Artificial Intelligence and Machine Learning
Prof. Tatiana Tommasi

Homework I

Nearest Neighbors / SVM

Luigi Pirisi
s254989

Master Degree Course in
Data Science (Computer Engineering)
Politecnico di Torino
Academic Year 2019/2020

Contents

List of Figures

Listings

1. Introduction

The goal of the homework is to analyze the the behaviour of KNN and SVM in some specific conditions. More to the point: the way the data is supplied to the model will be changed, or the accuracy will be studied with the variation of certain model parameters.

2. The dataset

The dataset used is **Wine**. It contains 3 class described by 13 biological features. It was imported and the first two features were selected [*Points 1 and 2*]:

```
from sklearn.datasets import load_wine
X, y = load_wine(return_X_y=True)
X_2Dsel = X[:, :2]  # make a selection of the first 2 coloumns
```

Listing 2.1: *import and selection of the dataset*

2.1 Preprocessing and data splitting

Now the data are randomly splitted into train validation and test sets in proportion 5:2:3.
By setting the *random state* parameter you can control the random splitting in such way that the same split is repeatable. [*Point 3*]

Even if it is not required by the homework, it is good practice to standardize the data to avoid wrong computation. When the feature are different (e.g. different order of magnitude) certain measures can assume an unjustified importance in computing the final result. In this specific case the standardization will not change the output at all probably because the two features are similar in terms of order of magnitude. The fitting phase of the standardization (i.e. the computation of mean and variance) depends on the training and validation set, the test set is kept out of the process. Once the fit of the *StandardScaler* is done, the train plus validation set is actually transformed. The test set will be transformed later by using the same scaler without a new fitting.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_2Dsel, y, test_size
    =0.3, random_state=19)
scaler = StandardScaler()
scaler.fit(X_train)
X_2Dsel = scaler.transform(X_train)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
    test_size=2 / 7, random_state=19)
```

Listing 2.2: *data splitting and standardization*

3. K-Nearest Neighbors

3.1 Overview

The K-Nearest Neighbors (KNN for short) is a classifier that first identifies the K points in the training data that are closest to the point to predict. It then estimates the conditional probability for class j as the fraction of points in the training space $\mathcal{N}_0$ whose response values equal j:

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{j \in \mathcal{N}_0} I(y_i = j).$$

Finally, KNN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.

3.2 Application

At this point KNN model is trained on the train set. K parameter varies in the range [1, 3, 5, 7]. The method needs to specify the *weights* to create a classifier. It is set to *uniform* rather than *distance* which gives more importance to the closer points. [*Point 4a*]

```
from sklearn import neighbors
for n_neighbors in [1,3,5,7]:
...
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights='uniform')
    clf.fit(X_train, y_train)
```

```
6 ...(Plot decision boundaries)
```

Listing 3.1: *KNN classfier*

These are the decision boundaries computed by the model. The points on them belong to the train set.[*Point 4b*]

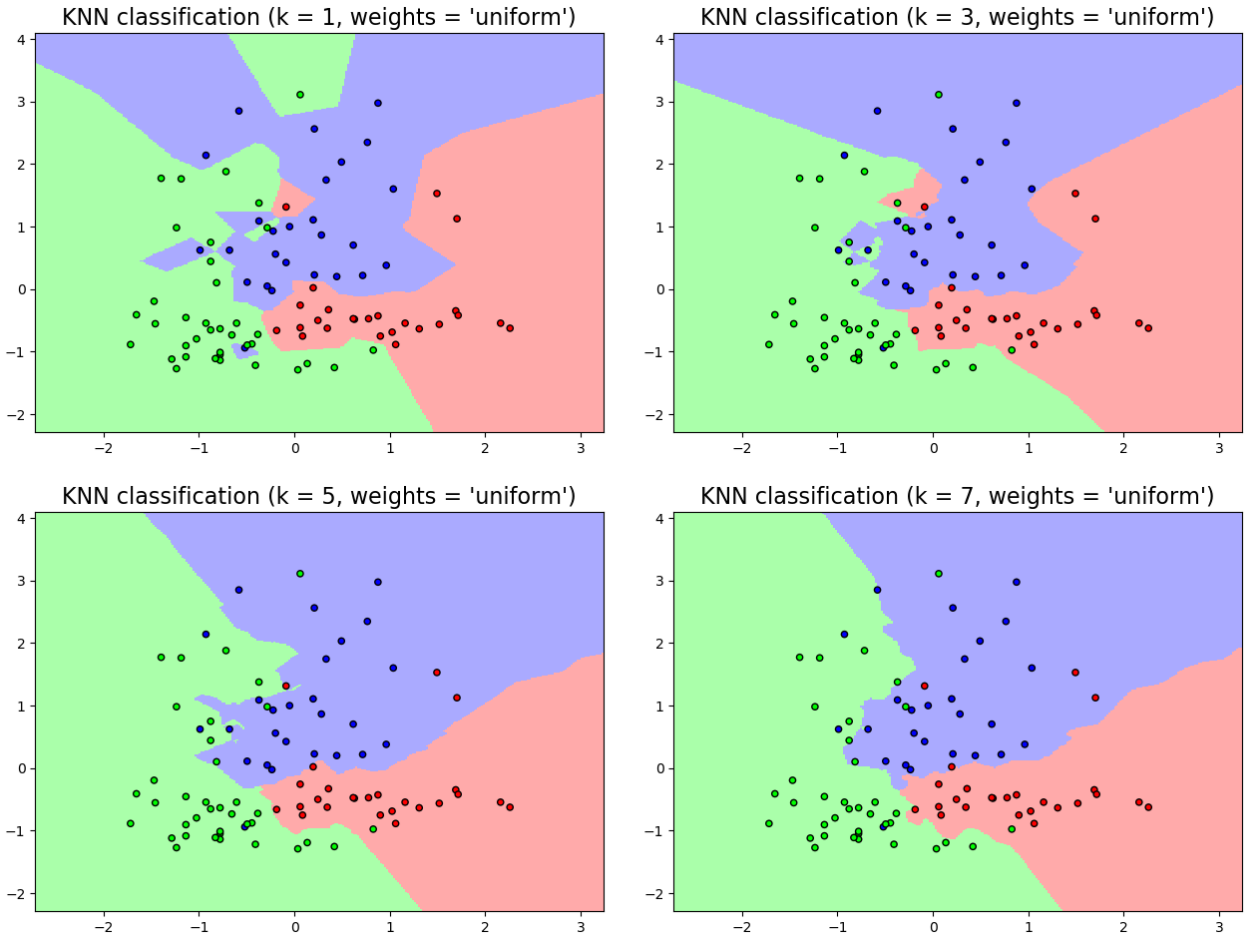

Figure 3.1: *KNN Decision boundaries plot with different k value*

As it shown in the plots, as the parameter k increases, the boundaries shape becomes more regular. This situation depends on the fact that single points *can allocate* their class to the region if and only if the majority of the k closest points belongs to the same class. As a result, for lower values of k class outliers will have more impact.[*Point 6*]

3.3 Evaluation on validation set

At this stage the the labels of the validation set are predicted and compared with the real ones in order to compute the accuracy. This process is made for every k. At the end of this part the k that produce the best accuracy will be chosen.[*Point 4c*]

