Machine Learning and Artificial Intelligence: Homework_1

Description of the assignment

The purpose of this work is to use some classification algorithms and compare their results on a toy dataset given by the sklearn library.

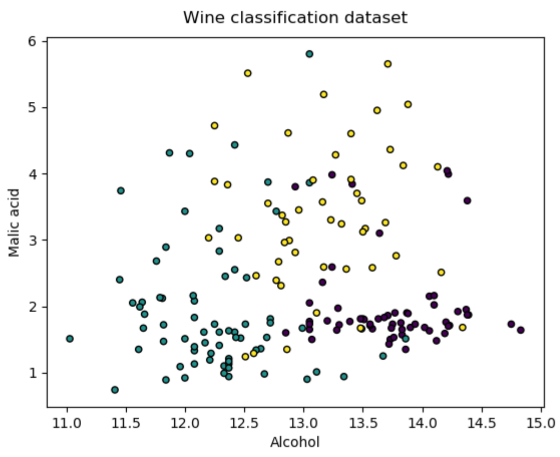
Dataset

The Wine dataset is a preset of sklearn, it is a copy of the UCI ML Wine recognition dataset.
The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are different measurements taken for different constituents found in the three types of wine.

Characteristics:

- 178 rows
- 3 classes
- 13 numeric continuos attributes plus the class label
- No missing attribute values

From the 13 attributes we chose to take the first two, which are respectively *Alcohol* and *Malic acid*



Training procedure

We split data into Train, Validation and Test with 50%, 20% and 30% partition.
Then, we train our model on Train Set, we tune our hyperparameters on Validation Set and finally we evaluate performance on the Test set.

Normalization

Both SVM and K-Nearest Neighboors are critical with respect to distances measurament, therefore I decided to give the option to apply the classifiers on normalized data to be able to see the difference between the two scenarios.
For the normalization procedure I've used `sklearn.preprocessing.StandardScaler` which standardize features by removing the mean and scaling to unit variance fitted on the train+validation set.

$$z = (x - u)/s$$

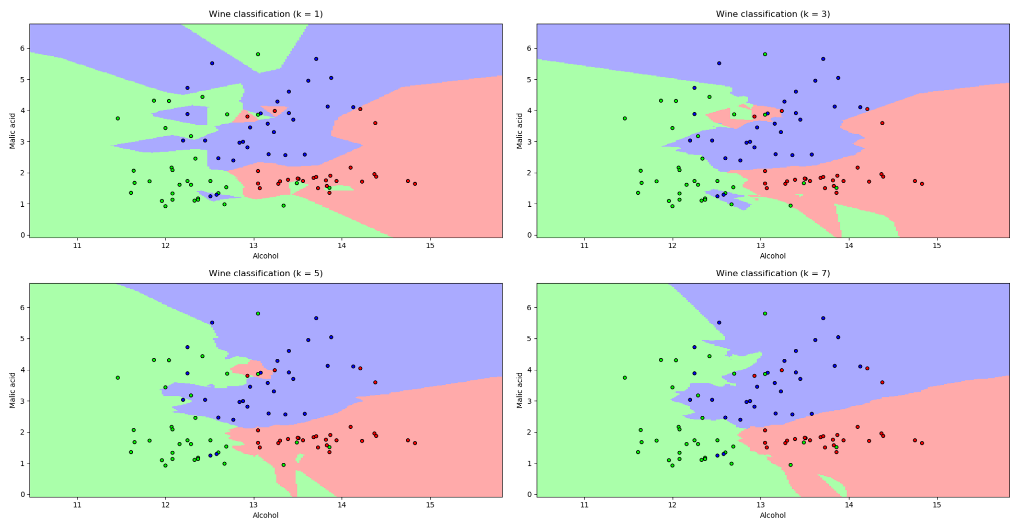
Disclaimer: After I tried different random seeds for the splitting, I noticed the small size of the dataset influence a lot the results of the classifiers, with varying accuracy from 70% to 90%.
So, even if normalization is suggested for kNN and SVM, with this dataset it doesn't affect always the results and for some seeds normalization can reduce the accuracy of the classifier.

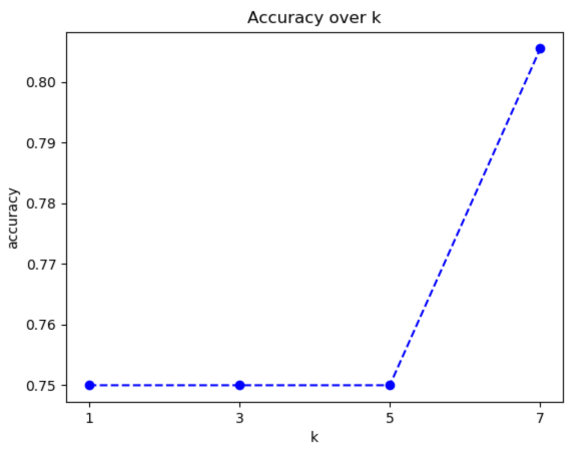
K-Nearest Neighboors Classifier

The intuition is to classify a new entry calculating the *distances* from this point to its nearest *k-points* and assigning the label according to major voting in its k-neighborhood.
So, the critical parameters to choose are *k* and the *distance* metric.
I used for the K-NN algorithm `sklearn.neighbors.KNeighborsClassifier` and trained different model with k = [1,3,5,7] and the minkowski distance metric

