Artificial Intelligence and Machine Learning

Homework 2: Support Vector Machines

Jeanpierre Francois S243920

January 16, 2019

1 Description

In this homework I had the opportunity to touch by hand support vector machine algorithm with multiple kinds of kernels using python libraries.
The homework was split in different steps:

1. Linear SVM
2. RBF Kernel
3. K-Fold cross validation

There were also some specific questions 5 about certain crucial points.

1.1 Code index

2 Linear SVM

In this homework we had to work with the Iris dataset, a famous multivariate dataset often used in machine learning, selecting only the first two dimension without applying any PCA dimensionality reduction. Once splitting the data in training/validatio/test set with 5:2:3 proportion, I began to finetune the C hyper parameter (in a range from 10ˆ-3 to 10ˆ3) training the algorithm on the training set and evaluating the score onto the validation one. After plotting how the accuracy on the validation varies when changing the C, I took the best C value found in order to evaluate the model on the test set where I got a score not so lower then the one obtained on the validation set which is quite a result.

3 RBF Kernel

The finetunement of C for the radial basis function (rbf) kernel, with the gamma hyper parameter set to auto, resulted in a score similar to the linear one. Its boundaries are of course different as they appear as curves, that's because rbf implement a non linear kernel. In this rbf case I have C and Gamma to finetune so to choose the best hyper parameters the grid search comes in handy. Finetuning C and Gamma bring onto an overall good score onto the test set.

4 K-Fold Cross Validation

The K-Fold cross validation is a resampling procedure used to score models on a limited dataset. It's often used when the training set is small. To deploy this kind of validation I merged the previous training and validation set into a single set that represents 70% of the data. This set is then used for a grid search implementing the K-Fold cross validation method, with k equal to 5, which obtains a better result than the previous simple Training/Validation/Test paradigm data split.

5 Questions

***6.* Q** : How do the boundaries change? Why?

A: The decision boundaries varies a lot when varying the C penalty parameter of the error term. This hyper parameter influence the SVM algorithm margin size, the bigger C we decide to take the smaller margin will be chosen for the hyperplane and viceversa. The misclassification rate of the boundaries will change according to the given C. There is no absolute best C value for every dataset in the world but it should be finetuned for each different dataset.

***7.* Q** : Use the best value of C and evaluate the model on the test set. How well does it go??

A: Using the best value of the C hyper parameter found onto the test set I got a result that it's quite close to the best score obtained onto the validation set, which is nice.

***10.* Q** : Are there any differences compared to the linear kernel? How are the boundaries different?

A: There are many differences between linear kernel and rbf boundaries plots and that's because the latter uses non-linear combinations of dataset's features to uplift the samples into a higher-dimensional feature space where linear decision boundary can be used to separate different classes. The main difference is that the linear kernel separates with lines while the rbf kernel instead divides data with curves thanks to its capability to follow with higher precision data distribution. Simply looking at the scores it seams that both linear and rbf kernel SVM give equally good results but there are differences: the linear svm is a parametric model and scales better on big datasets, the complexity of the rbf instead grows with the size of the training set. Not only the latter is more expensive to train but there are more hyper parameters to tune and furthermore it's much easier to overfit a complex model than a simpler one. That said, if the dataset is not linearly separable a complex model like rbf, which exploits a higher-dimensional feature space to separate samples, is needed.

