Homework1-s263138

November 15, 2019

1 Import dataset,split and normalize

In this section i imported some needed stuff from sklearn,numpy and matplot which I will need later. before imported the dataset with load_wine() function from sklearn and then i took the first two attributes. i have splitted two time the dataset with the function 'train_test_split' from sklearn:

- first split : 50% train-split 50% temporary
- second split from **temporary**: 40% validation-split and 60% test-split

this means respect on full-dataset:

- 50% train split
- 20% validation split
- 30% test split

please note the attributes **stratify** and **random_state=x**,

- **stratify**: balance the number of different clesses for each split ex (#3 label A,#3 label B, #3 label C)
- **random_state=X:** note that train_tes_split split random by default, X is the seed for generate random number, in this case on every execution the splits generated are the same, without random_state=X on every execution the splits are generated random.

normalization: i have normalized the split with the function **StandardScaler** from sklearn,first i fit the scaler with the train+validation then i scaled train,test and validation splits . i've also created a new split called "x_train_val" needed for train the algorithms after tuning the usefull values on validation set and then valuate it on test set.

2 KNN

the basic function of the algorithm is to evaluate the class based on the medium of the classes of nearest neighbors, ex:

if k-nearest neighbors on average are 'red' the considered point is probably red.

is very important the K value : the number of nearest neighbors tath the algorithm evaluate. in theory an higher k reduce the noise but the classifier is less robust

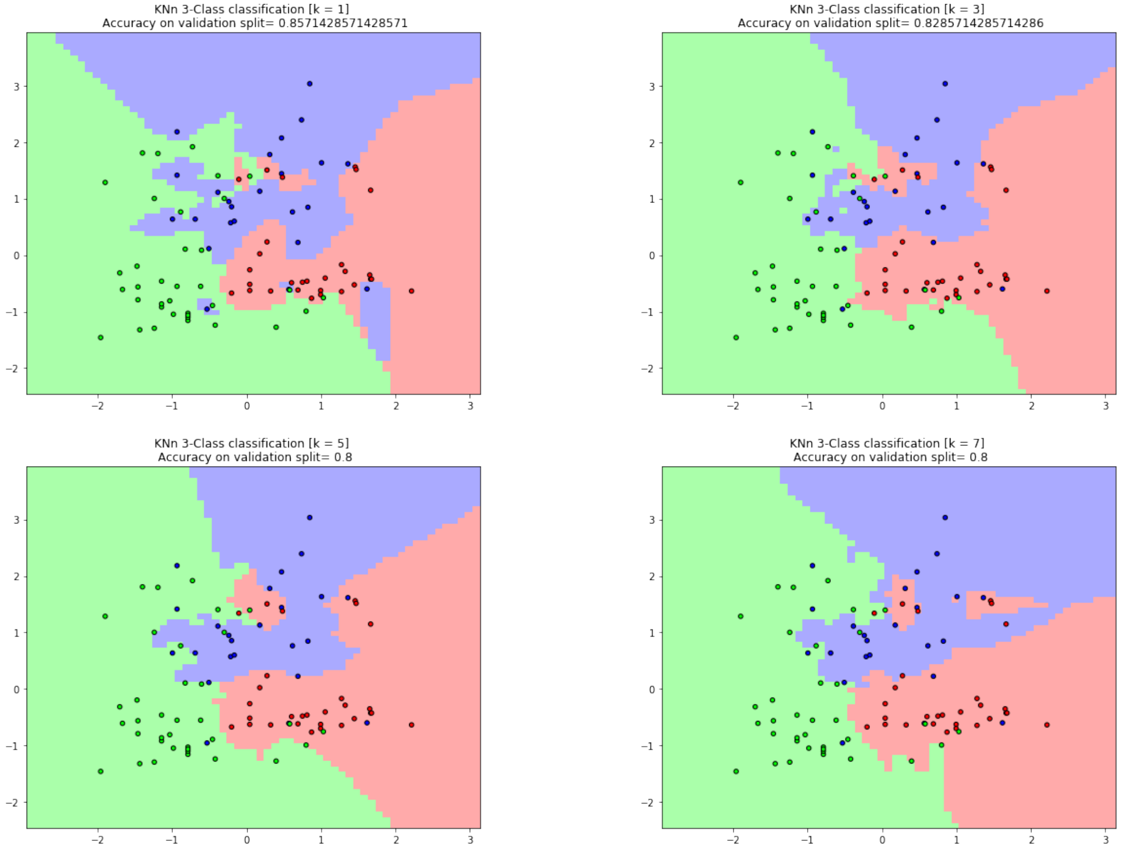
2.0.1 PLOT DATA AND DECISION BOUNDARY

in this section i iterate over k values;

for each k values create a knn-classifier which is fitted with train data,every classifier is evaluated on validation set for find the K-value which give better score.

for each classifier are printed the boundaries and the train point,

we see the trend of the graphs as the parameter k changes.



