POLITECNICO DI TORINO

Master’s degree in Computer Engineering

Artificial Intelligence and Machine Learning

HOMEWORK 1
WINE CLASSIFICATION

Student:
Enrico Loparco, s261072

Professor:
Tatiana Tommasi

ACADEMIC YEAR 2019-2020

Table of Contents

Introduction

The aim of the homework is to perform classification over a dataset about wines, using different machine learning algorithms: starting from K-Nearest Neighbors to linear SVM and kernel SVM, to compare the approaches and discuss the results obtained.

For each algorithm a hyper-parameter tuning phase is conducted to choose the optimal parameters, using the train and validation set.
Then, in the model evaluation, the parameters selected are used to train (using train and validation set together) the model and compute the accuracy over the test set.

No initial data analysis is included, since it is not required by the homework and because a general description of data is directly provided with the dataset.
In preprocessing, standardization is applied to deal with attributes distributed over different ranges.

Dataset

The dataset used is taken from the scikit-learn library.
It includes 178 samples belonging to 3 different classes (representing different wine categories), each one characterized by 13 measurements, such as alcohol content, color intensity and malic acid quantity.
Instances are well balanced among the classes, all features are continuous values and labels are categorical (as expected for classification).
There are no missing values.

KNN

In all the processes, only the first two features are considered (‘alcohol’ and ‘malic_acid’), to obtain a clear 2D plot and visualize decision boundaries.
The dataset is split into train, validation and test sets in proportion 5:2:3 by applying 2 consecutive splits, since default split operation is binary in scikit-learn.
Standardization (subtraction of mean and division by standard deviation to obtain zero mean and unit variance) of data is used: first standard deviations and means for train and validation set are calculated, then applied to both train+validation and test set.

A basic version of KNN is adopted, using the euclidean distance as distance metric and uniform weights so that all points in each neighborhood are weighted equally.

For parameter tuning 1, 3, 5 and 7 (odd numbers) are used as values for K.