$$(\sum_{i=1}^{n} |X_i - Y_i|^p)^{1/p}$$

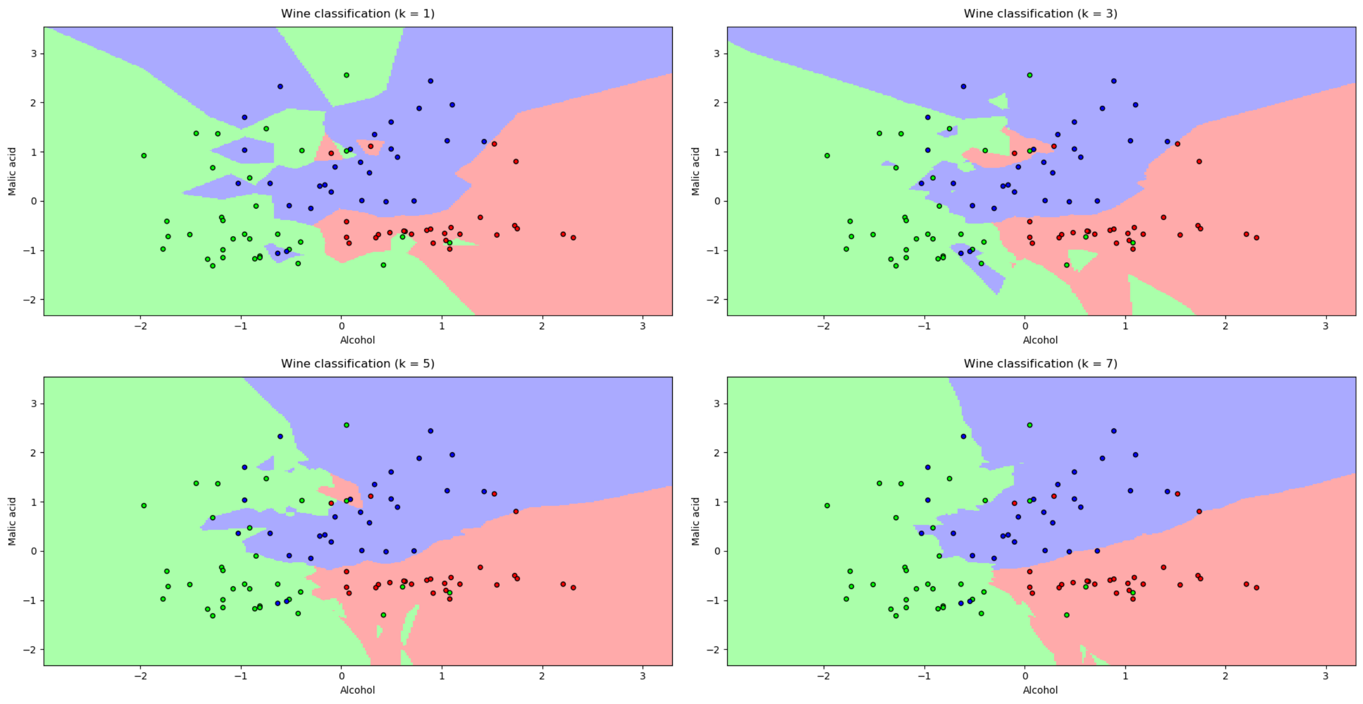
that in 2-dimensions(p=2) is equal to Euclidean distance.
Choosing *k* is crucial, too little and the classifier is too sensible to outliers, too large and the classifier is not able to generalize enough.
For example with k=1 we can see that the decision boundaries highlight the outliers and the classification is compromised.

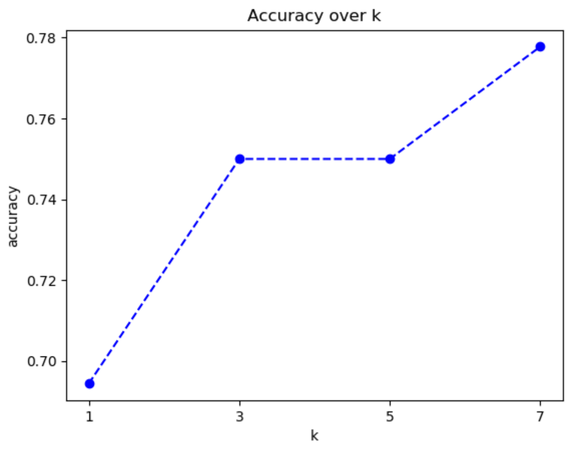




According to our validation process the best value for k is 7 with an accuracy of 80.5%.
Then, we obtain an accuracy on test set of 81.5%.

Normalization is suggested for kNN. In fact, in this case, the result on test set is a little bit higher with a 81.5% accuracy.