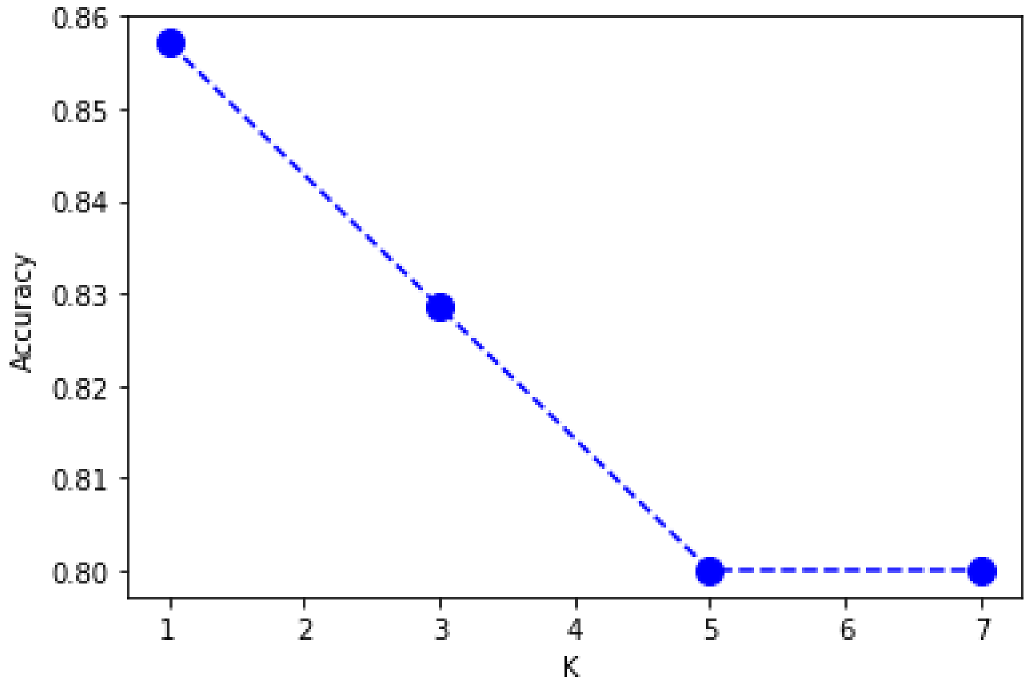
```
[BEST K = 1] accuracy: 0.8571428571428571
```

2.1 How the boundaries change?

it seems to me that as K increases, the boundaries of the densest point shrink, while the boundaries of the most scattered point widen

2.1.1 ACCURACY (ON VALIDATION SPLIT) CHANGING K

this graph show how varies the accuracy with different k, please note that this graph is extremly dipendent on which split are generated at first point, ex: with another split we can see a completely different graph.



2.1.2 EVALUATE THE MODEL ON TEST SET

now i evaluated the model (with the best k found on validation set) on test set, we can see the accuracy lower than the accuracy on validation set.

```
Accuracy on TEST SET (k=1): 0.7407407407407407
              precision    recall  f1-score   support

           0       0.75      0.83      0.79        18
           1       0.84      0.76      0.80        21
           2       0.60      0.60      0.60        15

    accuracy                           0.74        54
   macro avg       0.73      0.73      0.73        54
weighted avg       0.74      0.74      0.74        54
```

3 SVM

the purpose of SVM is to find a linear hyperplane to separate the data.

we can find different hyperplane, but which one is better?

the goal is to find the hyperplane to maximizes the margins from different classes.

but if the boundaries are not linearly separable?