```
for n_neighbors in [1,3,5,7]:
    if(n_neighbors==1):
            ACCs=[n_neighbors, acc]
        else:
            ACCs = np.vstack([ACCs,[n_neighbors, acc]])

bestK = np.int(ACCs[np.argmax(ACCs[:,1])][0]) #K that produce max accuracy
```

Listing 3.2: *KNN - computation of accuracy on validation set*

[*Point 5*]

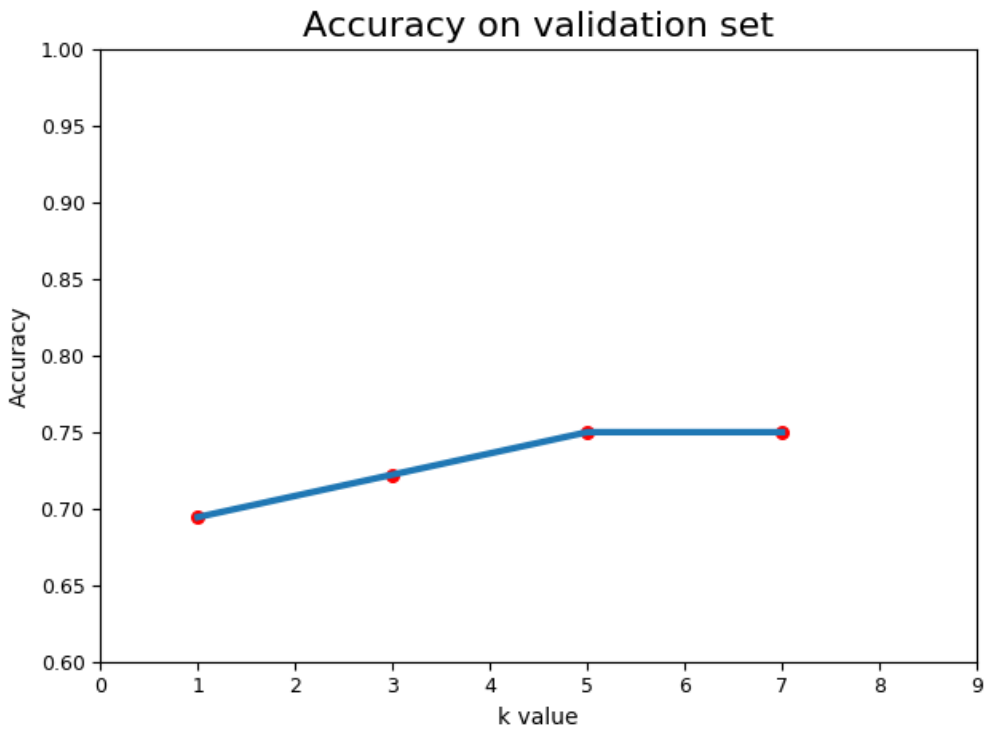

Figure 3.2: *KNN accuracy on validation set*

Here is the output of the program:

```
The Best k found is 5
The accuracy on the validation set with k=5 is 75.00%
```

3.4 Testing the classifier

Once the best parameter is found, the corresponding classifier can be used to compute the accuracy of label prediction. Here is the output:[*Point 7*]