***16.* Q** : Evaluate the parameters on the test set. Is the final score different? Why?

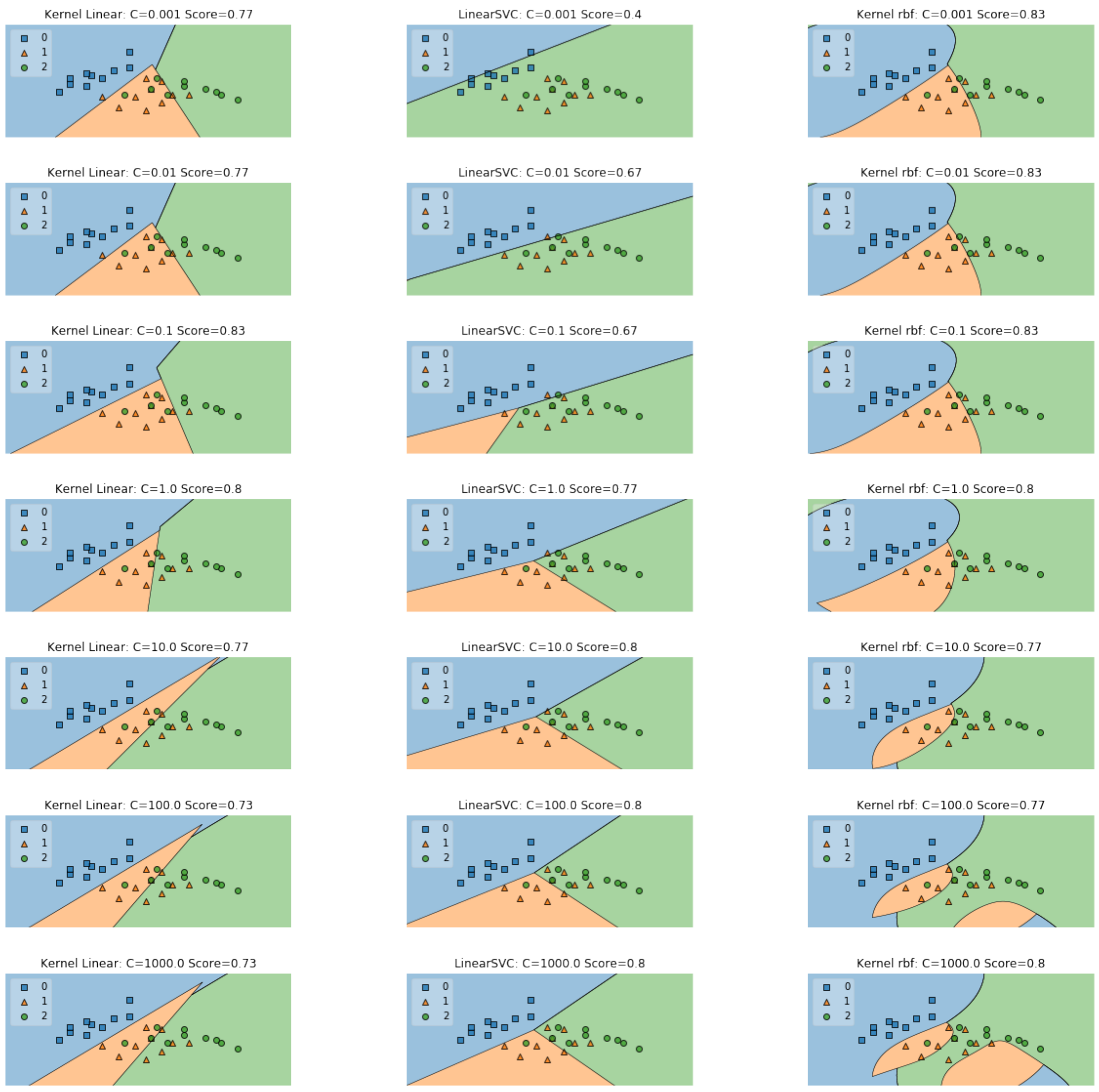
A: The final score using K-Fold cross validation during grid search, in order to finetune the rbf hyper parameters, results higher than the simple grid search, that's because in this case I didn't have a big dataset which caused a small training set. In this case is better to use cross validation in order to extract the most from our dataset.

Code setup & constants definitions

```
In [3]: import operator
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import svm, datasets
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        import sklearn.preprocessing as prepro
        from matplotlib import gridspec
        from mlxtend.plotting import plot_decision_regions as plt_reg
        import warnings
        warnings.filterwarnings("ignore")

        iris = datasets.load_iris()
        X = iris.data[:, :2]
        y = iris.target

        x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
         ↪  random_state=1,stratify=y)
        x_val, x_test, y_val, y_test = train_test_split(x_test, y_test,
         ↪  test_size=0.6, random_state=1, stratify= y_test)

        C_vals = [] #10^3 up to 10^3, step = 10^1
        g_vals = [] #10^-11 up to 10^3, step= 10^1
        C_i = 0.001 #10^-3
        g_i = 0.00000000001 #10^-11

        for i in range(7):
            C_vals.append(C_i)
            C_i*=10
        for i in range(15):
            g_vals.append(g_i)
            g_i*=10
            g_i = round(g_i,11-i)

        scores = np.empty((len(C_vals),3))
```