Transform data into higher dimensional space!

we can use a **kernel** to do this.
there are two important params for **tuning** the algorithm:

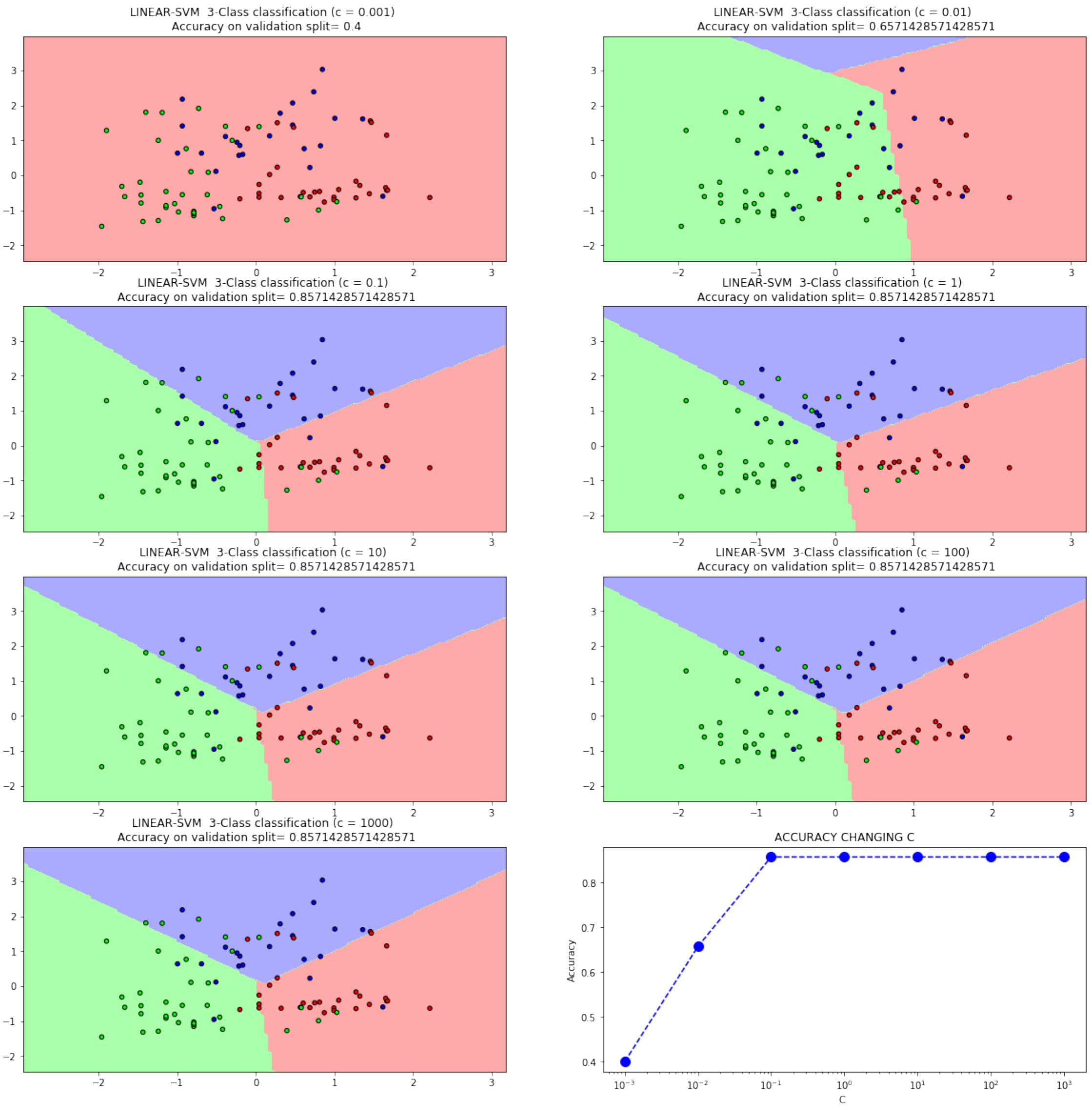
- **C** value: is like a threshold that tell to SVM how much you want to avoid misclassifying each training. ex:
    - lower C: generate a higher-margin hyperplane that smoothly the classifier -> increase the probability of the outliers
    - higher C: generate a lower-margin hyperplane that minimize the missclassification

- **Gamma** value: determine the influence distance of a single sample:
    - lower gamma: also distant points are considered from classifier
    - higher gamma: only nearbly points are considered from classifier

3.1 Linear SVM

i trained a linear-SVM for each given C ([0.001, 0.01, 0.1, 1, 10, 100,1000]) with test split, for each classifier there is printed the boundaries, the train points and the score on validation set. each classifier is evaluated on validation set to find the best C value for this classifier, then when the best c is found i trained a new classifier with train set+validation and c set and evaluate the accuracy on it

nb: in my implementation if found the same accuracy with different C i prefeer to chose the lowest C value.



```
[ VALIDATION SET] BEST C = 0.1 => accuracy = 0.8571428571428571

[ EVALUATE ON TEST SET with BEST c ] => Accuracy =  0.8703703703703703
```

3.1.1 Evaluate on test set

now i trained the model with train+val split and evaluated it on test set the score is better than the score on validation set, the tuning of the c parameter seems to be working.