```
The accuracy on the test set with the best k is 81.48%
```

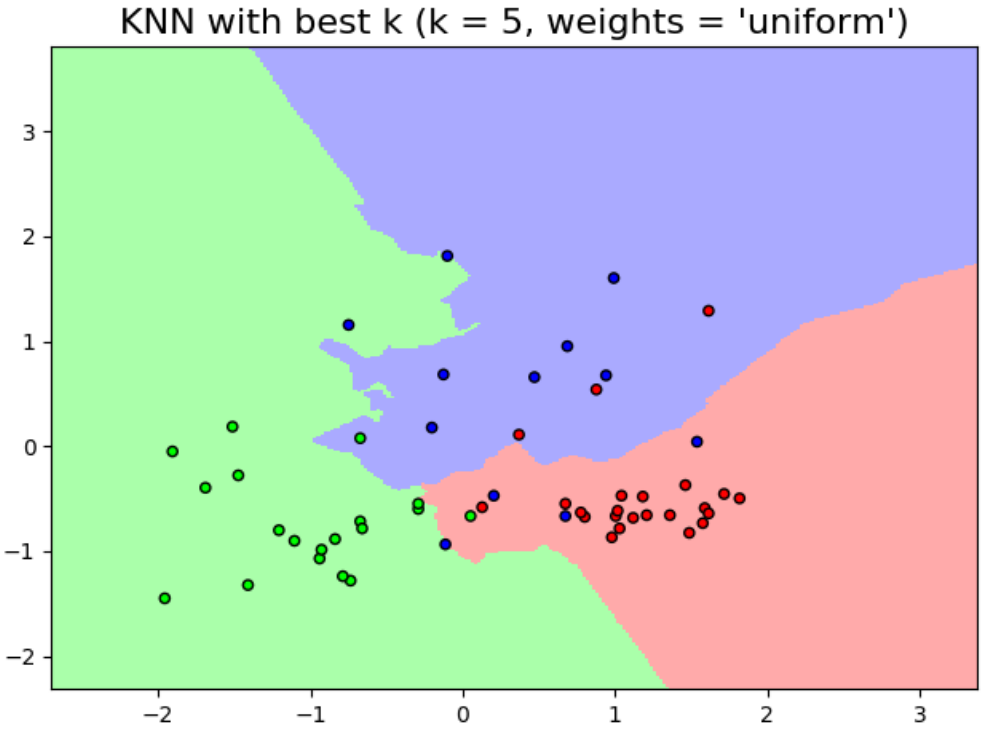

Figure 3.3: *KNN Decision boundaries for best k plot with test points*

4. Support Vector Machine

The support vector machine (SVM) is an approach for classification that was developed in the computer science community in the 1990s and that has grown in popularity since then. SVMs have been shown to perform well in a variety of settings, and are often considered one of the best "out of the box" classifiers. The support vector machine is a generalization of a simple and intuitive classifier called the maximal margin classifier. Though maximal margin classifier is elegant and simple, it's unfortunately cannot be applied to most data sets, since it requires that the classes be separable by a linear boundary. Support vector classifier is an extension of the maximal margin classifier that can be applied in a broader range of cases. Finally the support vector machine, which is a further extension of the support vector classifier comes in order to accommodate non-linear class boundaries. Support vector machines are intended for the binary classification setting in which there are two classes. When the classes are more than two, the SVM can simplify the problem as if it is a binary classification of one class versus all the others.

4.1 Overview

As previously announced, Support Vector Machines comes from the Support Vector Classifier, wich in turn derive from the Maximal Margin Classifier.

4.1.1 Separating hyperplane

The main goal of the classifier is to find the optimal *separating hyperplane* that split two classes of points. In a p-dimensional space an hyperplane is defined by

the equation:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0. \tag{4.1}$$

So, if a point lies on the hyperplane, it will satisfy 4.1; otherwise, assuming that the points $x_1, ..., x_n$ belong to a class $y_1 ... y_n \in \{-1, 1\}$, a separating hyperplane has the property that

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) > 0 \tag{4.2}$$

for all $i = 1, ..., n$.

4.1.2 Maximal Margin Classifier

In general, if our data can be perfectly separated using a hyperplane, then there will in fact exist an infinite number of such hyperplanes. The goal of the maximal margin classifier is to find the optimal separating hyperplane. More formally, we need to:

maximize M

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$,

$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq M \ \forall i = 1, ..., n.$

So M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, .., \beta_p$ to maximize M. In the figure 4.1, on the left side, training observations are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin. These three observation

are known as *support vectors*, since they are vectors in p-dimensional space. Interestingly, the maximal margin hyperplane depends directly on the support vectors, but not on the other observations: a movement to any of the other observations would not affect the separating hyperplane.

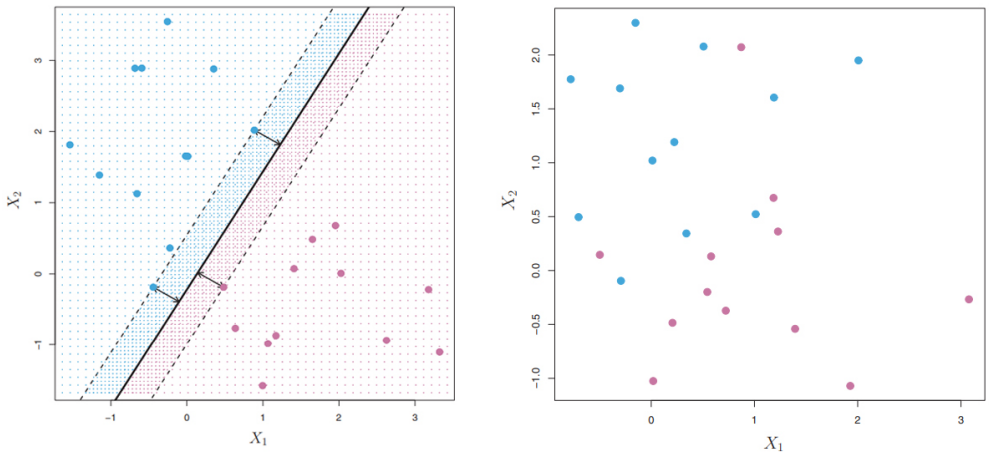

Figure 4.1: On the left side: *two classes of observation separated by a maximal margin hyperplane.* On the right side: *there are two classes of observation that are not separable by an hyperplane*

On the left side it is shown a case in which two classes of observation are not linearly separable so the separating hyperplane doesn't exist and so there is not maximal margin classifier. In this case, the concept of a separating hyperplane can be extended in order to *almost* separate the classes, using a so-called *soft margin*.

4.1.3 Support Vector Classifier

In order to almost separate two classes of observation that are not linearly separable a *soft margin* is introduced to allow some observations to be on the incorrect side of the margin. Observations on the wrong side of the hyperplane correspond to training observations that are misclassified by the support vector classifier. The optimization problem become the following:

maximize M

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \ ,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq U.$$

where U is a nonnegative tuning parameter, $\epsilon_1, ..., \epsilon_n$ are slack variable that allow individual observation to be on the wrong side of the margin or the hyperplane. The slack variable ϵ_i tells where the ith observation is located, relative to the hyperplane and relative to the margin. If $\epsilon_i = 0$ then the ith observation is on the correct side of the margin. If $\epsilon_i > 0$ then the ith observation is on the wrong side of the margin, If $\epsilon_i > 1$ then the ith observation is on the wrong side of the hyperplane. U bounds the sum of the ϵ_i's, and so it determines the number and severity of the violations to the margin (and to the hyperplane) tolerated. For $U = 0$ then $\epsilon_i = 0$ so the classification become a maximal margin one.

4.1.4 Support Vector Machines

The Support Vector Machines were introduced because of the limits of linear classifiers to produce non-linear decision boundaries. The SVM does this in an automatic way. In figure 4.2 it is shown that SVC perform poorly when the boundary is not linear. The *support vector machine* (SVM) is an extension of the support vector support vector machine classifier that results from enlarging the feature space in a specific way, using *kernels*. It turns out that the solution to the support vector classifier problem involves only the *inner products* of the observations (as opposed to the observations themselves).

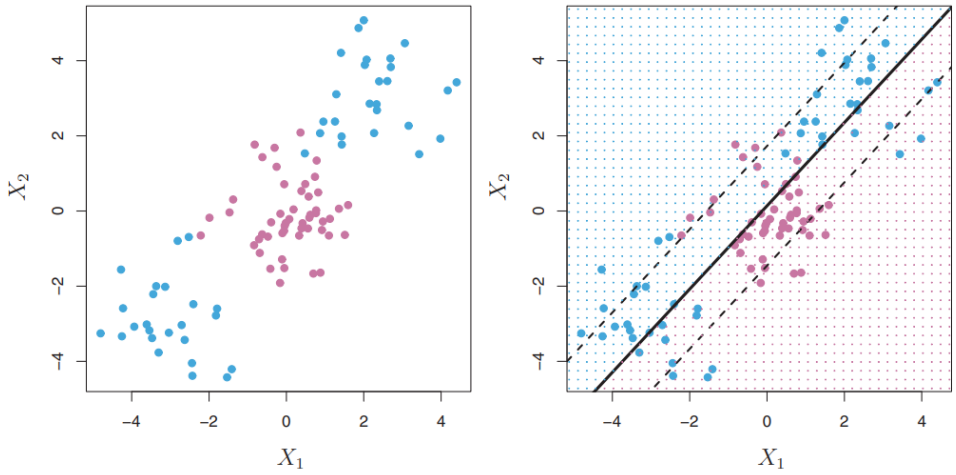

Figure 4.2: Left: *The observations fall into two classes, with a non-linear boundary between them.* Right: *The support vector classifier seeks a linear boundary, and consequently performs very poorly.*

The inner product of two observations $x_i,\ x_i$ is given by

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}. \tag{4.3}$$

The inner product can be equalled by a function K named *kernel* that quantifies the similarity of two observation. So 4.3 become

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}. \tag{4.4}$$

The case of 4.4 is an example of *linear* kernel used by SVC. For the next analysis it will used a *radial* kernel of the form

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} x_{ij} x_{i'j})^2) \tag{4.5}$$

where γ is a positive constant.The Figure 4.3 shows an example of an SVM with a radial kernel on this non-linear data

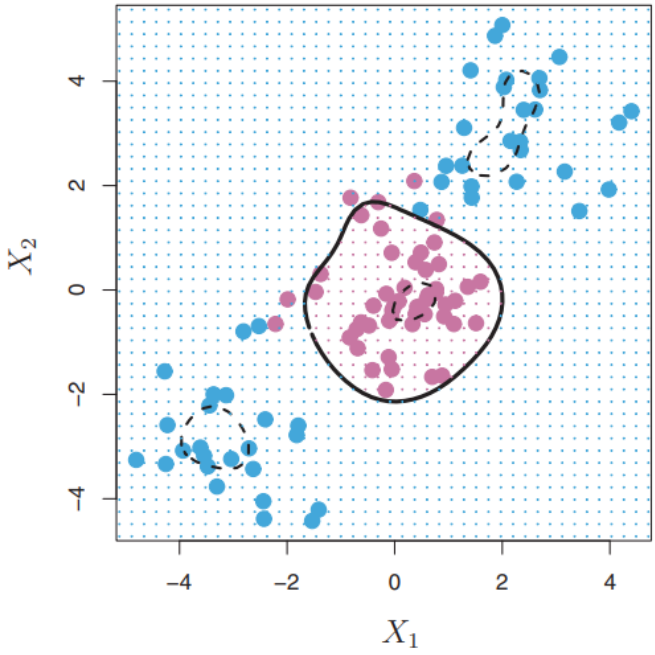

Figure 4.3: *An SVM with a radial kernel is applied.*

4.2 Linear Kernel SVM

4.2.1 Application

At this point the Support Vector Classifier (or linear SVM) is trained on the train set.[*Point 8a*]