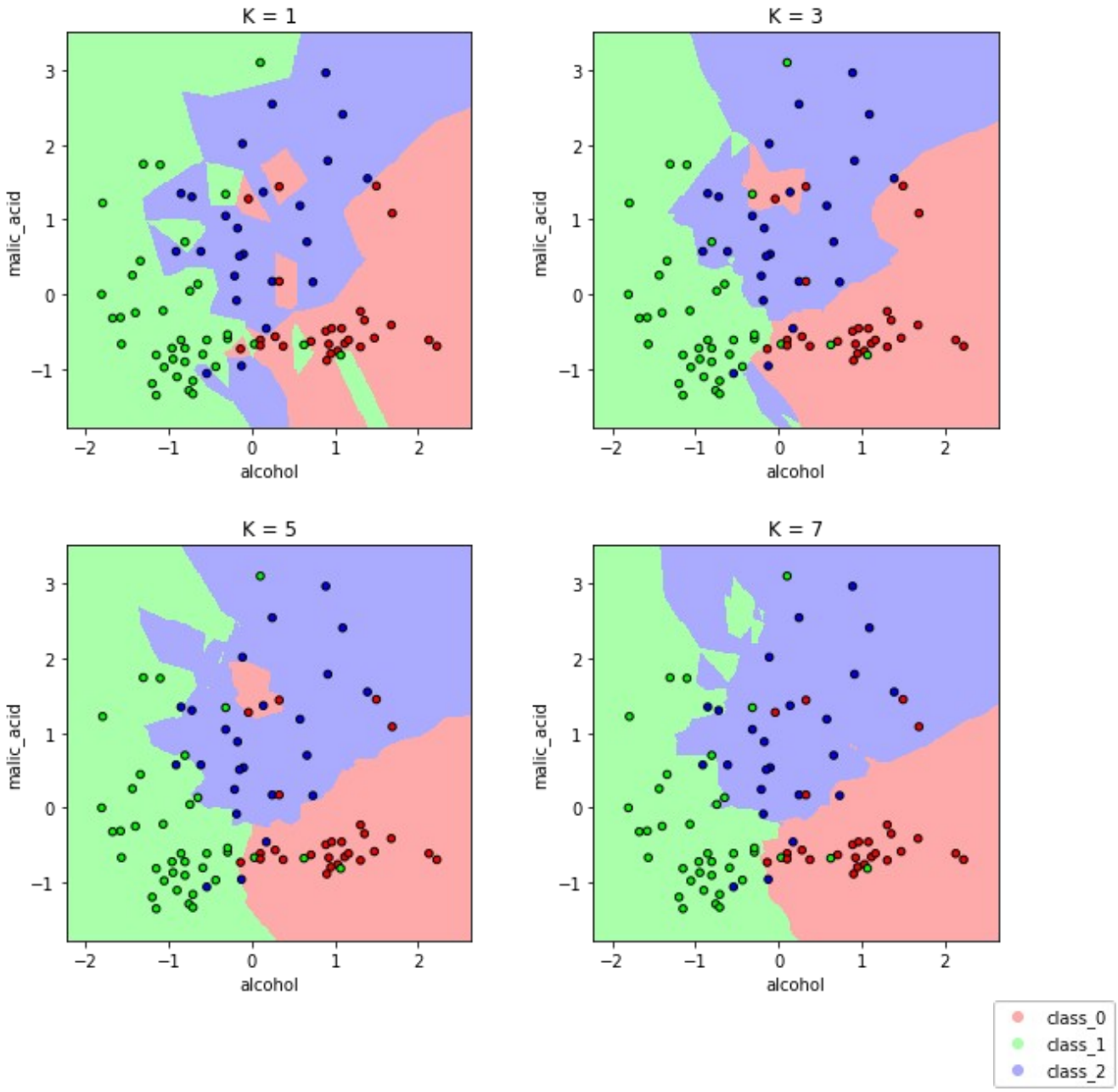

Figure 1: KNN - Decision Boundary Visualization

As expected, the boundaries show that the greater K the more stable/smoother is the separation. In fact, with small K the algorithm is more sensitive to outliers and has a bad generalization (is prone to overfitting).
The boundaries are obtained generating a mesh grid, specifying how far apart the points should be (the 'step') and the window in which the visualization is preferred (with a 'bound' distance from the min/max values).

k	1	3	5	7
accuracy	0.75	0.861111	0.916667	0.888889

Figure 2: KNN - Values of K and accuracy

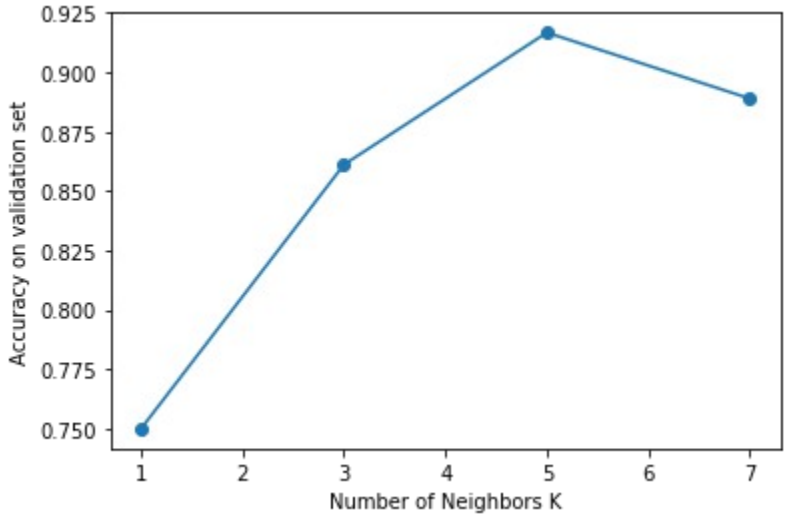

Figure 3: KNN - Model evaluation

The accuracy is calculated as number of correct predictions over the total number of predictions. Using the model learned with K=5 the accuracy obtained on the test set is about 0.7592.