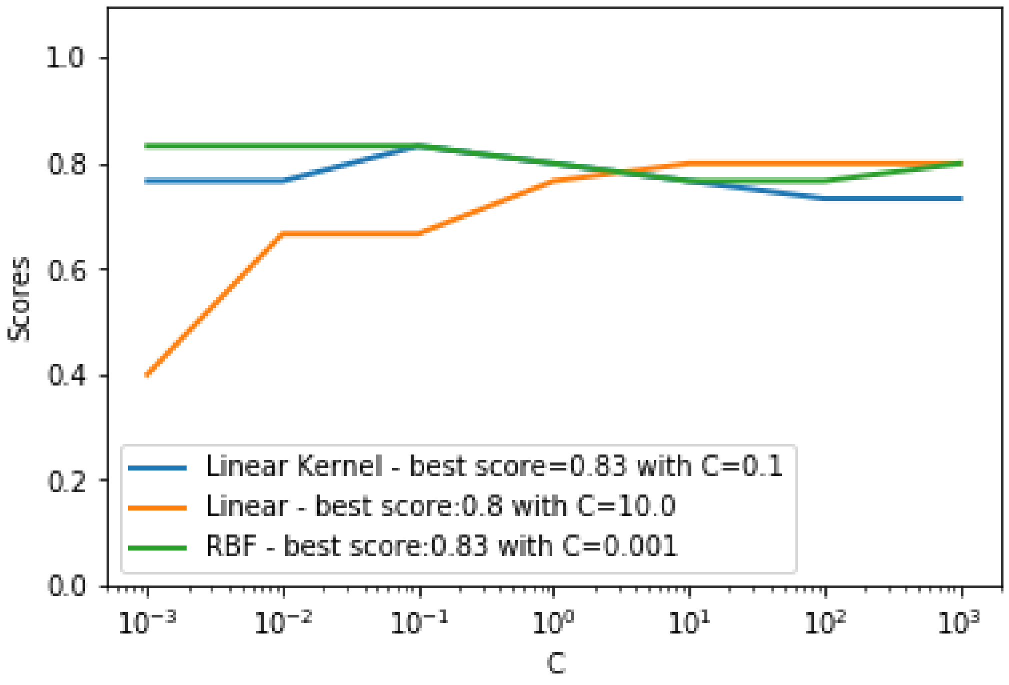
Finetuning C

```
In [4]: fig = plt.figure(figsize=(20,20))
        fig.subplots_adjust(hspace=0.4, wspace=0.4)
        i = 0
        for j,C in enumerate(C_vals):
            fig.add_subplot(7, 3, i+1)
            model_k = svm.SVC(kernel='linear',C=C)
            model_l = svm.LinearSVC(C=C, max_iter=1000000)
            model_rbf = svm.SVC(kernel='rbf',C=C,gamma='auto')
            model_k.fit(x_train,y_train)
            model_l.fit(x_train,y_train)
            model_rbf.fit(x_train,y_train)
            scores[j,0] = model_k.score(x_val,y_val)
            scores[j,1] = model_l.score(x_val,y_val)
            scores[j,2] = model_rbf.score(x_val,y_val)
            plt_reg(x_val,y_val,clf=model_k, legend=2)
            plt.title('Kernel Linear: C='+str(C)+'
             ↪  Score='+str(round(scores[j,0],2)))
            plt.xticks([])
            plt.yticks([])
            fig.add_subplot(7, 3, i+2)
            plt_reg(x_val,y_val,clf=model_l, legend=2)
            plt.title('LinearSVC: C='+str(C)+' Score='+str(round(scores[j,1],2)))
            plt.xticks([])
            plt.yticks([])
            fig.add_subplot(7, 3, i+3)
            plt_reg(x_val,y_val,clf=model_rbf, legend=2)
            plt.title('Kernel rbf: C='+str(C)+' Score='+str(round(scores[j,2],2)))
            plt.xticks([])
            plt.yticks([])
            i += 3
        plt.show()
```