```
from sklearn import svm
from sklearn.svm import SVC
for c in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
    clf = svm.SVC(kernel='linear', C=c)
    clf.fit(X_train, y_train)
```

Listing 4.1: *SVC classfier*

In figures 4.4 and 4.5 and are the plots of the decision boundaries for each C. As it shown in the figure 4.4 The first iteration (the one with $C = 0.001$) perform really poorly while the other iterations perform better and similar to each other.[*Point 8b*]

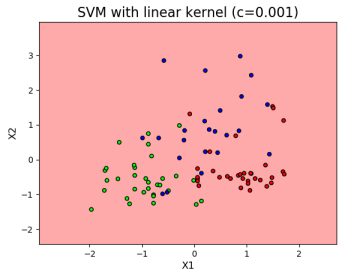

Figure 4.4: *Linear kernel SVM Decision boundaries for C=0.001.*

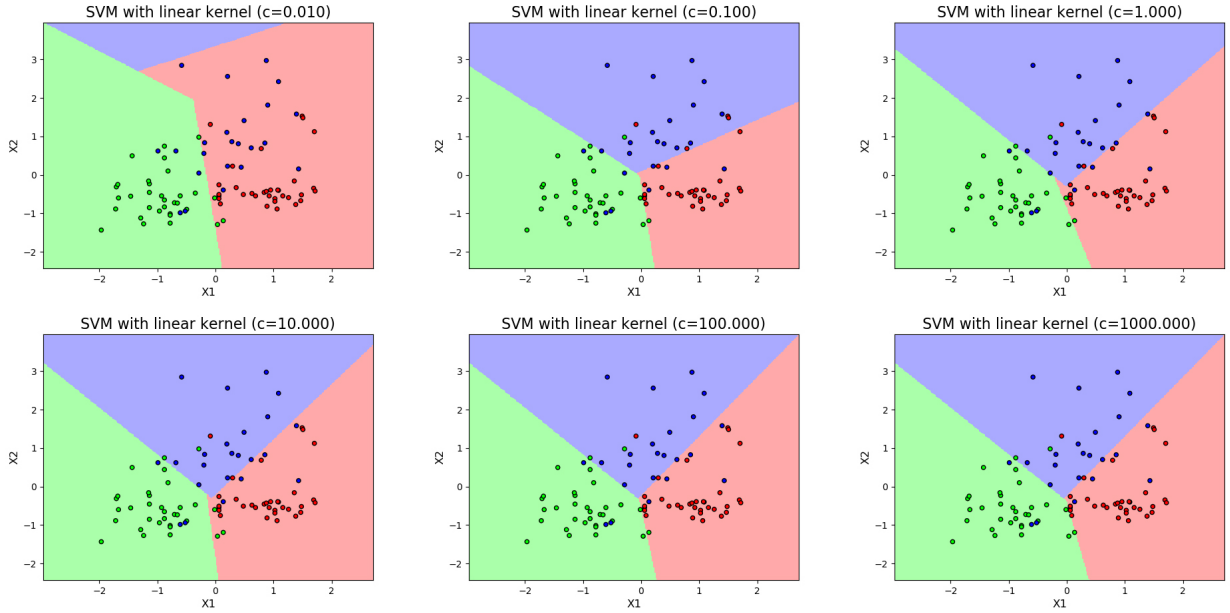

Figure 4.5: *Linear kernel SVM decision boundaries.*

After that the accuracy of each model is evaluated on the validation set and this is the output: [*Point 8c*]