Linear SVM

Fist of all, even if standard SVMs are binary classifiers (the hyperplane separates two classes), in order to construct a multi-class SVM two approaches can be used: the One-vs-One and the One-vs-All.
The former trains N*(N-1)/2 SVMs (where N is the number of classes): one SVM for each pair of classes. For the prediction, the final outcome is assigned thanks to a majority vote among all the outcomes from all the SVMs.
The latter train as many SVMs as the number of classes (the *i*th SVM will see the *i*th class as positive and the others as negatives). As above, the final result if obtained by means of majority vote.
The linear SVM algorithm in scikit-learn implements “one-against-one” multi-class strategy.

With linear SVM, C is the parameter to optimize.

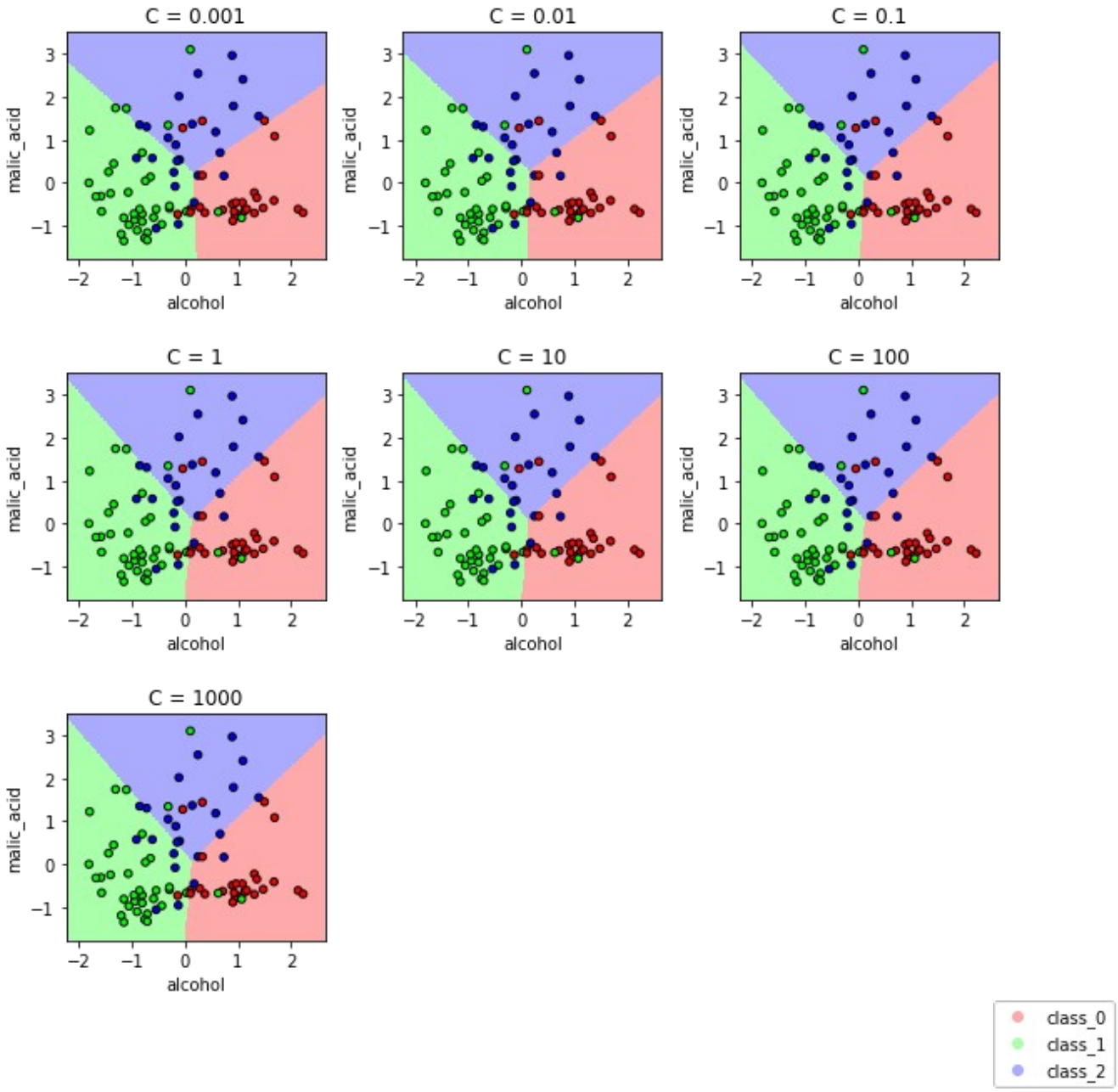

Figure 4: Linear SVM - Decision Boundary Visualization

C is a regularization parameter, that represents the trade-off between smoothness and correctness.
With a large C, more train points are classified correctly and the resulting margin is smaller.
With a small C, the boundary is smoother and the models generalize better (less prone to overfit); in this case the margin is larger.