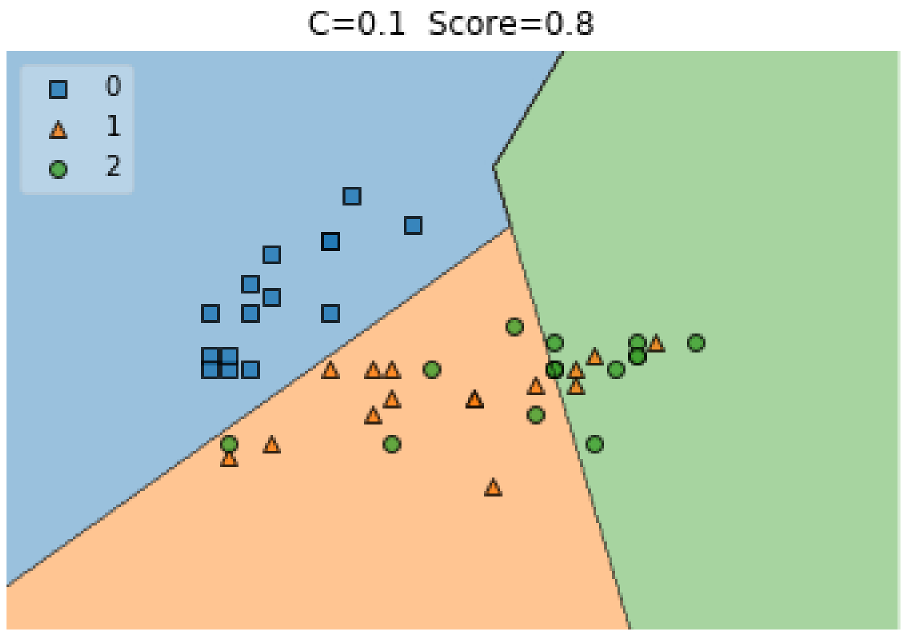
```
In [5]: param_range = np.logspace(-3, 3, len(C_vals))
        plt.ylim(0.0, 1.1)
        lw = 2
        bestC = np.empty((len(scores), 3))
        for j in range(len(scores[0])):
                c, value = max(enumerate(scores[:,j]),key=operator.itemgetter(1))
                bestC[j,0] = C_vals[c]
                bestC[j,1] = value
                bestC[j,2] = c

        plt.semilogx(param_range, scores[:,0], label="Linear Kernel - best
         ↪ score="+str(round(bestC[0,1],2))+" with C="+str(bestC[0,0]), lw=lw)
        plt.semilogx(param_range, scores[:,1], label="Linear - best
         ↪ score:"+str(round(bestC[1,1],2))+" with C="+str(bestC[1,0]), lw=lw)
```

```
plt.semilogx(param_range, scores[:,2], label="RBF - best
↪  score:"+str(round(bestC[2,1],2))+" with C="+str(bestC[2,0]), lw=lw)
plt.xlabel('C')
plt.ylabel('Scores')
plt.legend(loc='best')
plt.show()
```



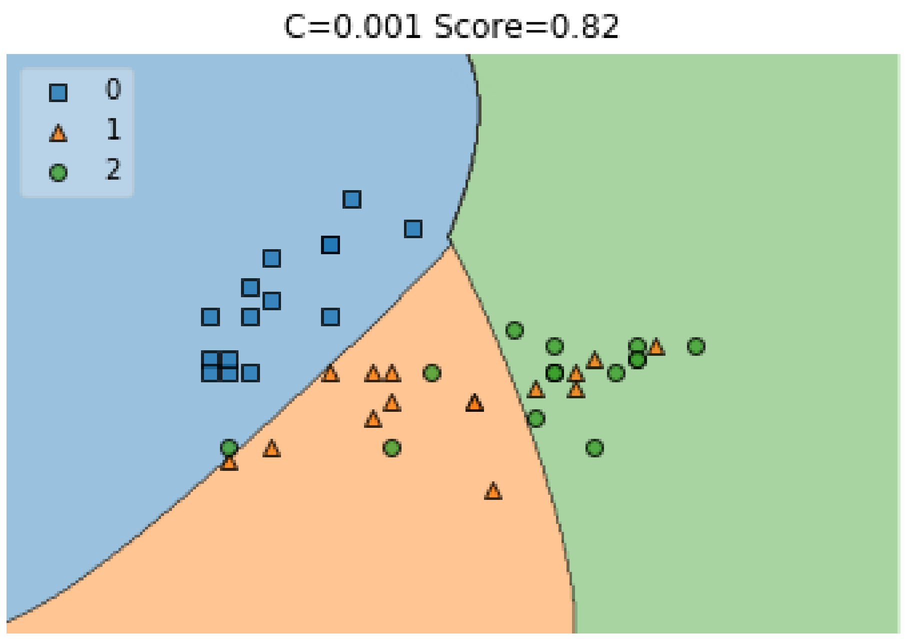
Best C found for linear kernel

```
In [6]: C = bestC[0,0]
        model_k = svm.SVC(kernel='linear',C=C)
        model_k.fit(x_train,y_train)
        test_score = model_k.score(x_test,y_test)
        plt_reg(x_test,y_test,clf=model_k, legend=2)
        plt.title('C='+str(C)+'  Score='+str(round(test_score,2)))
        plt.xticks([])
        plt.yticks([])
        plt.show()
```



