Support Vector Machine

Giuseppe L'Erario

Introduction

Support vector machines are supervised learning models used mainly for classification purpose.

In *Support vector machines* objective is to maximize the distance (the **margin**) between the decision boundary (the **hyperplanes**) and the training samples closer to it. These samples are the **support vectors**.

One can find the good balance between bias an overfitting tuning the margins.

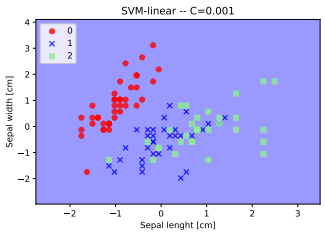
1 Linear Support Vector Machine

Linear SVMs are used for classification of linearly separable datasets. However, with the introduction of a *slack* variable we can relax the linear constraints in order to work on nonlinearly separable data.

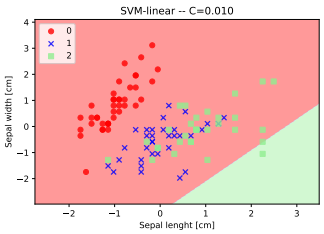
The coefficient of this variable is the **C** parameter, that rules the penalty for a misclassification. Large values of **C** correspond to large penalties and vice versa. To tune the **C** parameter is necessary to control the width of the margin and find the compromise between bias and overfitting.

In this homework we work on the famous *Iris dataset*. Only the first two features of the dataset are imported. Then, the dataset is divided in a train, a validation and a test set. The first two sets are used to train the classifier and tune the parameters, while the test set is used to measure the performance[1].

In fig.1 it is clear how the decision boundaries vary with the **C** parameter. To a low value of **C** (fig.1(a)) corresponds high bias. Increasing **C**, and therefore the penalty for a misclassification, the boundaries become more precise.



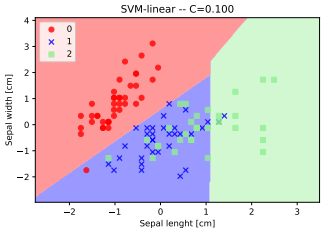
(a) C=0.001



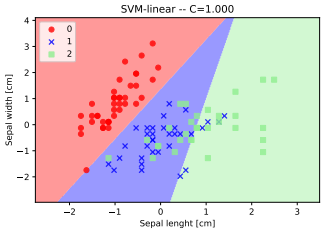
(b) C=0.01

The fig.2 shows how the accuracy varies with the **C** parameter.

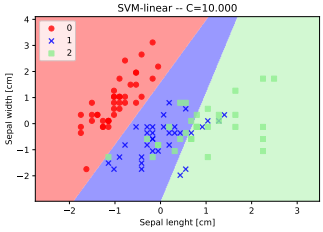
[1] Use the test set to tune the parameters in order to increase the accuracy transforms it in a train set at all.



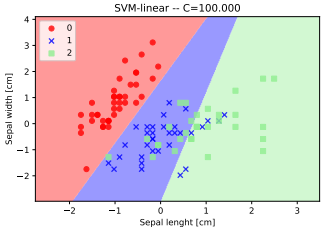
(c) C=0.1



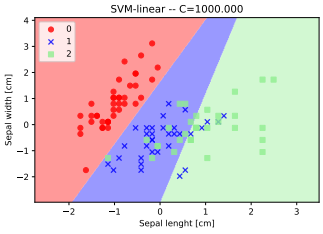
(d) C=1



(e) C=10



(f) C=100



(g) C=1000

Figure 1: Linear SVM.

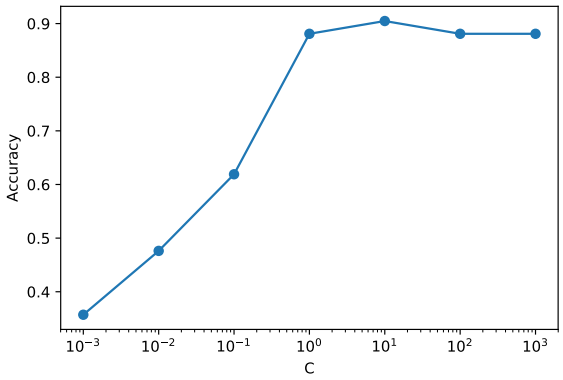


Figure 2: Accuracy - **C** parameter.