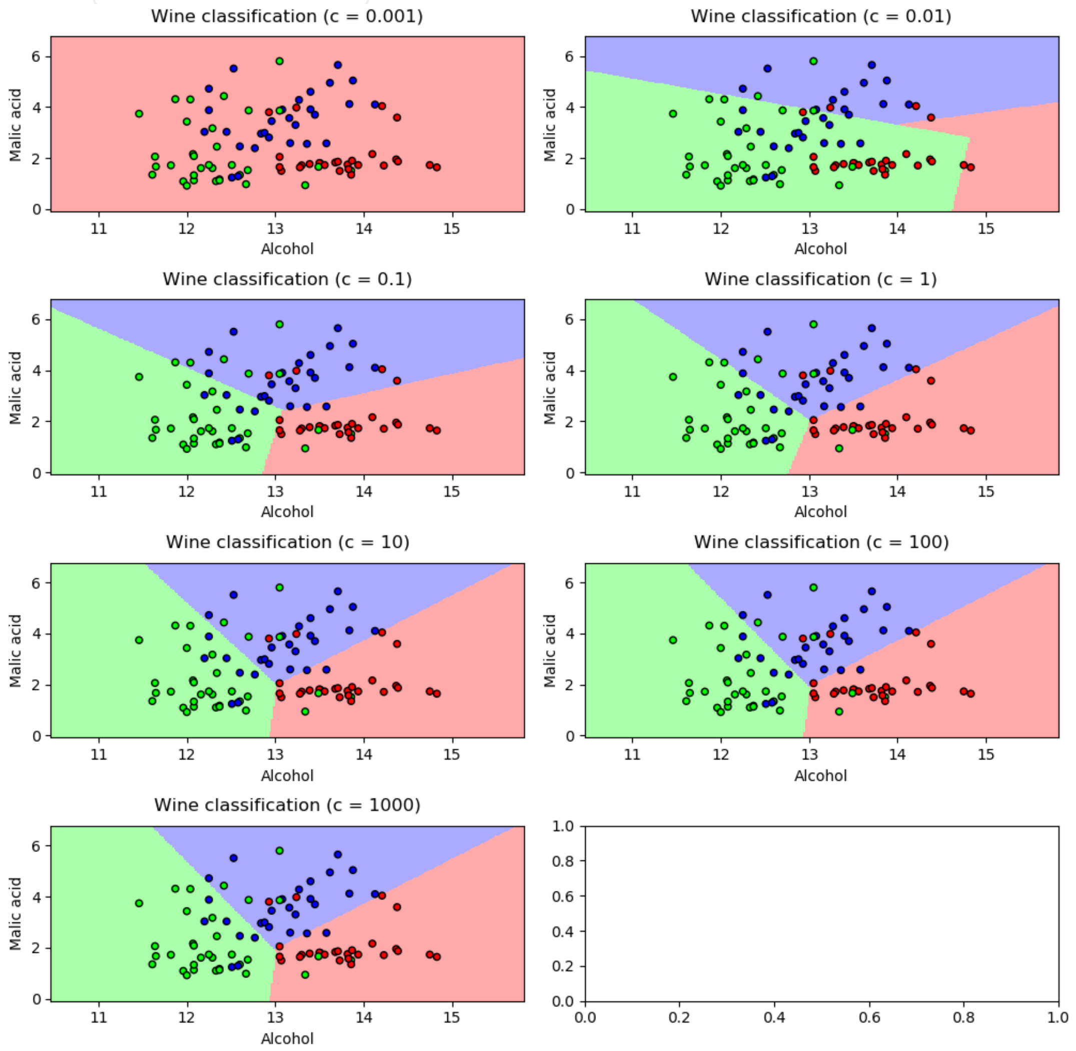


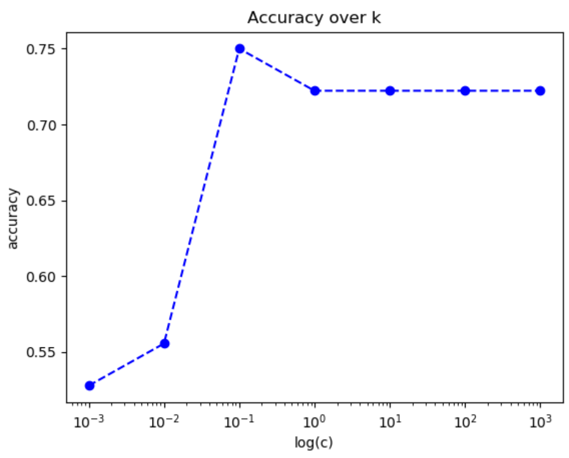
SVM-Linear

The intuition here is: if the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the “margin”, and the maximum-margin hyperplane is the hyperplane that lies halfway between them. If we have have more than 2 classes, this operation is performed for each couple.

Sometimes our data are not linearly separable, so we can define a slack variable that, roughly, indicates how much we must move our point so that it is correctly and confidently classified. When we introduce these variable we must decide how much we penalize these variables, so that our classification is not falsed too much by the use of them.
The C hyperparameter expresses how much we penalize slack variables, the higher the value the higher is the penalization.
I trained model with different C ([0.001,0.01,0.1,1,10,100,1000]) to see how our dataset change according to this parameter.