If C = 0 a hard-margin classifier is obtained, not allowing samples to be misclassified.
In this particular example the differences in boundaries are not much evident, the hyperplanes move only slightly when C varies.

C	0.001	0.01	0.1	1	10	100	1000
accuracy	0.805556	0.805556	0.833333	0.833333	0.833333	0.833333	0.833333

Figure 5: Linear SVM - Values of C and accuracy

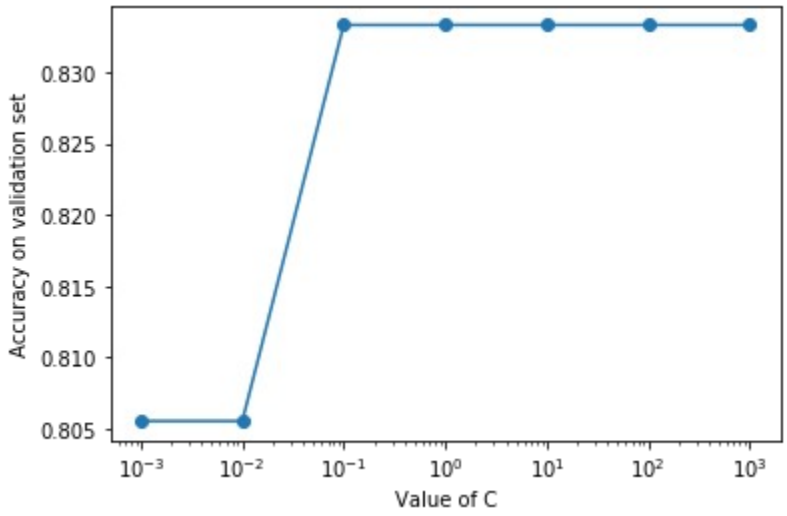

Figure 6: Linear SVM - Model evaluation

As anticipated by the boundary visualization, the accuracy values are close.
C = 10 is chosen, leading to an accuracy on the test set of 0.7592 (same as KNN, with the current value of seed for random number generator).

Kernel SVM

The general SVM algorithm in scikit-learn implements “one-against-one” strategy.
This time the kernel trick is used, since the optimization problem in SVM can be expressed in term of inner product.

The kernel used is the Gaussian/RBF (radial basis function) kernel, hence also the gamma parameter has to be tuned.
Initially only C is tuned, using the parameter gamma=’scale’ (in scikit-learn), which means that gamma is obtained as 1 / (2 * variance of X), where X is the training set.