```
1 The best value for C is 0.1 with an accuracy on the validation set equal to 75.00%
```

4.2.2 Evaluation on validation set

In 4.6 the accuracy on the validation set is shown for every C. Since the function reaches a plateau, the best C is chosen as the first best value.[*Point 9*]

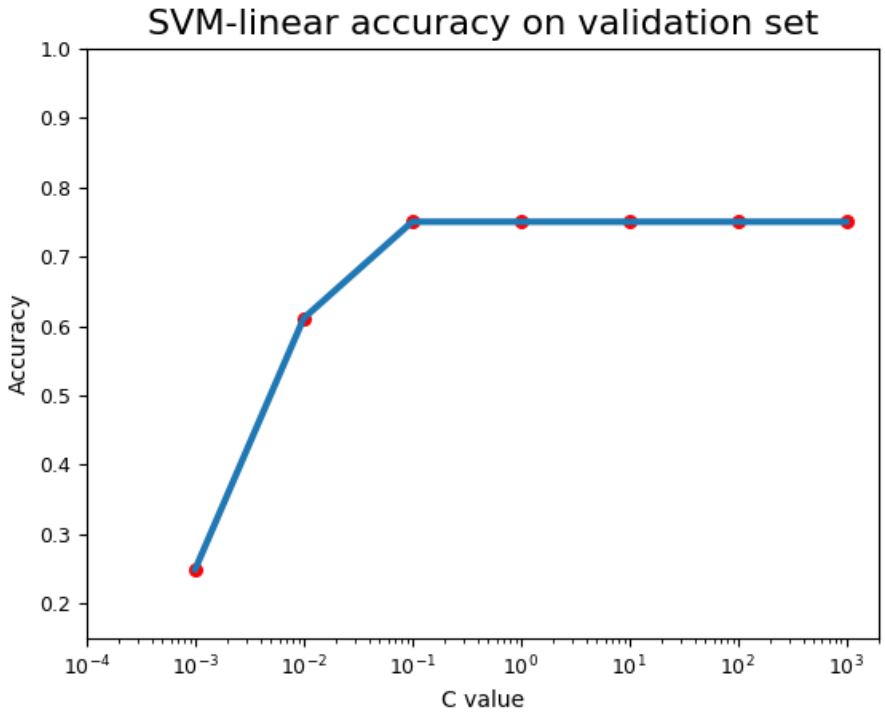

Figure 4.6: *Linear kernel SVM accuracy on validation set*

4.2.3 Testing the classifier

Once the best parameter C is found, the classifier is trained with the best C on the train plus validation set. Here is the output:[*Point 11*]

```
1 The accuracy for bestC=0.1 on the test set is 79.63%
```

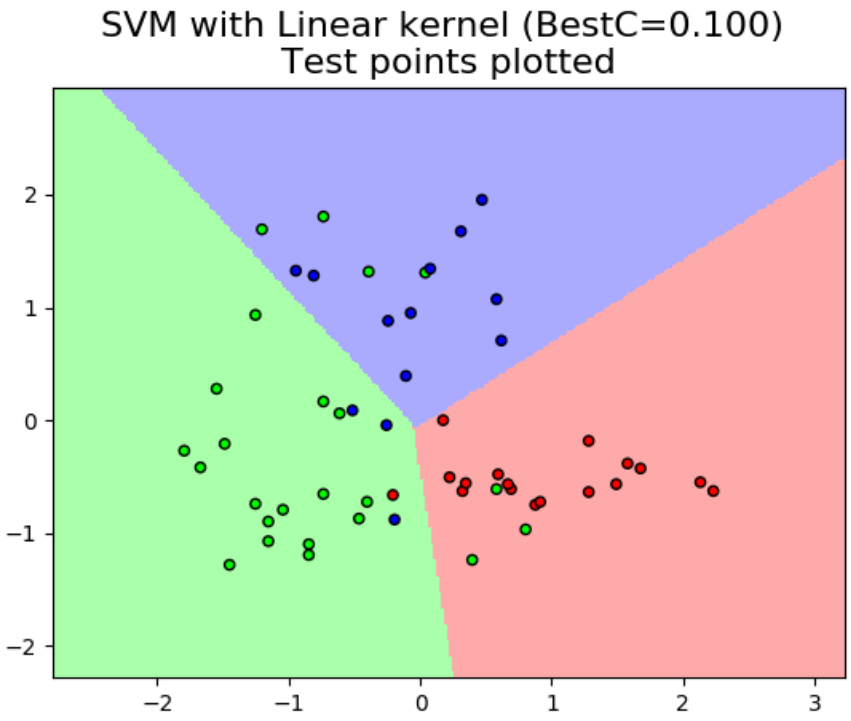

Figure 4.7: *Linear kernel SVM Decision boundaries for best C plot with test points*

The goals for SVM are mainly two: find a hyperplane with the largest minimum margin, and find a hyperplane that correctly separates as many instances as possible. The problem is that not always it will be able to get both things. As C increases the tendency to satisfy the latter goal will increase. In other words, C represents the cost of misclassification. In this particular case after C increases a certain threshold there is a sort of balance between this two goals. For low values of C the misclassification cost is so low that the model prefers to misclassify green and blue points. In fact a lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. [*Point 10*]

4.3 Radial Basis Function Kernel SVM

The RBF-SVM model uses the kernel described in 4.5 take as input two parameter: one is the same C seen before, the other is γ. γ defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

4.3.1 Application

First the SVM with rbf kernel is trained on the train set using γ set as *auto* and C in the same range as section 4.2.1.[*Point 12*]