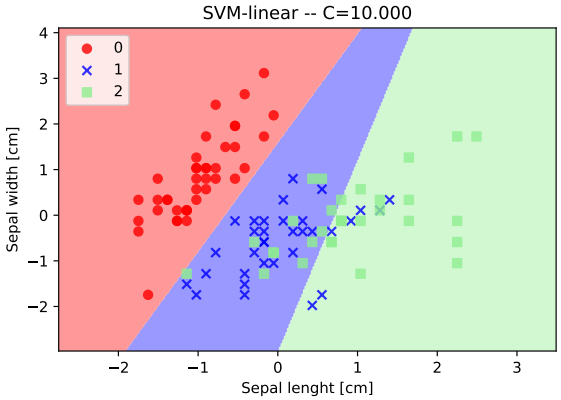


Figure 3: Evaluation on Test set.

The best score obtained on validation set is evaluated on the test set, in fig.3, achieving an accuracy of 71.1%.

2 RBF Vector Machine

To solve a nonlinear problem we transform the training data onto a higher dimensional space with a mapping function and train a linear classifier. Then the same mapping function is used to transform the data to classify.

This approach, based on the dot product between features, is computationally expensive. Here come out the benefit of a kernel function $k(x_i \cdot x_j) = f(x_i) \cdot f(x_j)$ that avoids the use of dot product.

One of the most used kernel function is the **Radial Basis function kernel**:

$$k(x_i \cdot x_j) = e^{-\gamma \|x_i - x_j\|^2} \tag{1}$$

where γ is a parameter to optimize.

Fixed γ

The behaviour on varying of **C** is similar to linear SVM. The difference is in the curved decision boundaries.

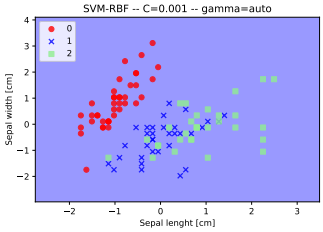
The best score obtained on validation set is evaluated on the test set, in fig.6, achieving an accuracy of 71.4%.

Grid search

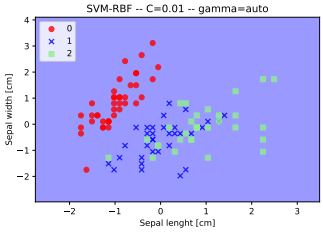
The γ parameter is not anymore fixed. To find the best combination we use a grid search.

The result of the grid search, in tab.1, is that the best parameters are $C = 100$ and $\gamma = 0.1$, as we can see in fig.7.

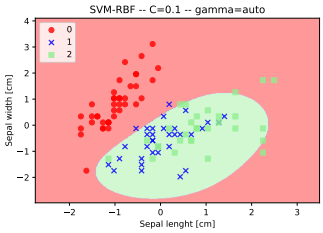
These parameters, evaluated on the test set (see fig.8), give an accuracy of 67%, worse then the accuracy on validation set.



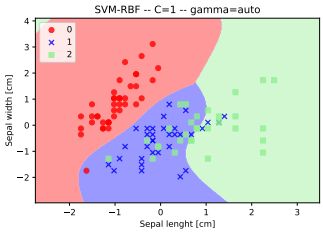
(a) C=0.001



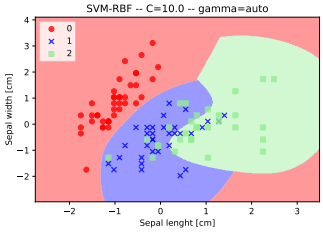
(b) C=0.01



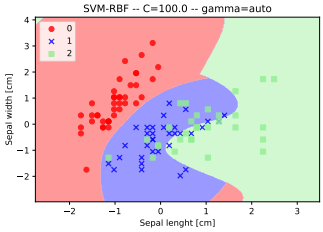
(c) C=0.1



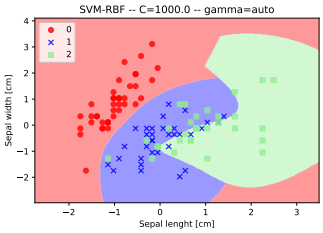
(d) C=1



(e) C=10



(f) C=100



(g) C=1000

Figure 4: RBF SVM with $\gamma = auto$.