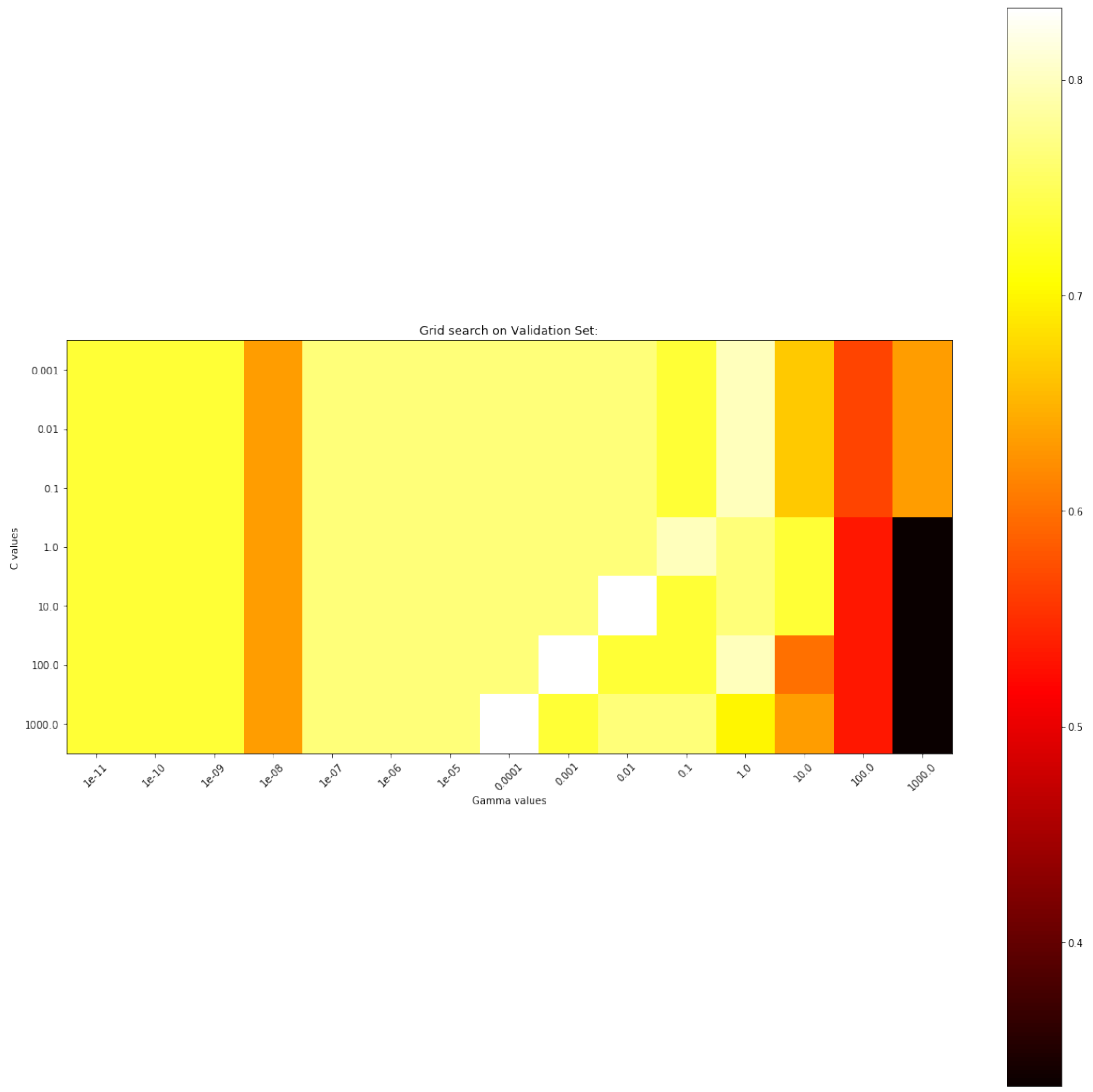
Best C found for rbf kernel

```
In [7]: C = bestC[2,0] #Best C value rbf
        model_rbf = svm.SVC(kernel='rbf',C=C,gamma='auto')
        model_rbf.fit(x_train,y_train)
        test_score = model_rbf.score(x_test,y_test)
        plt_reg(x_test,y_test,clf=model_rbf, legend=2)
        plt.title('C='+str(C)+' Score='+str(round(test_score,2)))
        plt.xticks([])
        plt.yticks([])
        plt.show()
```