```
from sklearn import svm
from sklearn.svm import SVC
for c in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:
    clf = svm.SVC(kernel='rbf', gamma='auto', C=c)
    clf.fit(X_train, y_train)
```

Listing 4.2: *SVC classfier*

Here are the plots.

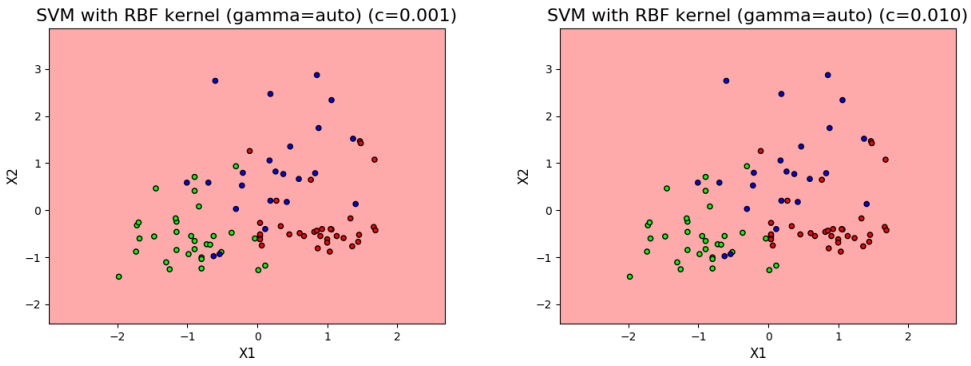

Figure 4.8: *RBF kernel SVM worst cases*

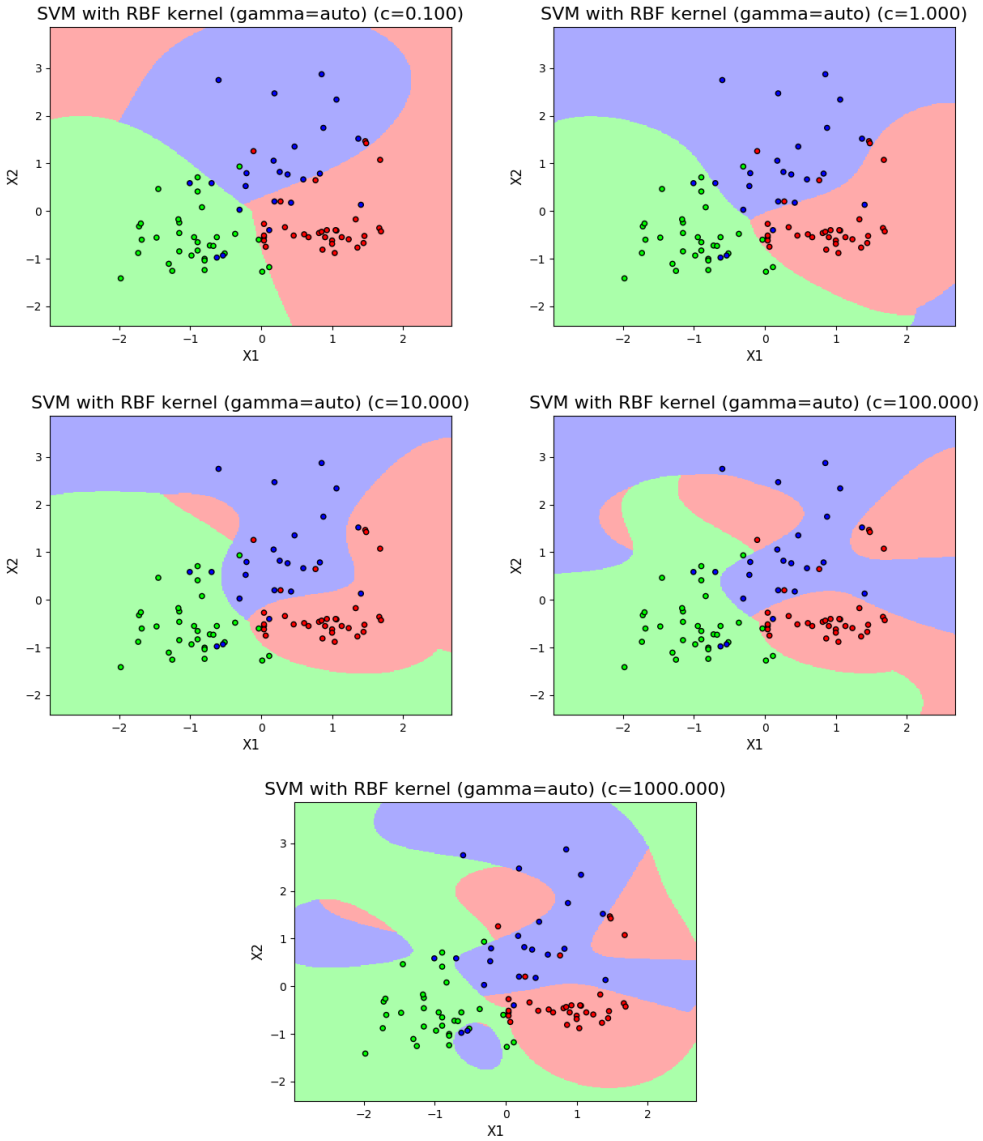

Figure 4.9: *RBF kernel SVM best cases*

4.3.2 Evaluation on validation set

The computation of the accuracy on validation set is the following.

```
1 The best value for C is 10.0 with an accuracy on the validation set equal to 80.56%
```

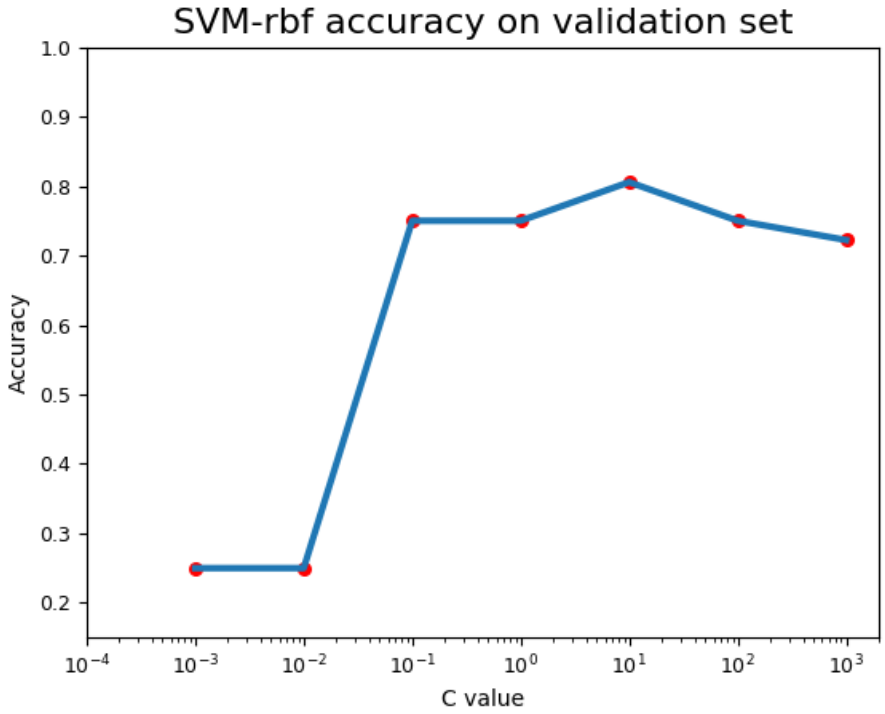

Figure 4.10: *RBF kernel SVM accuracy on validation set*

4.3.3 Testing the classifier

Here the best classifier, i.e. the one with the best accuracy on the validation set, is retrained on train plus validation set and that is the result.[*Point 13*]