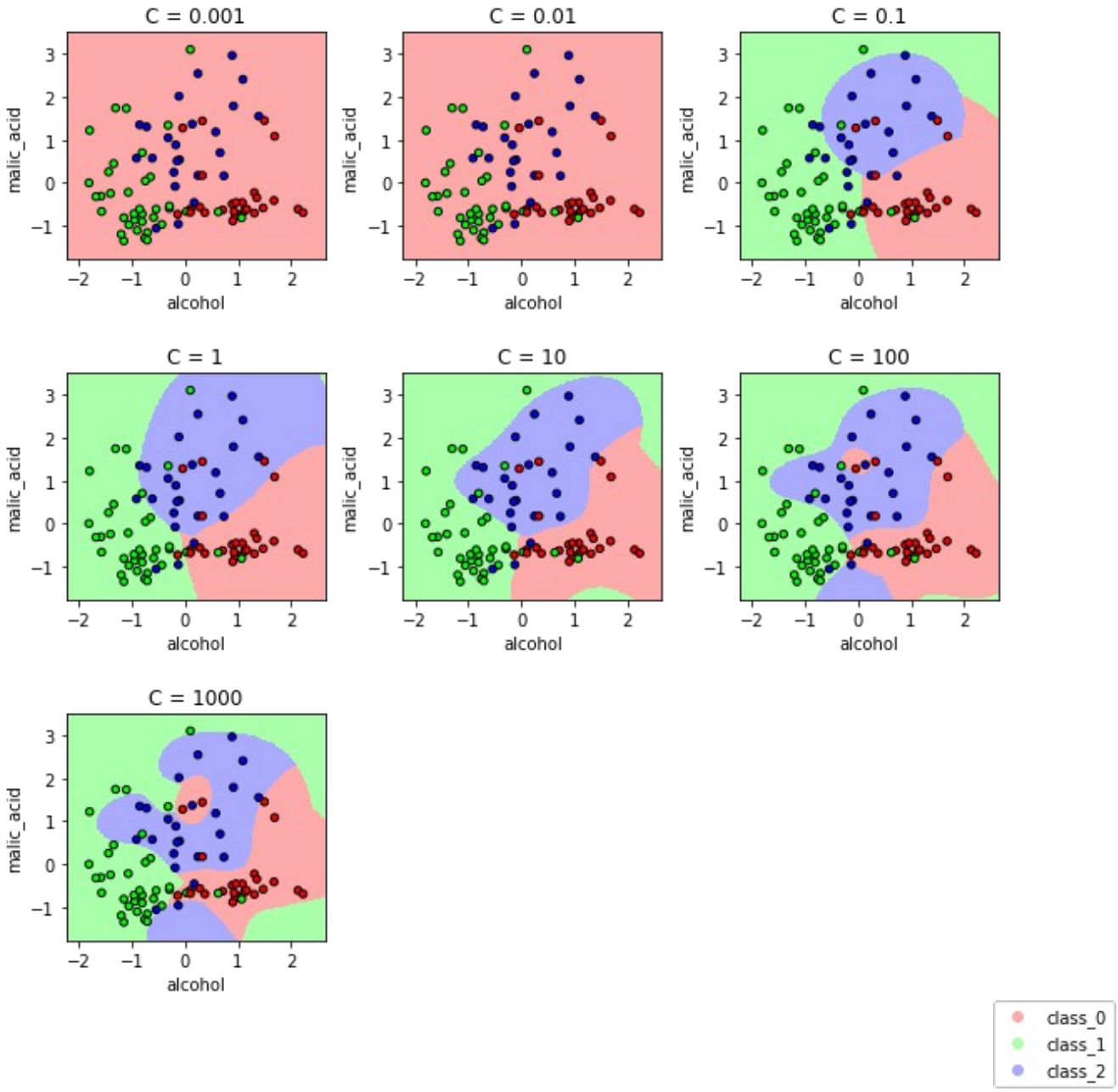

Figure 7: Kernel SVM - Decision Boundary Visualization

As expected, the boundaries are non-linear (linear in a higher space, the one of the kernel transformation).