3.1.2 How the boundaries change?

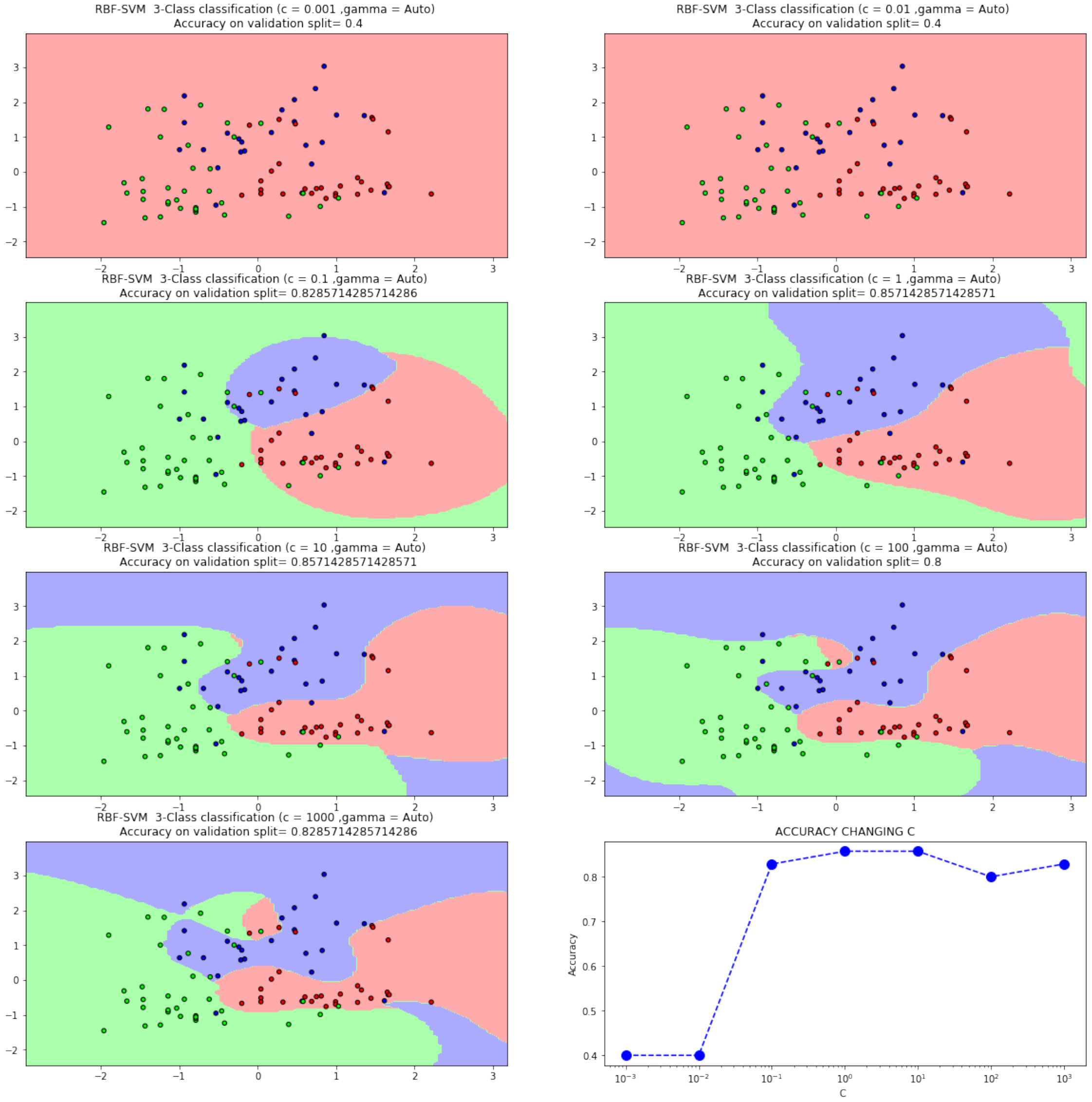
we can see on the lowest value of c (0.001) a single boundary 'red', this mean that the c is too small (and the margin to high) to allow the classifier to differentiate the various classes.

while increasing c value there we see the boundaries becoming more defined but from c=1 to c= 1000 i dont see substancial differences.