3 K-fold validation

In **k-fold validation** the training set is splitted in k folds, where 1 fold is used for validation and the others for training. This procedure is repeated k times (choosing every time one different fold for validation) and in order to obtain k evaluation. Then is computed the average accuracy for the model.

The aim is to find a less sensitive estimation to the partitioning of the set.

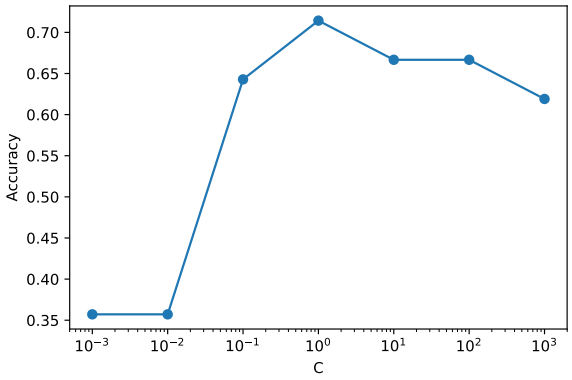


Figure 5: Accuracy - **C** parameter.

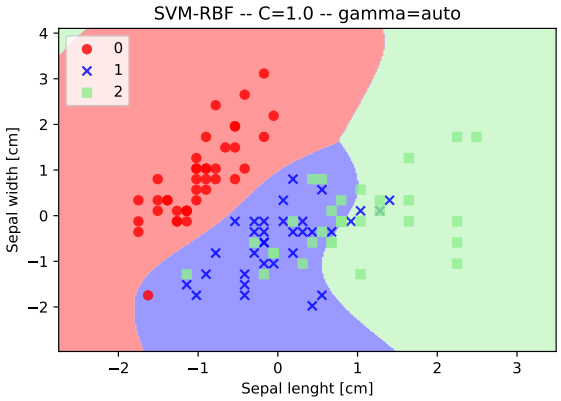


Figure 6: Evaluation on Test set.

C/γ	0.0001	0.001	0.1	10	1000	100000
0.001	21.42	21.42	21.42	21.42	21.42	21.42
0.01	21.42	21.42	21.42	21.42	21.42	21.42
0.1	21.42	21.42	23.82	21.42	21.42	21.42
1	21.42	21.42	69.04	57.14	21.42	21.42
10	21.42	35.71	69.04	52.38	21.42	21.42
100	21.42	69.04	**76.19**	52.38	21.42	21.42
1000	38.09	69.04	71.42	52.38	21.42	21.42

Table 1: Grid Search

From tab.2 and fig.9 we can see that the best accuracy is placed in correspondence of $C = 100$ and $\gamma = 0.1$, the same parameters found without k-fold validation, but with a different accuracy (83.10% vs 76.19%).

It might sound like that the k-fold validation is useless, but it is not. Infact running several simulation the performances measured are less oscillating than

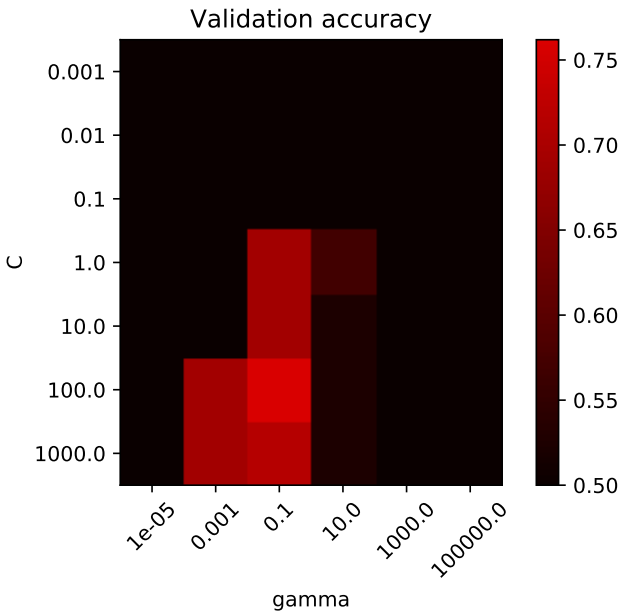


Figure 7: Heatmap for **C** and γ.