C	0.001	0.01	0.1	1	10	100	1000
accuracy	0.361111	0.361111	0.861111	0.861111	0.861111	0.861111	0.833333

Figure 8: Kernel SVM - Values of C and accuracy

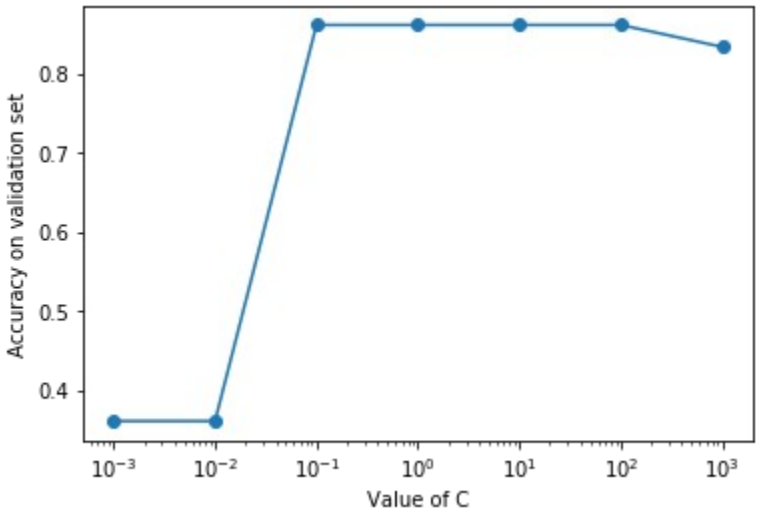

Figure 9: Kernel SVM - Model evaluation

Also this time C=10 is chosen, leading to an accuracy of 0.8333.

Now it is time to also tune gamma, performing a grid search, trying to optimize both parameters. Gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected, the narrower the Gaussian "bell" is (since gamma is inversely proportional to the variance of the Gaussian).