Grid search for rbf kernel with validation set Grid search on Validation Set

```
In [8]: best_vals = np.empty(3)
        scores = np.empty((len(C_vals),len(g_vals)))
        plt.figure(figsize=(20,20))
        plt.subplots_adjust(hspace=0.4, wspace=0.4)
        for i,C in enumerate(C_vals):
            for j,g in enumerate(g_vals):
                model_rbf = svm.SVC(kernel='rbf',C=C,gamma=g)
                model_rbf.fit(x_train,y_train)
                score = model_rbf.score(x_val,y_val)
                scores[i][j] = score
                #print('Score='+str(round(score,2))+' C='+str(C)+' Gamma='+str(g))
                if score > best_vals[0]:
                    best_vals[0] = score
                    best_vals[1] = C
                    best_vals[2] = g

         ↪  plt.imshow(scores.reshape(len(C_vals),len(g_vals)),interpolation='nearest',cmap=plt.
        plt.xlabel('Gamma values')
        plt.ylabel('C values')
        plt.colorbar()
        plt.xticks(np.arange(len(g_vals)),g_vals,rotation = 45)
        plt.yticks(np.arange(len(C_vals)),C_vals)
        plt.title('Grid search on Validation Set:')
        plt.show()
        print('Best score='+str(round(best_vals[0],2))+' with
         ↪  C='+str(best_vals[1])+' Gamma='+str(best_vals[2]))
```

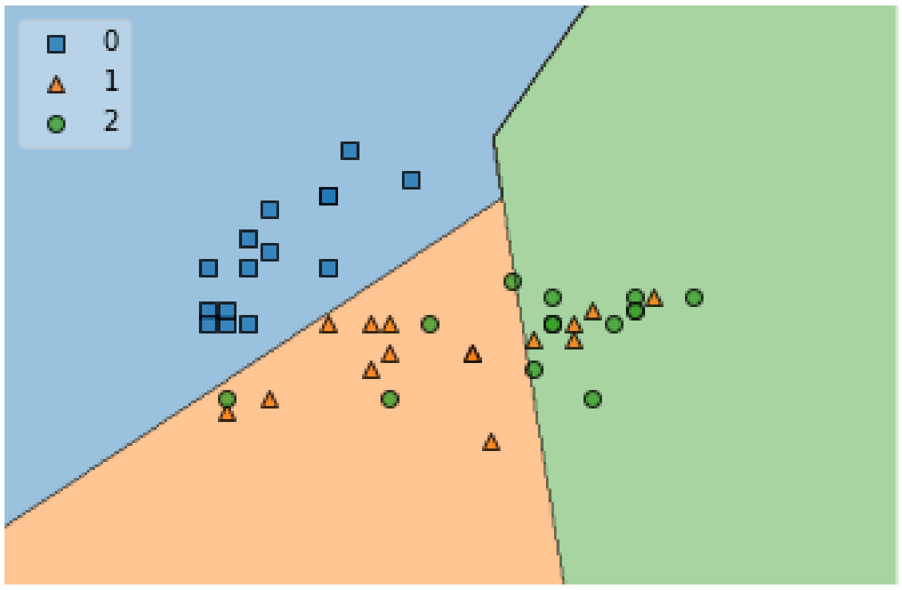


```
Best score=0.83 with C=10.0 Gamma=0.01
```