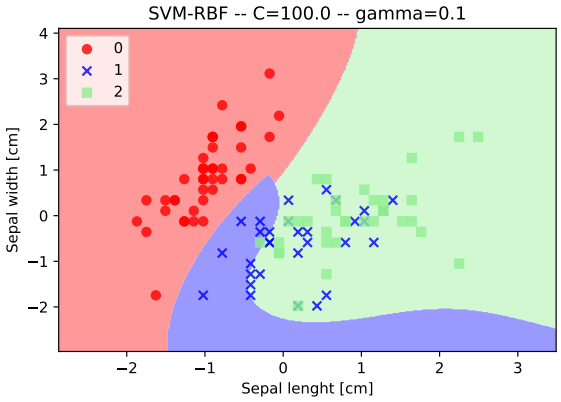


Figure 8: Evaluation on Test set

$\mathbf{C}/\gamma$	0.0001	0.001	0.1	10	1000	100000
0.001	38.09	38.09	38.09	38.09	38.09	38.09
0.01	38.09	38.09	38.09	38.09	38.09	38.09
0.1	38.09	38.09	40.00	38.09	38.09	38.09
1	38.09	38.09	78.09	81.90	67.61	67.61
10	38.09	53.33	78.09	80.00	67.61	61.61
100	38.09	78.09	**83.10**	80.00	67.61	67.61
1000	54.28	80.95	80.95	80.00	67.61	67.61

Table 2: Grid Search with k-fold validation

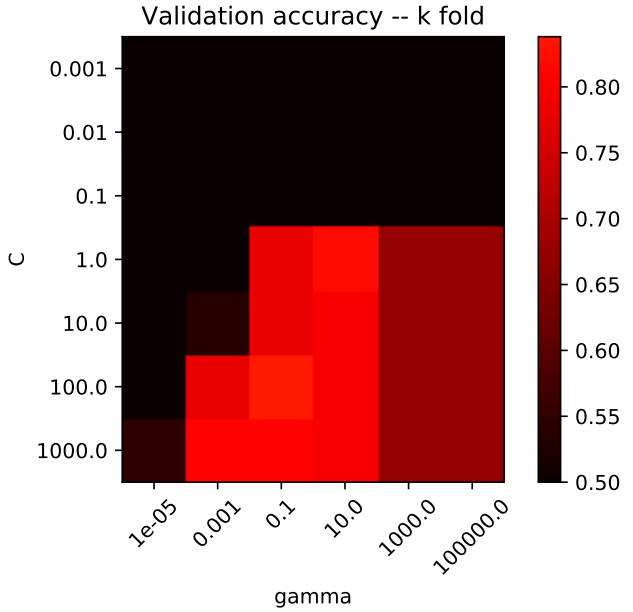


Figure 9: Heatmap for $\mathbf{C}$ and γ

the ones without k-fold validation. This means that the choice of the parameters is more robust.

The evaluation on the test set, in fig.10 gives an accuracy of 67%.

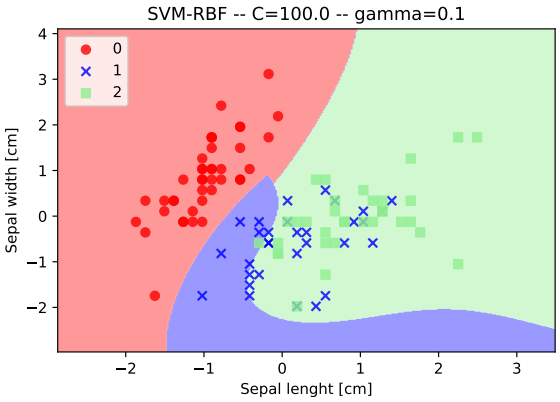


Figure 10: Evaluation on test set