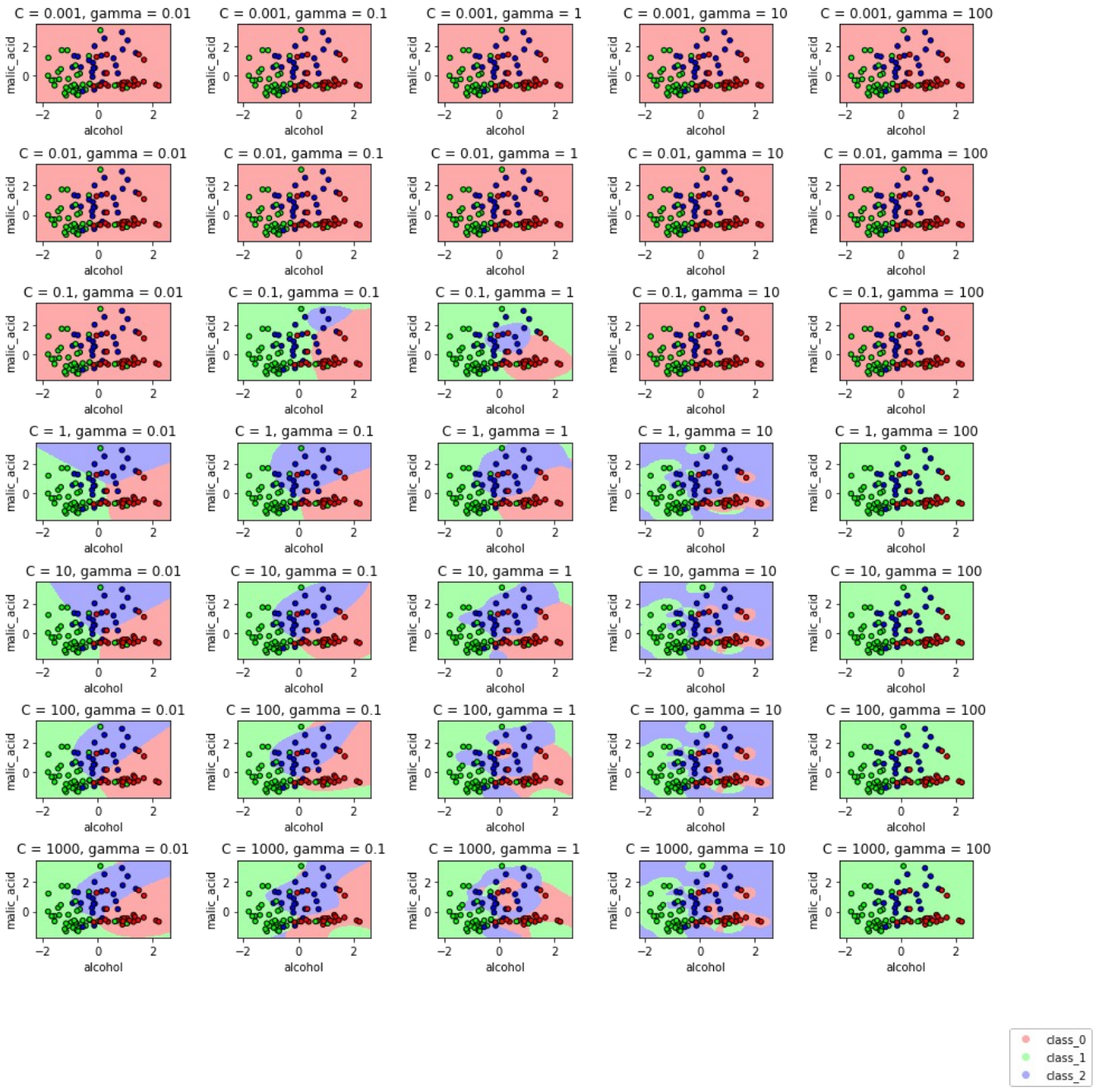

Figure 10: Kernel SVM with grid search - Decision Boundary Visualization

A high value of gamma can lead to overfitting, as can be seen from the boundary plots of the grid search above.

gamma \ C	0.001	0.01	0.1	1.0	10.0	100.0	1000.0
0.01	0.361111	0.361111	0.361111	0.75	0.833333	0.861111	0.805556
0.1	0.361111	0.361111	0.583333	0.888889	0.888889	0.861111	0.861111
1.0	0.361111	0.361111	0.805556	0.861111	0.861111	0.805556	0.666667
10.0	0.361111	0.361111	0.361111	0.805556	0.694444	0.638889	0.638889
100.0	0.361111	0.361111	0.361111	0.611111	0.638889	0.638889	0.638889

Figure 11: Kernel SVM with grid search - Model evaluation

The accuracy obtained using C = 10 and gamma = 0.1 is 0.8148.

Kernel SVM with k-fold validation

The k-fold validation is performed by using a specific class of scikit-learn, 'GridSearchCV'.
It uses a Stratified KFold strategy in splitting the dataset to ensure that the percentage of samples for each class is preserved.

In the notebook a standard KFold validation is also applied (commented in the code), doing the iterations explicitly, just to try and compare the results.
It turns out to be similar to the previous result with stratification, leading to the same optimal parameters of gamma and C.