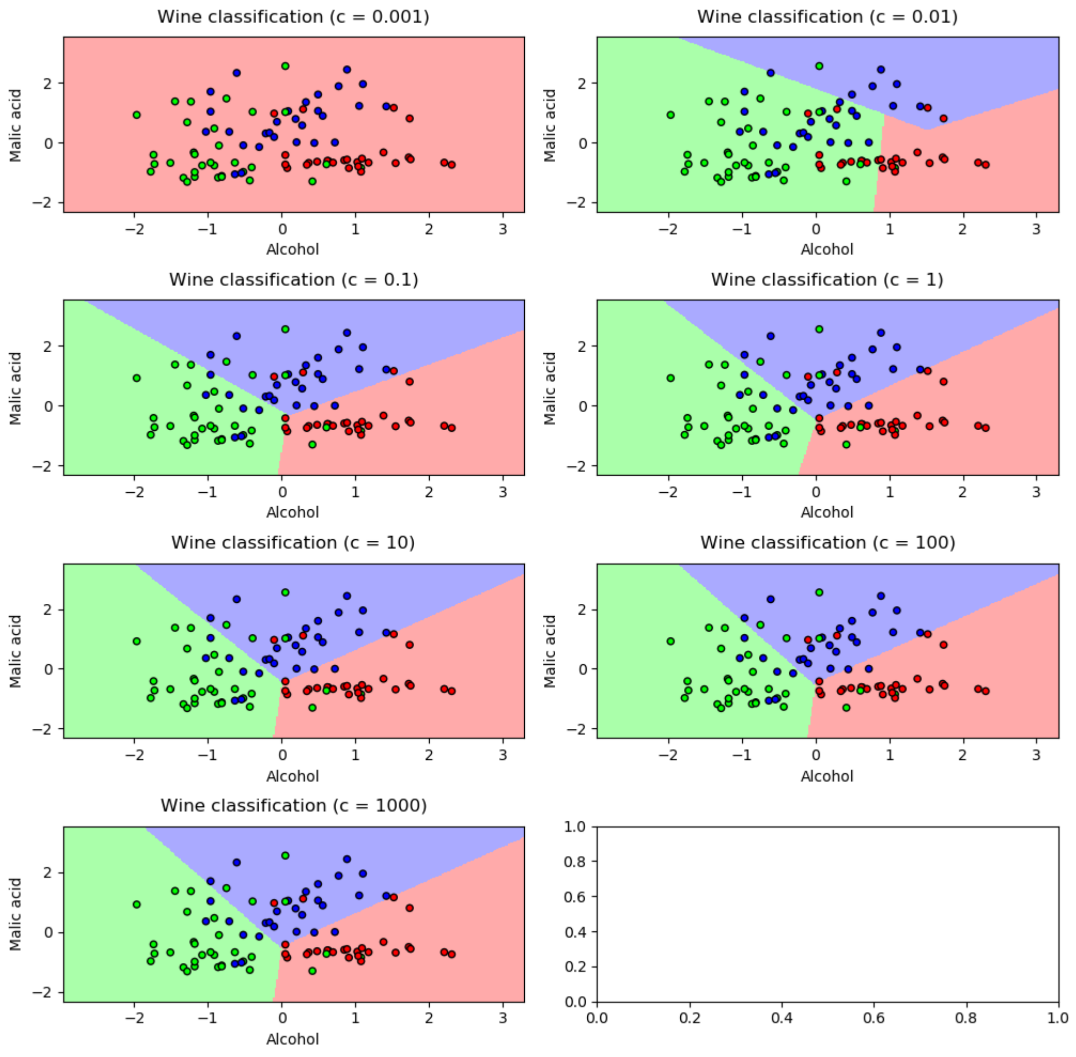
I used `sklearn.svm.SVC` with kernel = “linear”.

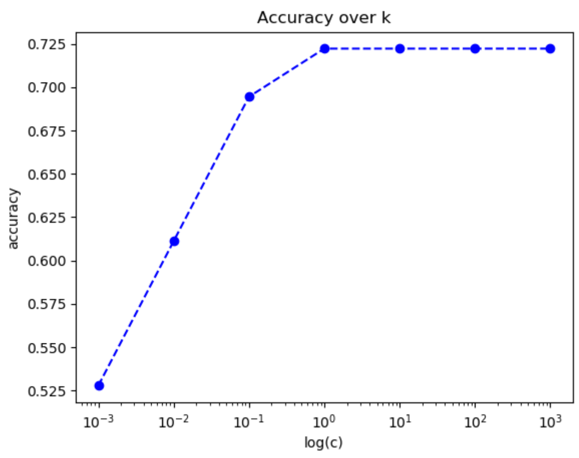




The best accuracy on validation set is obtained with C=0.1 (75.0%) and on our test set we peak to 85.1%. It is interesting to see that if we choose low values of C our boundaries are wrong because the relaxation are too strong.

For SVM normalization is crucial, because the core is base on distance measument.
But, this time with C=1 we got the same performance on train set with 85.1% accuracy.