3.2 RBF kernel: Radial Base Function

in this section i want to use the kernel RBF on classifier.

like the linear svm i trained a rbf-SVM for each given C ([0.001, 0.01, 0.1, 1, 10, 100,1000]) with test split to find the best C values. and then i create a new classifier (fitted with train+val) and evaluate the accuracy on it



```
[ VALIDATION SET]  BEST C = 1 , accuracy = 0.8571428571428571

[ TEST SET with BEST c] => Accuracy =  0.9444444444444444
```

3.2.1 Differences between linear and rbf

obviously the boundaries are no longer linear but are curved, compared to the linear kernel it can be noted that even with c = 0.01 the classifier cannot distinguish the three classes.

we can also see that as C grows, the contours become more 'complex' and detailed.

the difference between the various boundaries is more marked than the linear-svm

3.2.2 Evaluate on test set

the score is much better than the score on validation set, the tuning of the c parameter seems to be working very good on test set.

3.3 GRID SEARCH

grid search is a training procedure aimed at choosing best c and gamma parameters.

for each c and gamma in

```
C = [0.001, 0.01, 0.1, 1, 10, 100,1000]
gamma_values=[0.001, 0.01, 0.1, 1, 10, 100,1000]
```

i fitted an rbf-svm with train split and evaluate on it the accuracy on validation split. at the end of this double-iteraction i take the touple (c and gamma) that maximize the accuracy (on validation split)

```
[TUNING GAMMA AND C ON VALIDATION SET]
c=0.001 gamma=0.001 accuracy=0.4
c=0.001 gamma=0.01 accuracy=0.4
c=0.001 gamma=0.1 accuracy=0.4
c=0.001 gamma=1 accuracy=0.4
c=0.001 gamma=10 accuracy=0.4
c=0.001 gamma=100 accuracy=0.4
c=0.001 gamma=1000 accuracy=0.4
c=0.01 gamma=0.001 accuracy=0.4
c=0.01 gamma=0.01 accuracy=0.4
c=0.01 gamma=0.1 accuracy=0.4
c=0.01 gamma=1 accuracy=0.4
c=0.01 gamma=10 accuracy=0.4
c=0.01 gamma=100 accuracy=0.4
c=0.01 gamma=1000 accuracy=0.4
c=0.1 gamma=0.001 accuracy=0.4
c=0.1 gamma=0.01 accuracy=0.4
c=0.1 gamma=0.1 accuracy=0.7428571428571429
c=0.1 gamma=1 accuracy=0.8
c=0.1 gamma=10 accuracy=0.4
c=0.1 gamma=100 accuracy=0.4
c=0.1 gamma=1000 accuracy=0.4
c=1 gamma=0.001 accuracy=0.4
c=1 gamma=0.01 accuracy=0.8
c=1 gamma=0.1 accuracy=0.8285714285714286
```