The number of folds used is 5.

```
The best parameters are {'C': 1000, 'gamma': 0.01} with a score of 0.8303333333333335

Complete list of results
{'C': 0.001, 'gamma': 0.01} test score: 0.4033333333333333
{'C': 0.001, 'gamma': 0.1} test score: 0.4033333333333333
{'C': 0.001, 'gamma': 1} test score: 0.4033333333333333
{'C': 0.001, 'gamma': 10} test score: 0.4033333333333333
{'C': 0.001, 'gamma': 100} test score: 0.4033333333333333
{'C': 0.01, 'gamma': 0.01} test score: 0.4033333333333333
{'C': 0.01, 'gamma': 0.1} test score: 0.4033333333333333
{'C': 0.01, 'gamma': 1} test score: 0.4033333333333333
{'C': 0.01, 'gamma': 10} test score: 0.4033333333333333
{'C': 0.01, 'gamma': 100} test score: 0.4033333333333333
{'C': 0.1, 'gamma': 0.01} test score: 0.4033333333333333
{'C': 0.1, 'gamma': 0.1} test score: 0.7256666666666667
{'C': 0.1, 'gamma': 1} test score: 0.7906666666666667
{'C': 0.1, 'gamma': 10} test score: 0.41966666666666674
{'C': 0.1, 'gamma': 100} test score: 0.4033333333333333
{'C': 1, 'gamma': 0.01} test score: 0.7496666666666667
{'C': 1, 'gamma': 0.1} test score: 0.8146666666666667
{'C': 1, 'gamma': 1} test score: 0.7986666666666667
{'C': 1, 'gamma': 10} test score: 0.7416666666666668
{'C': 1, 'gamma': 100} test score: 0.5883333333333334
{'C': 10, 'gamma': 0.01} test score: 0.806
{'C': 10, 'gamma': 0.1} test score: 0.8066666666666666
{'C': 10, 'gamma': 1} test score: 0.8063333333333335
{'C': 10, 'gamma': 10} test score: 0.7093333333333334
{'C': 10, 'gamma': 100} test score: 0.6203333333333333
{'C': 100, 'gamma': 0.01} test score: 0.8063333333333335
{'C': 100, 'gamma': 0.1} test score: 0.7983333333333333
{'C': 100, 'gamma': 1} test score: 0.7656666666666667
{'C': 100, 'gamma': 10} test score: 0.668
{'C': 100, 'gamma': 100} test score: 0.6203333333333333
{'C': 1000, 'gamma': 0.01} test score: 0.8303333333333335
{'C': 1000, 'gamma': 0.1} test score: 0.7986666666666667
{'C': 1000, 'gamma': 1} test score: 0.7336666666666667
{'C': 1000, 'gamma': 10} test score: 0.668
{'C': 1000, 'gamma': 100} test score: 0.6203333333333333
```

Figure 12: Kernel SVM - k-fold validation

This times the cross-validation suggests to use C=1000 and gamma=0.01 leading to an accuracy on test set of 0.7777.

Differences between KNN and SVM

In the table below (taken from the notebook), a brief recap of the main characteristics of the two algorithm.

	KNN	SVM
intuition	close/similar samples have the same label	learn from most challenging points to separate (support vectors)
problem type	classification (variants for regression)	classification (variants for regression)
linear/non-linear	non-linear classifier	linear classifier (non-linear using kernel trick)
training	fast, no real training	slow, iterate over all the samples to learn the model
prediction	slow, iterate over all the samples (does not scale)	fast, apply the model learned
space complexity	need to store all samples	only support vectors are relevant, no need to keep the whole training set
multi-class	yes	yes, using One-vs-One or One-vs-All approaches
main hyper-parameter	K (number of neighbors)	C (soft-margin parameter, regularization)
other decisions/parameters to choose	distance metric	kernel to use and kernel parameters (gamma if RBF)
curse of dimensionality	is an issue	not affected
resiliance to outliers	sensitive to outliers (less sensitive with larger k)	more robust with larger C ("softer" margins)

Figure 13: KNN / SVM comparison

Other pairs of attributes

In this last part of the homework another pair of attributes is selected.
In particular, a classification based on colors is applied, using color intensity and hue attributes.

This time: first SVM with gamma='scale' and C obtained through parameter tuning is used; then directly k-fold validation is applied.

This experiment is interesting as in this case, with respect to the previous one using the first two attributes, the k-fold validation gives better results compared to holdout validation (0.8333 against 0.7962).
This is valid in general, since k-fold validation usually gives a better indication of how well the model will perform on unseen data, because it is less dependent on the split.

Further details can be checked in the Jupyter notebook.