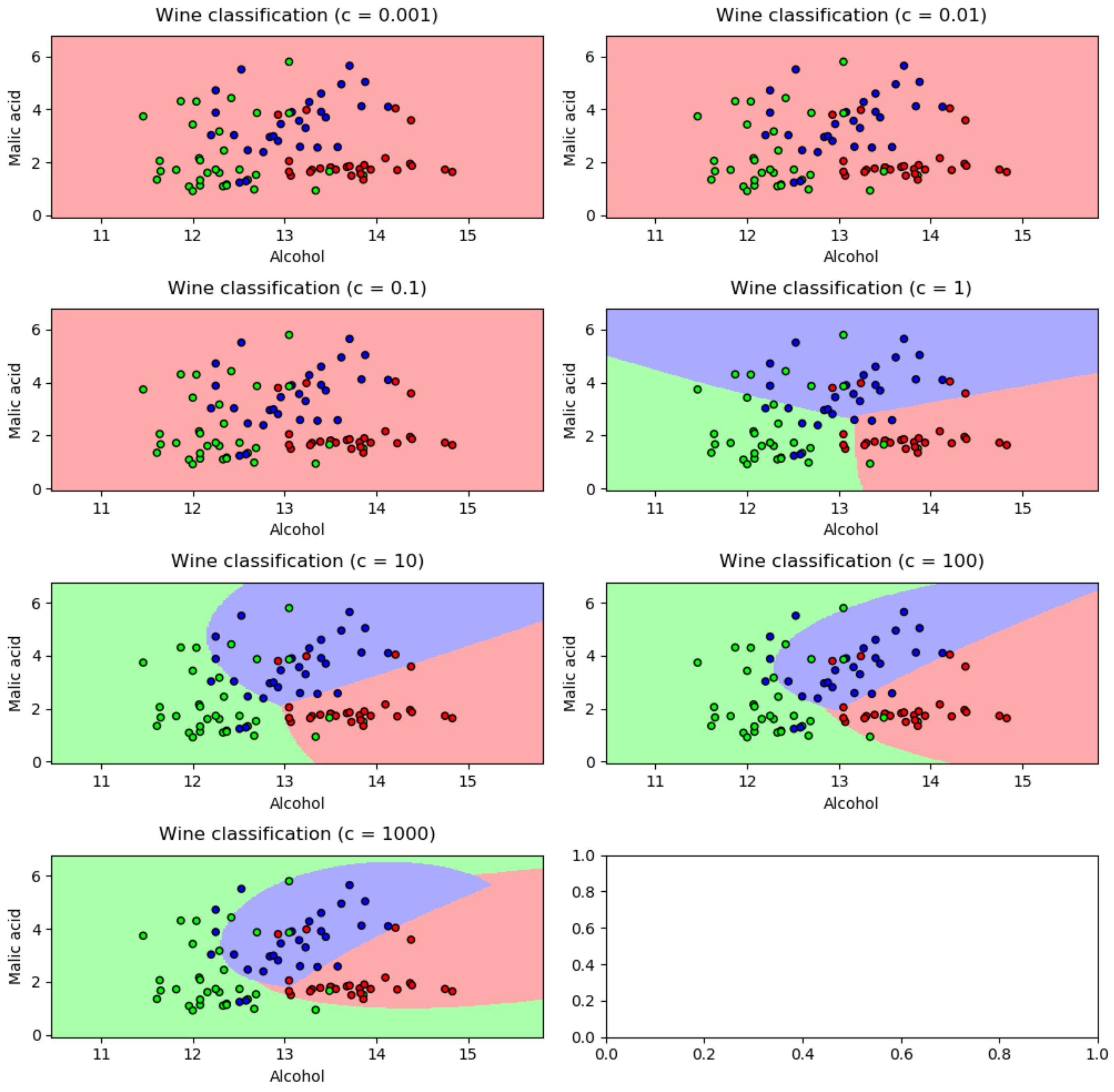
SVM-RBF

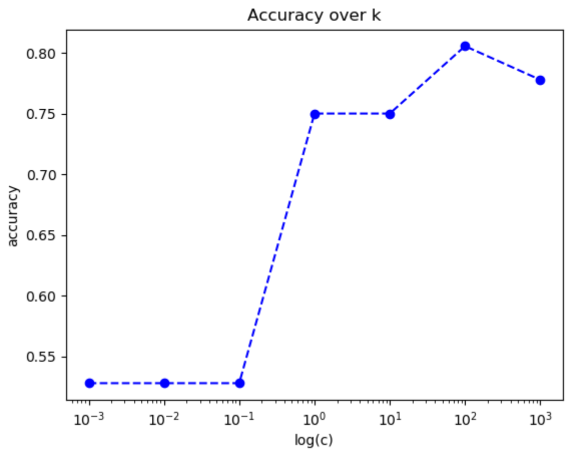
In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the *kernel trick*, implicitly mapping their inputs into high-dimensional feature spaces. The kernel trick avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary, we can define an arbitrary kernel function k(x_n,x_m) that we can substitute for our inner products when we are learning the SVM.
RBF kernel:

$$K(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma^2}\right)$$

As with the linear SVM we train our model for C = [0.001,0.01,0.1,1,10,100,1000] and gamma set to *scale*. Gamma is the hyperparameter which express how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.
scale is equal to:

$$gamma = 1/(n\ features\ * X\ var\ )$$

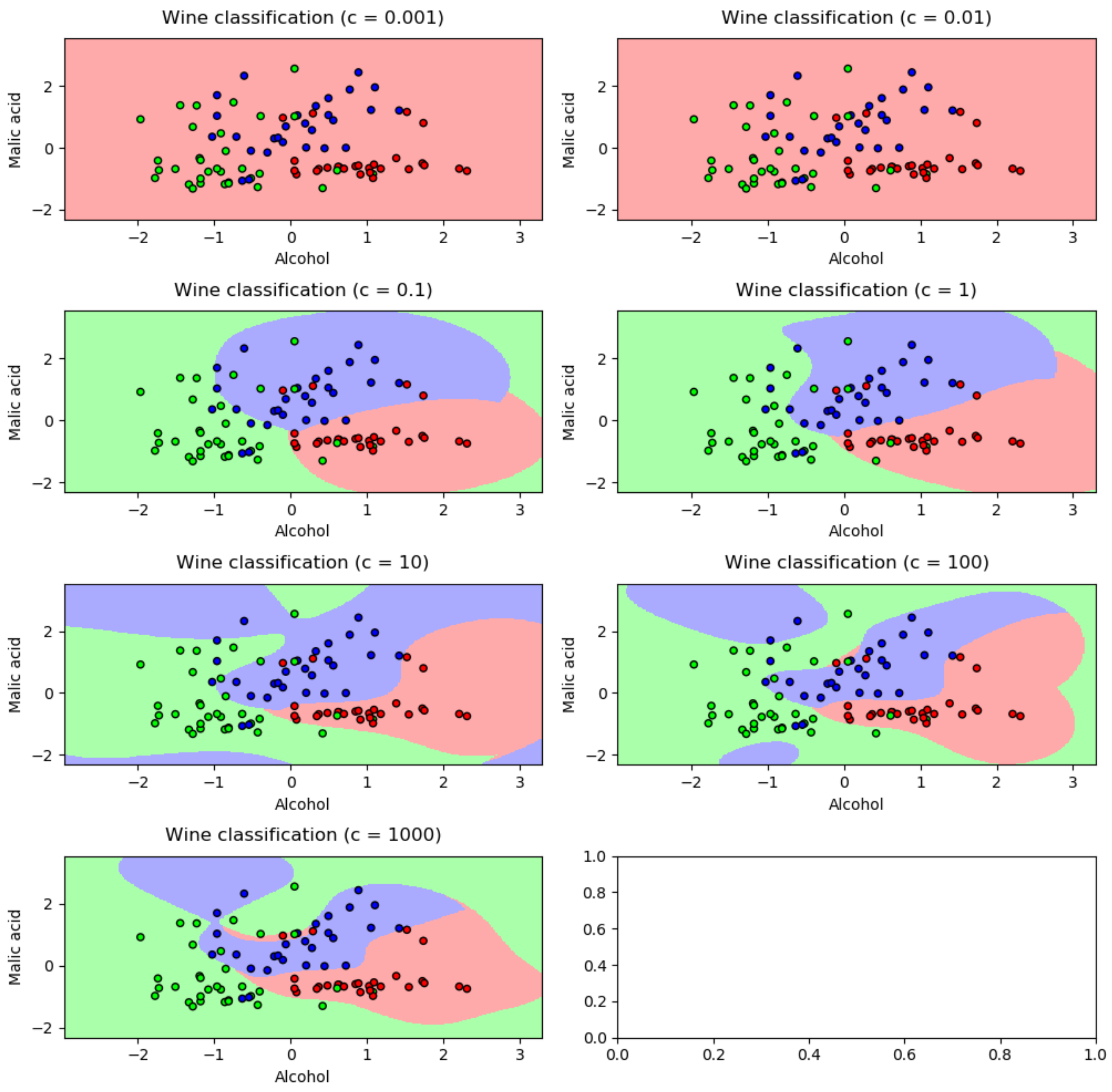


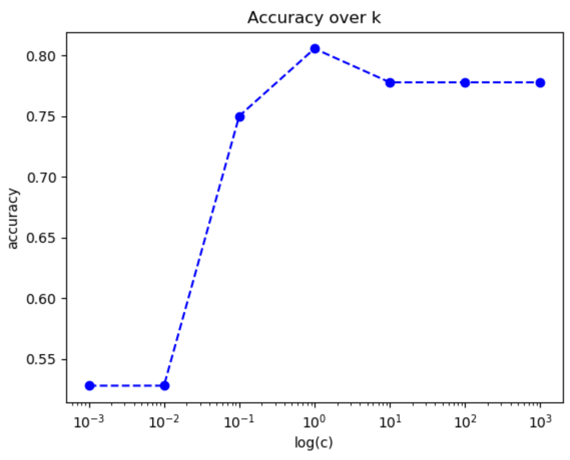


The boundaries are changed, in fact we can clearly obser that with C = 1000, for example, the shape is spheric which is not possible with linear kernel.
The highest accuracy on validation is with C = 100 (80.5%), and this model has a 85.1% accuracy on test

set.

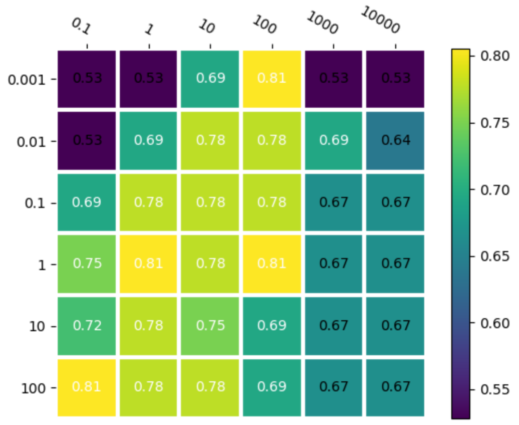
Normalizing, we obtain the higher accuracy on validation set(83.3%), and a little bit lower accuracy on test set with C=1 (83.3%).





