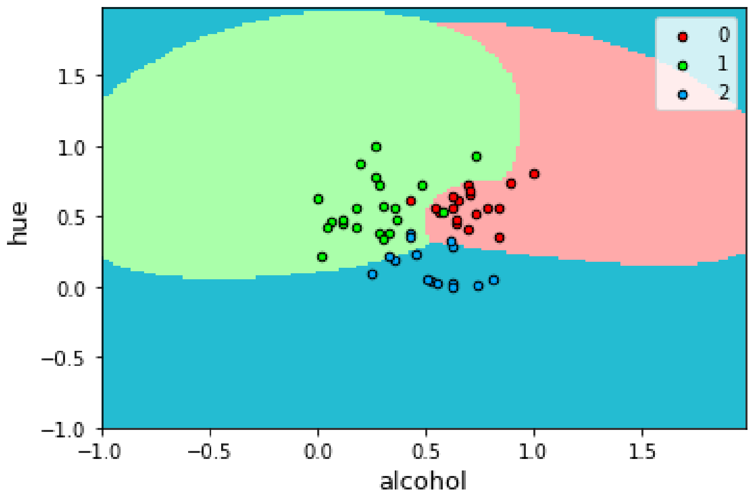
Classification has been done using an SVM classifier with 'rbf' kernel.
As in previous tasks, a gridsearch has been performed over gamma and C parameters' values with gamma = [0.01, 0.1, 1, 10] and C = [1, 10, 100, 1000].

Using GridSearchCV class form sciki-learn library, cross validating over the trainset, with cv = 3. Between all the available pairs I kept track of best parameters (gamma and C pair) and best accuracy, related to the best pair of attribute found, ending with the following results:

- best accuracy = 0.992
- best parameters = {'C': 100, 'gamma': 1}
- best attributes = ('alcohol', 'hue')

Training a model with the selected best features and parameters, on the test set (which size stands as 30% of the dataset) the predictions reached an accuracy = 0.89.
Around 10% of accuracy has been lost at test time, and this may be symptomatic of some overfitting.

The new classifier outperform both in training and test phase all the ones defined in previous sections. This are the non linear boundaries learned:


hue
alcohol