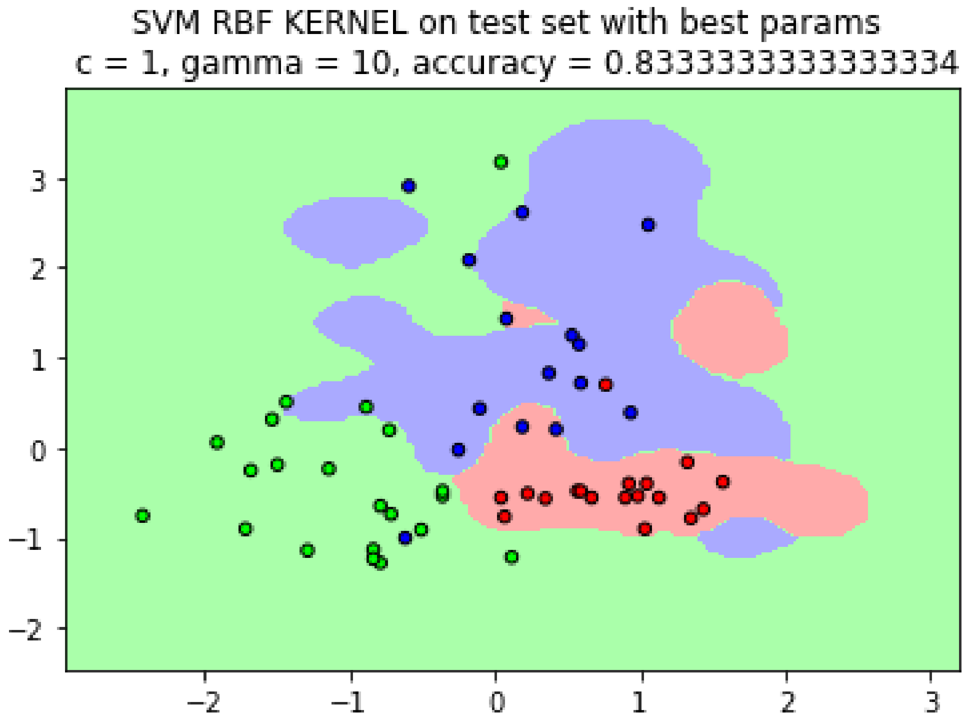
```
c=1 gamma=1 accuracy=0.8571428571428571
c=1 gamma=10 accuracy=0.8857142857142857
c=1 gamma=100 accuracy=0.6285714285714286
c=1 gamma=1000 accuracy=0.4
c=10 gamma=0.001 accuracy=0.7714285714285715
c=10 gamma=0.01 accuracy=0.8285714285714286
c=10 gamma=0.1 accuracy=0.8571428571428571
c=10 gamma=1 accuracy=0.8
c=10 gamma=10 accuracy=0.8857142857142857
c=10 gamma=100 accuracy=0.6571428571428571
c=10 gamma=1000 accuracy=0.45714285714285713
c=100 gamma=0.001 accuracy=0.8285714285714286
c=100 gamma=0.01 accuracy=0.8571428571428571
c=100 gamma=0.1 accuracy=0.8571428571428571
c=100 gamma=1 accuracy=0.8571428571428571
c=100 gamma=10 accuracy=0.8285714285714286
c=100 gamma=100 accuracy=0.6571428571428571
c=100 gamma=1000 accuracy=0.45714285714285713
c=1000 gamma=0.001 accuracy=0.8571428571428571
c=1000 gamma=0.01 accuracy=0.8857142857142857
c=1000 gamma=0.1 accuracy=0.8571428571428571
c=1000 gamma=1 accuracy=0.8
c=1000 gamma=10 accuracy=0.8285714285714286
c=1000 gamma=100 accuracy=0.6571428571428571
c=1000 gamma=1000 accuracy=0.45714285714285713

[GRID RESULT] c=1 gamma=10 accuracy=0.8857142857142857
```

nb: in my implementation if found the same accuracy with different C and Gamma i prefeer to chose the one with lowest C value.

3.3.1 Evaluate best params found with grid search on test set

unlike previous, the model seems to be worse on test set with the best c and gamma founded.



3.4 K-FOLD cross validation (5 split) and grid search

now i use the cross validation to grid-search the best values of c and gamma;

the k-fold cross validation automatically decide the partition that will ensure a better train for doing this i merged train and validation splits in a single split used by kfold.

for simplicity i use the function GridSearchCV from sklearn to perform both cross-fold validation and gridSearch.

now the grid search is executed for each split generated by k-fold and for each tuple of c and gamma values

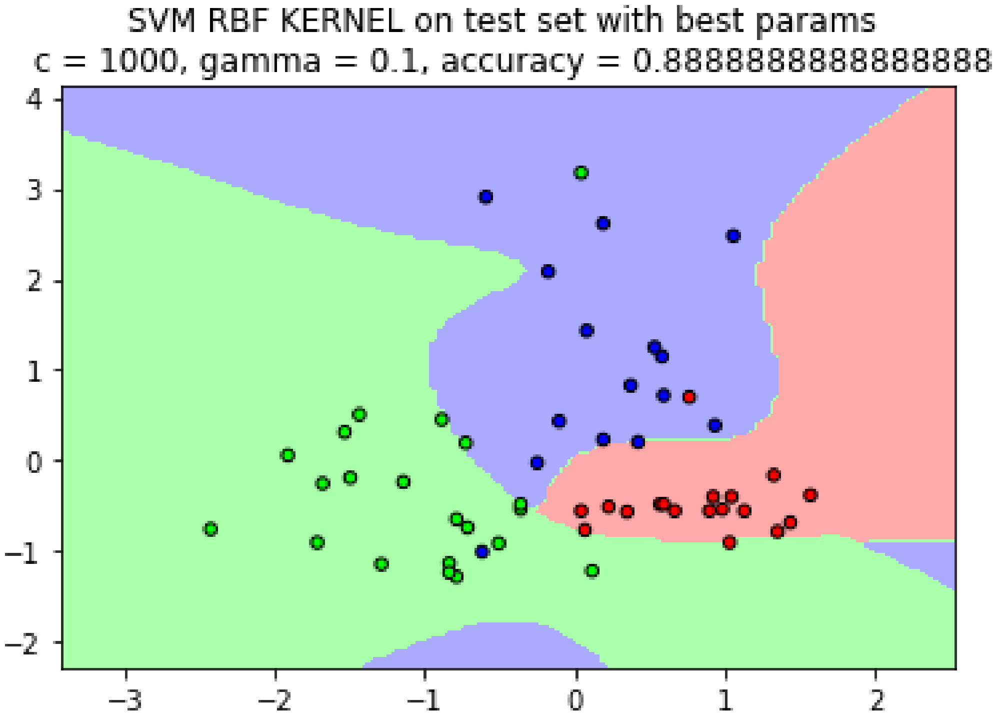
and then the result show us the best values found over iterations of cross-validation

```
[GRID RESULT] {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
```