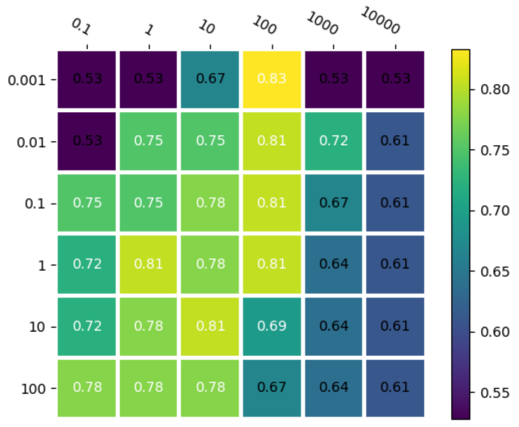
GridSearch

When we have more than one hyperparameter to tune is suggested to try each combination.
For example, with RBF kernel both gamma and C have to be tuned in order to reach the best result.
I've trained several models with all the combinations of C= [0.1, 1, 10, 100, 1000, 10000] and gamma = [0.001, 0.01, 0.1, 1, 10, 100]



We can get that gamma too low is not able to shape correctly the dataset and gamma too high overfits the data. The best accuracy on validation set is obtained with C=0.1 and gamma=1 (80.5%), this model reaches 85.1% on test set.

With normalization the best model is with C=0.1 and gamma=1 (83.3% accuracy on validation), than 83.3% on test set.



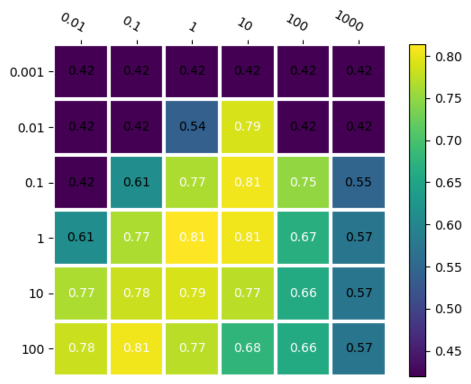
GridSearch and K-Fold

Usually when the dataset is small, as it is ours, it is a good choise to use cross-validation. K-Fold is the most common cross validation method and it is one among the best approach if we have limited data because it ensures that every observation from the original dataset has the chance of appearing in training and test set.

It consists in:

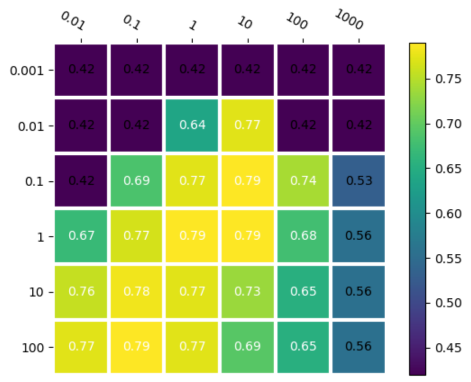
- Shuffle the dataset randomly.
- Split the dataset into k groups
    - For each unique group:
    - Take the group as a hold out or test data set
    - Take the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

In our case k is choosen to be 5.
I used `sklearn.model_selection.GridSearchCV` , which permits to do the k-fold alongside the combination of all parameters.



The best model is with C=10 and gamma=0.1, it reaches 81.4% accuracy on validation set and 85.19% on test set

With normalization, implemented with `sklearn.pipeline.Pipeline` to guarantee a correct normalization process in each k-fold iteration, we grant ourselves better result on validaion set 83.3% and same on test set.