```
1 The accuracy for C=10.0 on the test set is 79.63%
```

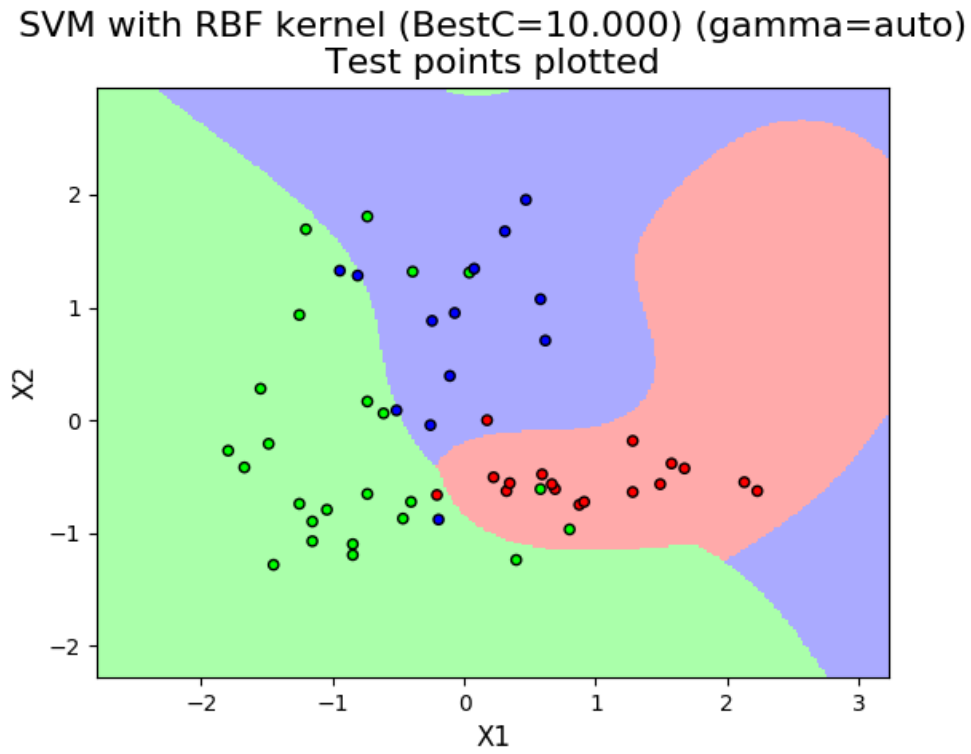

Figure 4.11: *RBF kernel SVM Decision boundaries for best C and gamma auto, plot with test points*

In comparison with the linear kernel, the RBF kernel model shows non linear boundaries. For this reason the accuracy should be slightly better. In this particular case it is clear that this model needs tuning also gamma parameter to perform better.

4.3.4 Grid Search of the best parameters

In this section a Grid search is perform in order to find the best couple of parameters C and γ.[*Point 15*]

```
C_range = np.logspace(-2, 6, 9)
gamma_range = np.logspace(-4, 4, 9)
ACCs=np.zeros((len(C_range),len(gamma_range)))
for i,C in enumerate(C_range):
    for j,gamma in enumerate(gamma_range):
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_train, y_train)
```

```
8           pred_test = clf.predict(X_val)
9           ACCs[i][j]=metrics.accuracy_score(y_val, pred_test)
```

Listing 4.3: *RBF SVM GridSearch*

This is a graphical representation of the result.

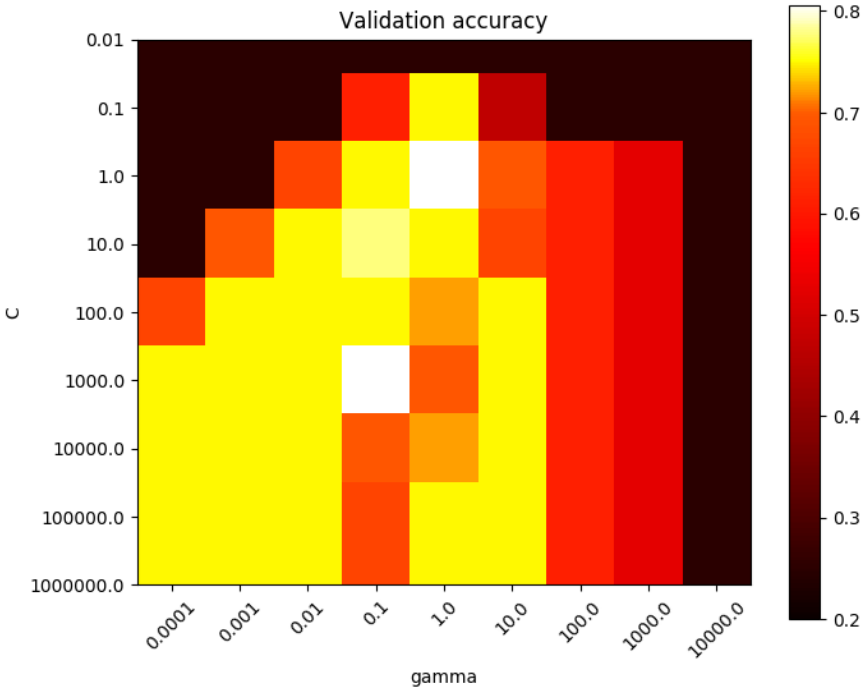

Figure 4.12: *RBF kernel SVM grid search heat map*

Here is the output of the program.

```
1 GridSearch found best accuracy for C=1.0 and gamma=1.0
2 The parameters above produce on the test set an accuracy equal to 85.19%
```

These are the decision boundaries.

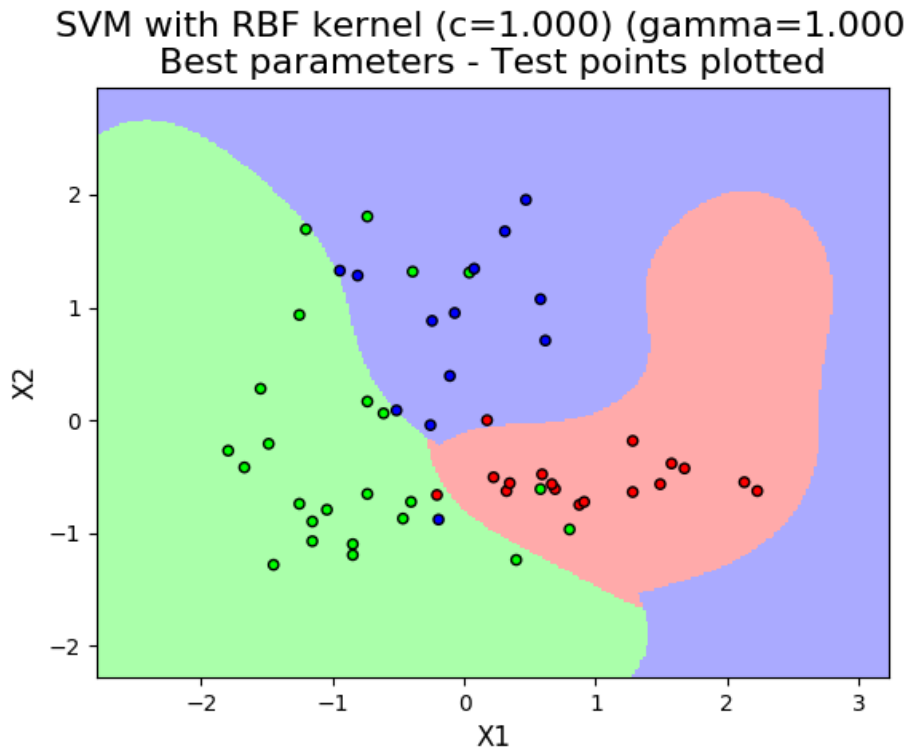

Figure 4.13: *RBF kernel SVM Decision boundaries for best C and γ*

4.4 Cross-validation

4.4.1 Overview

The Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called *K-fold cross-validation*. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation. Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. It is a popular method because it is simple to

understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

4.4.2 Application

In this section a grid search with K-fold split with $k = 5$ is performed.[*Point 16,17*]