Grid search results for rbf kernel

```
In [9]: model_rbf = svm.SVC(kernel='rbf',C=best_vals[1],gamma=best_vals[2])
        model_rbf.fit(x_train,y_train)
        test_score = model_rbf.score(x_test,y_test)
        plt_reg(x_test,y_test,clf=model_rbf, legend=2)
        plt.title('C='+str(best_vals[1])+' Gamma='+str(best_vals[2])+'
         ↪  Score='+str(round(test_score,2)))
        plt.xticks([])
        plt.yticks([])
        plt.show()
```

C=10.0 Gamma=0.01 Score=0.8

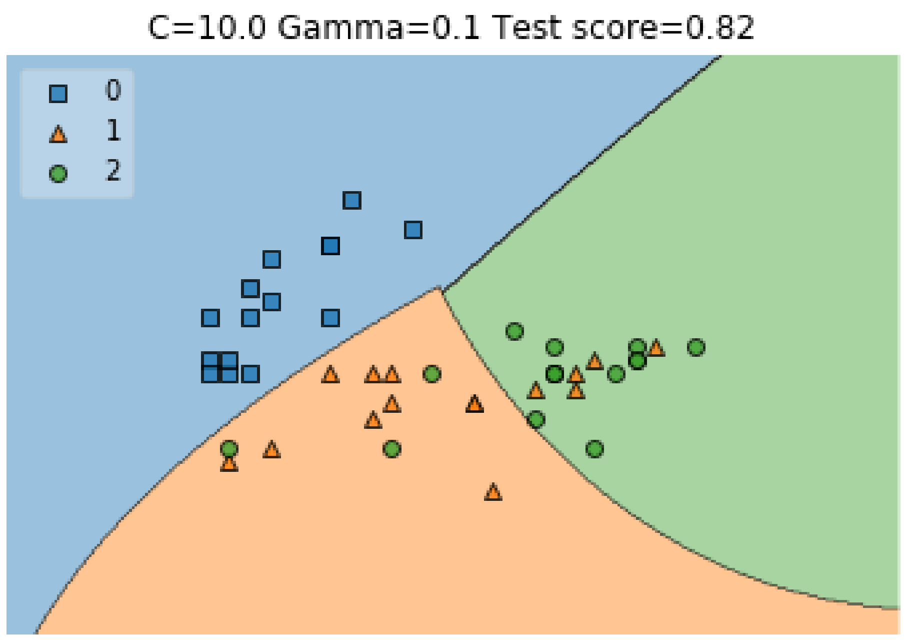


K-Fold cross validation with grid search for rbf kernel

```
In [10]: k = 5
         x_folds = np.concatenate((x_train,x_val),axis=0)
         y_folds = np.concatenate((y_train,y_val))
         best_vals = np.empty(3)
         xk_folds = np.array_split(x_folds,k)
         yk_folds = np.array_split(y_folds,k)
         for i,C in enumerate(C_vals):
             for j,g in enumerate(g_vals):
                 avg_score = 0
                 for z in range(k):
                     xk_train = list(xk_folds)
                     xk_val = xk_train.pop(z)
                     xk_train = np.concatenate(xk_train)
                     yk_train = list(yk_folds)
                     yk_val = yk_train.pop(z)
                     yk_train = np.concatenate(yk_train)
                     model_rbf = svm.SVC(kernel='rbf',C=C,gamma=g)
                     model_rbf.fit(xk_train,yk_train)
                     avg_score += model_rbf.score(xk_val,yk_val)
                 avg_score /= k
                 if avg_score > best_vals[0]:
                     best_vals[0] = avg_score
                     best_vals[1] = C
                     best_vals[2] = g

         print('K-fold grid search with k = '+str(k))
```

```
    print('Best average score='+str(round(best_vals[0],2))+' with
    ↪  C='+str(best_vals[1])+' Gamma='+str(best_vals[2]))
    model_rbf = svm.SVC(kernel='rbf',C=best_vals[1],gamma=best_vals[2])
    model_rbf.fit(x_folds,y_folds)
    test_score = model_rbf.score(x_test,y_test)
    plt_reg(x_test,y_test,clf=model_rbf, legend=2)
    plt.title('C='+str(best_vals[1])+' Gamma='+str(best_vals[2])+' Test
    ↪  score='+str(round(test_score,2)))
    plt.xticks([])
    plt.yticks([])
    plt.show()
```

```
K-fold grid search with k = 5
Best average score=0.81 with C=10.0 Gamma=0.1
```


C=10.0 Gamma=0.1 Test score=0.82