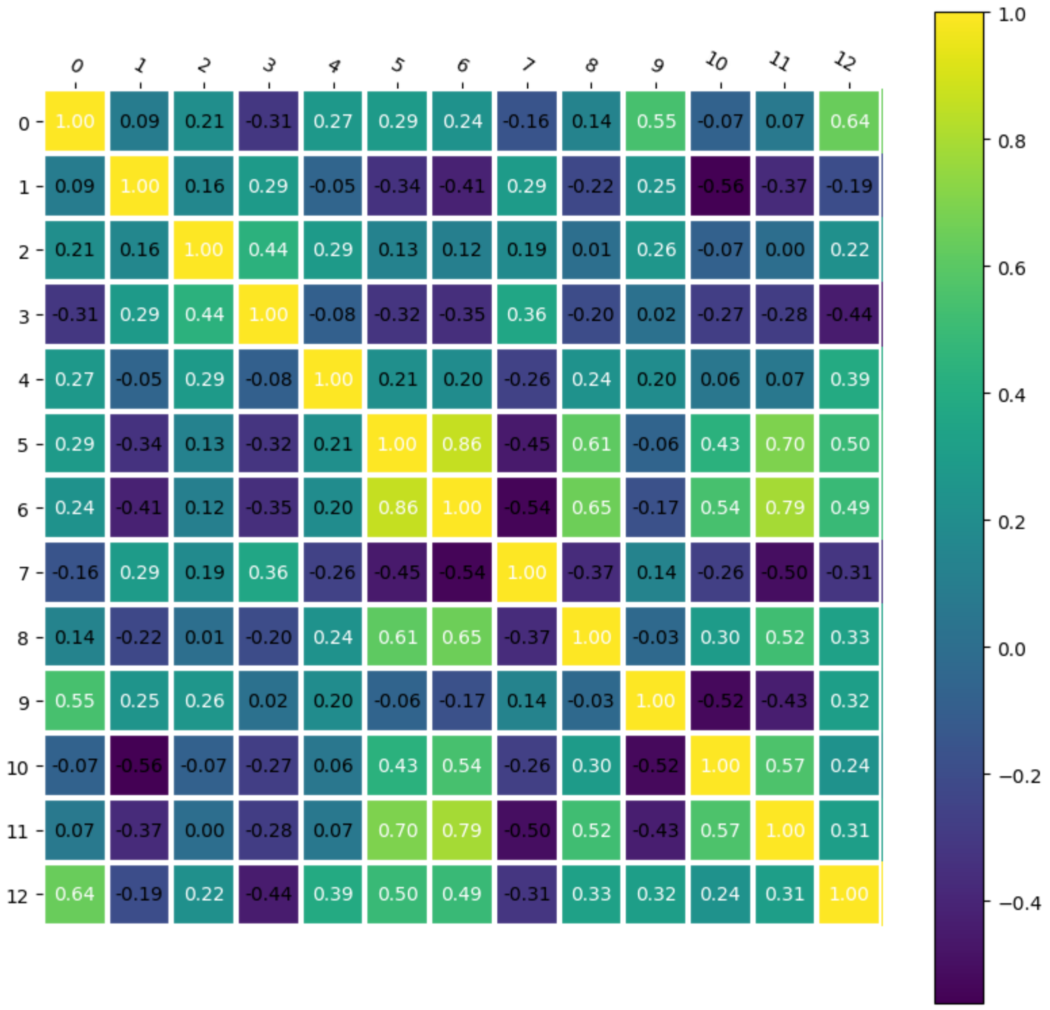
SVM or kNN

For this dataset the best option is to use SVM with linear of RBF kernel, it is able to shape better data and it

is way faster to classify once we have trained the model.
kNN is easy to tune, support multi-class classification easily and it doesn't need a proper training phase but each time we want to classify a entry we need to compute the distance of each point from the entry and this is computationally expensive.

Different attributes

Since now, we have tried our model with the first 2 attributes, but what if we try with different attributes? For better choosing I decided to plot the correlation matrix.



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1.00	0.09	0.21	-0.31	0.27	0.29	0.24	-0.16	0.14	0.55	-0.07	0.07	0.64
1	0.09	1.00	0.16	0.29	-0.05	-0.34	-0.41	0.29	-0.22	0.25	-0.56	-0.37	-0.19
2	0.21	0.16	1.00	0.44	0.29	0.13	0.12	0.19	0.01	0.26	-0.07	0.00	0.22
3	-0.31	0.29	0.44	1.00	-0.08	-0.32	-0.35	0.36	-0.20	0.02	-0.27	-0.28	-0.44
4	0.27	-0.05	0.29	-0.08	1.00	0.21	0.20	-0.26	0.24	0.20	0.06	0.07	0.39
5	0.29	-0.34	0.13	-0.32	0.21	1.00	0.86	-0.45	0.61	-0.06	0.43	0.70	0.50
6	0.24	-0.41	0.12	-0.35	0.20	0.86	1.00	-0.54	0.65	-0.17	0.54	0.79	0.49
7	-0.16	0.29	0.19	0.36	-0.26	-0.45	-0.54	1.00	-0.37	0.14	-0.26	-0.50	-0.31
8	0.14	-0.22	0.01	-0.20	0.24	0.61	0.65	-0.37	1.00	-0.03	0.30	0.52	0.33
9	0.55	0.25	0.26	0.02	0.20	-0.06	-0.17	0.14	-0.03	1.00	-0.52	-0.43	0.32
10	-0.07	-0.56	-0.07	-0.27	0.06	0.43	0.54	-0.26	0.30	-0.52	1.00	0.57	0.24
11	0.07	-0.37	0.00	-0.28	0.07	0.70	0.79	-0.50	0.52	-0.43	0.57	1.00	0.31
12	0.64	-0.19	0.22	-0.44	0.39	0.50	0.49	-0.31	0.33	0.32	0.24	0.31	1.00

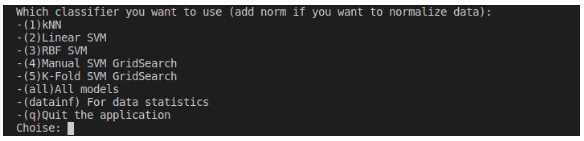
The best choise is to take 2 attributes with correlation near 0, because generally it means we can exploit the maximum quantity of information from them. We can notice that the first and second, our previous choise, were a good choise (0.07 correlation).
Attribute 0 and 11 have correlation equal to 0.07, so we are gonna try with these two.
This table shows the result of the model choosen by the validation process when permorms on test set.

Classifier	Parameter	Normalization	Accuracy on test set
kNN	k=3	N	0.944
kNN	k=3	Y	0.944
SVM-Linear	C=100	N	0.963
SVM-Linear	C=1	Y	0.944
SVM-rbf	C=1000 gamma=scale	N	0.944
SVM-rbf	C=0.1 gamma=scale	Y	0.963

We can notice that choosing the right attributes to analyze is as import as tuning a model.
Since, it is computationally expensive to use all attributes, for linear dimensionality reduction is commonly used Singular Value Decomposition of the data to project it to a lower dimensional space and so having good result using just few attributes.

Note to code

I decided to build the code into two modules, one(homework1.py) is the core of the program with all the classifiers and the possibility of choosing the preferred one with or without normalization, the other one (utils.py) is composed of plotting utility functions.



```
Which classifier you want to use (add norm if you want to normalize data):
-(1)kNN
-(2)Linear SVM
-(3)RBF SVM
-(4)Manual SVM GridSearch
-(5)K-Fold SVM GridSearch
-(all)All models
-(datainf) For data statistics
-(q)Quit the application
Choise: 
```