```
X_trainVal, y_trainVal = np.concatenate((X_train, X_val)), np.concatenate((
    y_train, y_val))
k = 5
k_Fold = KFold(n_splits=k)
ACCs = (np.size(C_range), np.size(gamma_range))
ACCs = np.zeros(ACCs)
for indTrain, indVal in k_Fold.split(X_trainVal, y_trainVal):
    for i, C in enumerate(C_range):
        for j, gamma in enumerate(gamma_range):
            clf = SVC(C=C, gamma=gamma)
            clf.fit(X_trainVal[indTrain], y_trainVal[indTrain])
            pred_test = clf.predict(X_trainVal[indVal])
            acc = metrics.accuracy_score(y_trainVal[indVal], pred_test)
            ACCs[i, j] += acc

# obtain mean value of accuracy matrix
ACCs /= k
```

Listing 4.4: *Grid search with K-fold split*

At this point the heat map is plotted to show the distribution of accuracy over C and γ.

```
The best accuracy performed on the validation set with K Fold = 5 is 79.87%
Found with C=100.000 and gamma=0.010
```

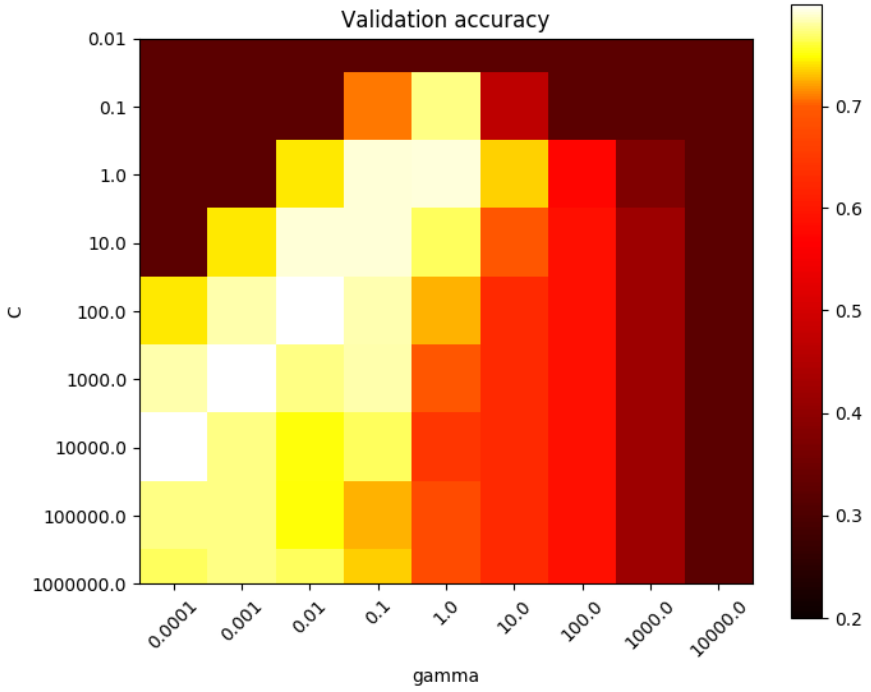

Figure 4.14: *RBF kernel SVM with K-fold grid search heat map*

4.4.3 Testing the classifier

This is the output of the program and, consequently, the decision boundaries.[*Point 18*]

```
1 The parameters above produce on the test set an accuracy equal to 79.87%
```

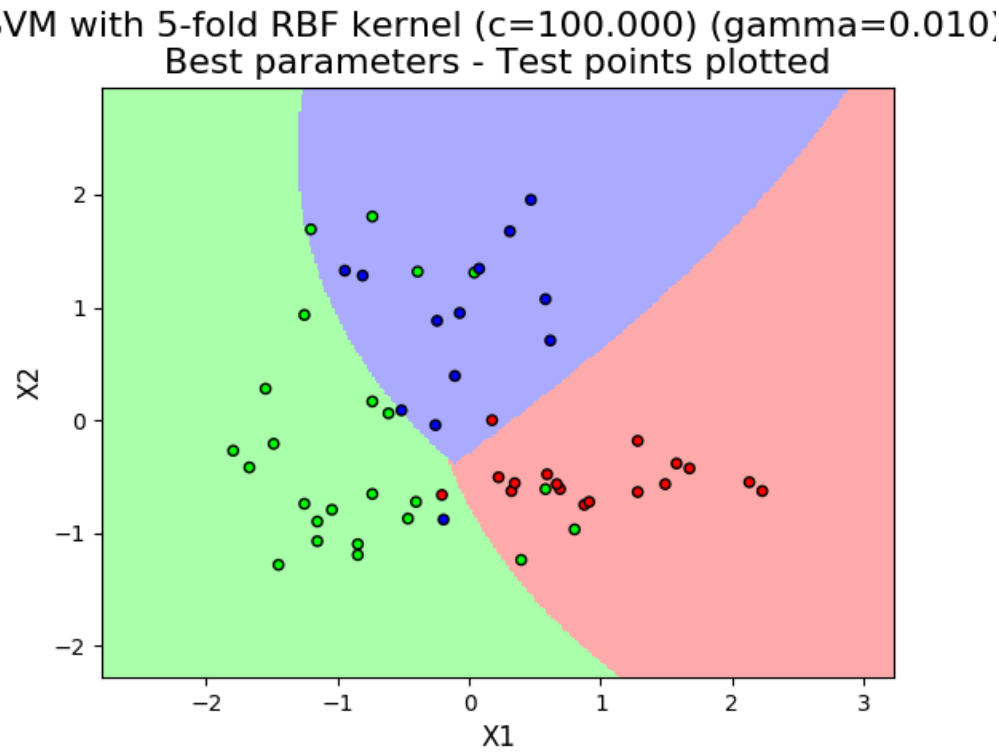

Figure 4.15: *RBF kernel SVM with K-fold best Decision boundaries with test set plotted*

As it said before K-fold Cross Validation is a less optimistic method to estimate correctly and more generally the accuracy of a classifier. The proof of this is in the fact that the accuracy on the validation set is the same as the test set.