3.4.1 Evaluate best params found with kfold and grid search on test set

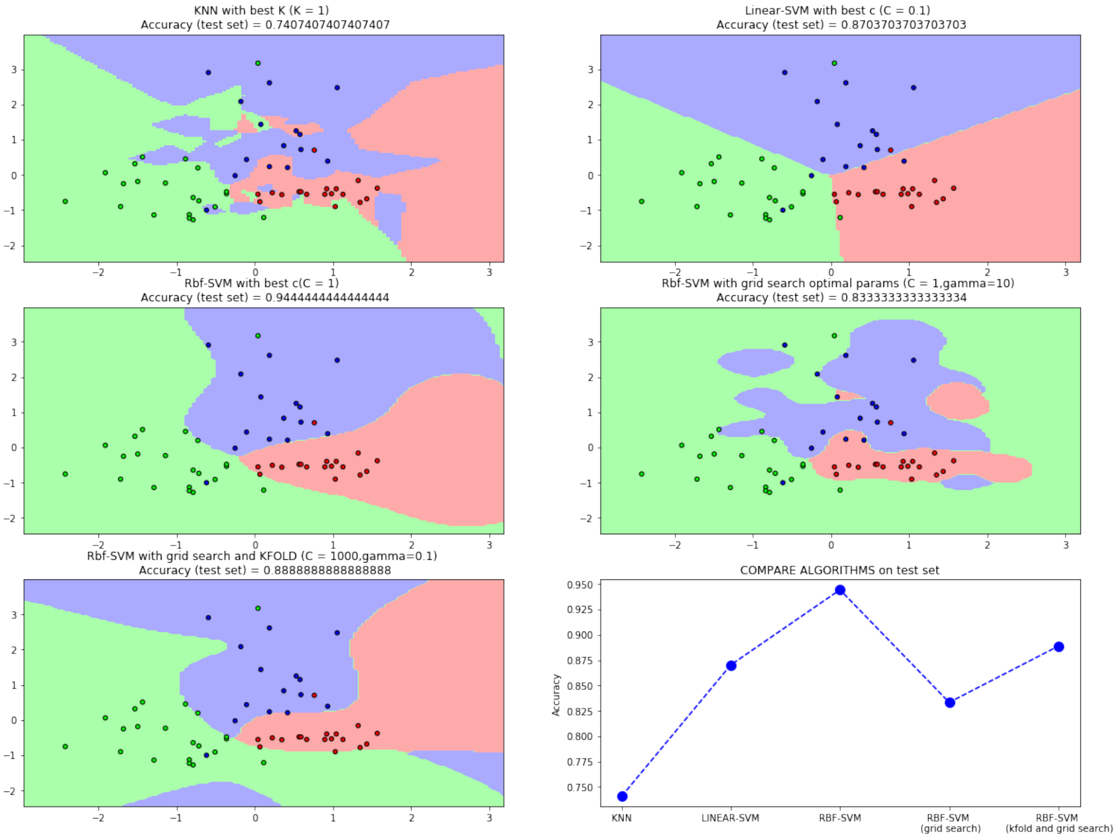
the tuning of the c and gamma parameters seems to be working very good on test set giving an accuracy on average better than earlier.

nb: the data-points in the plot are the test point not the training point



4 ..at the end : Compare all different model on test set

I decided to put on this page a general comparison of the algorithms used in this homework, and a chart to show the accuracy for each model evaluated on test set



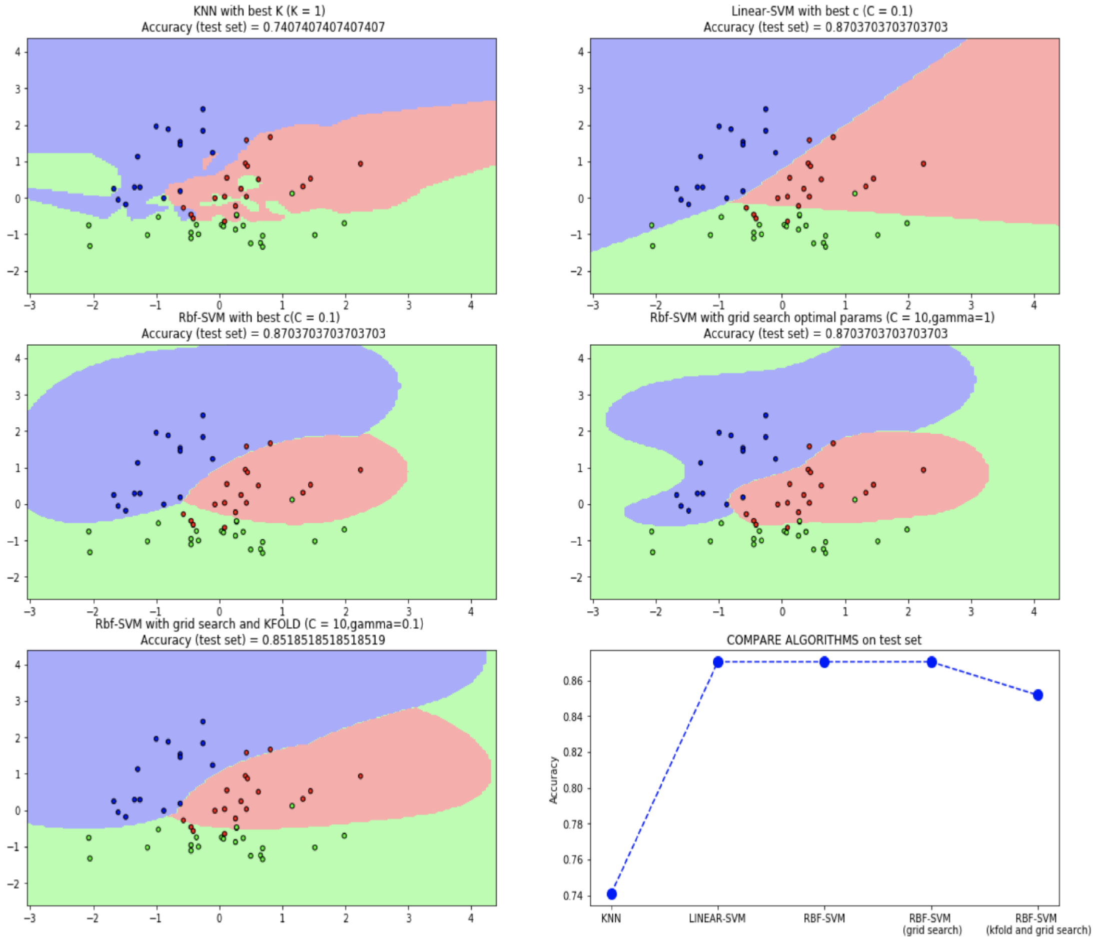
4.1 Extra: differences between knn and svm

in strictly visual terms, we can see more jagged and variable boundaries on the Knn, while in the SVM the boundaries seem more smooth and defined, in this case we found the rbf-svm with a best accuracy on test set, but this as i said early is extremly dipendent on how and wich splits are generated, i make several running with different random_state seed and generally the pattern of the last graph is not respected, sometimes for example, the knn model returns better results than the SVM

4.2 Extra: choose different attributes

i've also tryed with different attributes, model behavior, accuracy and boundaries changing accordingly. (for trying this on code just uncomment the line on first page of code script)

Example with attributes #8 and #9 (instead #0 and #1)


KNN with best K (K = 1)
Accuracy (test set) = 0.7407407407407407
Linear-SVM with best c (C = 0.1)
Accuracy (test set) = 0.8703703703703703
Rbf-SVM with best c(C = 0.1)
Accuracy (test set) = 0.8703703703703703
Rbf-SVM with grid search optimal params (C = 10,gamma=1)
Accuracy (test set) = 0.8703703703703703
Rbf-SVM with grid search and KFOLD (C = 10,gamma=0.1)
Accuracy (test set) = 0.8518518518518519
−3
−2
−1
0
1
2
3
4
COMPARE ALGORITHMS on test set
Accuracy
0.74
0.76
0.78
0.80
0.82
0.84
0.86
KNN
LINEAR-SVM
RBF-SVM
RBF-SVM
(grid search)
RBF-SVM
(kfold and grid search)