Homework 2 Report **Piero Macaluso s252894**
Machine Learning and Artificial Intelligence 2018/2019 Due Date: 19/01/2019
Prof. Barbara Caputo

1 Linear SVM

In the first part of this homework I trained, validated and finally tested a Linear SVM, using the C with the highest accuracy in validation. The essential part of the code is algorithm 1.

Algorithm 1: Searching the best value of C in Linear SVM

```
    acc = np.empty(7)
    c_i = 1e-3
    c_best = 1e-3
    a_best = 0
    i = 1
    xx, yy = make_meshgrid(x[:, 0], x[:, 1])
    while c_i <= 1e3:
        clf = svm.LinearSVC(C=c_i, random_state=r_state)
        acc[i - 1] = clf.fit(x_train, y_train).score(x_val, y_val) * 100
        if acc[i - 1] > a_best:
            c_best = c_i
            a_best = acc[i - 1]
        ax = fig.add_subplot(4, 2, i)
        plot_data(ax, x_train, y_train, xx, yy, clf)
        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xlabel('Sepal length')
        ax.set_ylabel('Sepal width')
        ax.set_xticks(())
        ax.set_yticks(())
        ax.legend()
        ax.set_title('C=%2.2E A=%2.1f%% ' % (c_i, acc[i - 1]))
        c_i = c_i * 10
        i = i + 1

    fig.suptitle("Linear SVM - C tuning - C_best = %2.2E A_best = %2.1f%%" % (c_best,
        a_best), fontsize=14,
                 fontweight='bold')
```

As we can see from the plots figs. 1 and 2 on page 3 and on page 4, the best accuracy found on validation set was **73.33%**, with C equal to **1e−1**. I noticed that boundaries became more and more precise on the training set with the increment of C.

C is the hyper parameter of SVM that represents the penalty for misclassifying a data point, so it describe how much we want to avoid misclassification during the training. **When C is large**, the optimization will choose a smaller-margin hyperplane, because the classifier is heavily penalized for misclassified data. The larger is C, the better the classification on training set will be. On the contrary, when **C is small**, the optimizer will use a larger-margin separating hyperplane, causing the misclassification of more training data points.

After this training, I tests the data on the test set obtaining a greater accuracy, it goes very well with an accuracy of **88.89%** as we can see in fig. 3 on page 4. I tried to repeat the code a lot of times using a different `random_state` variable, obtaining validation values from 60% to 90%. This may be due to the initial state of the algorithm of SVM or the splitting of the data in training, validation and test sets: maybe one time the is more overfitting on the training data, next time less. We may conclude that this type of validation is not stable and may be influenced by overfitting (see section 4 on page 10).

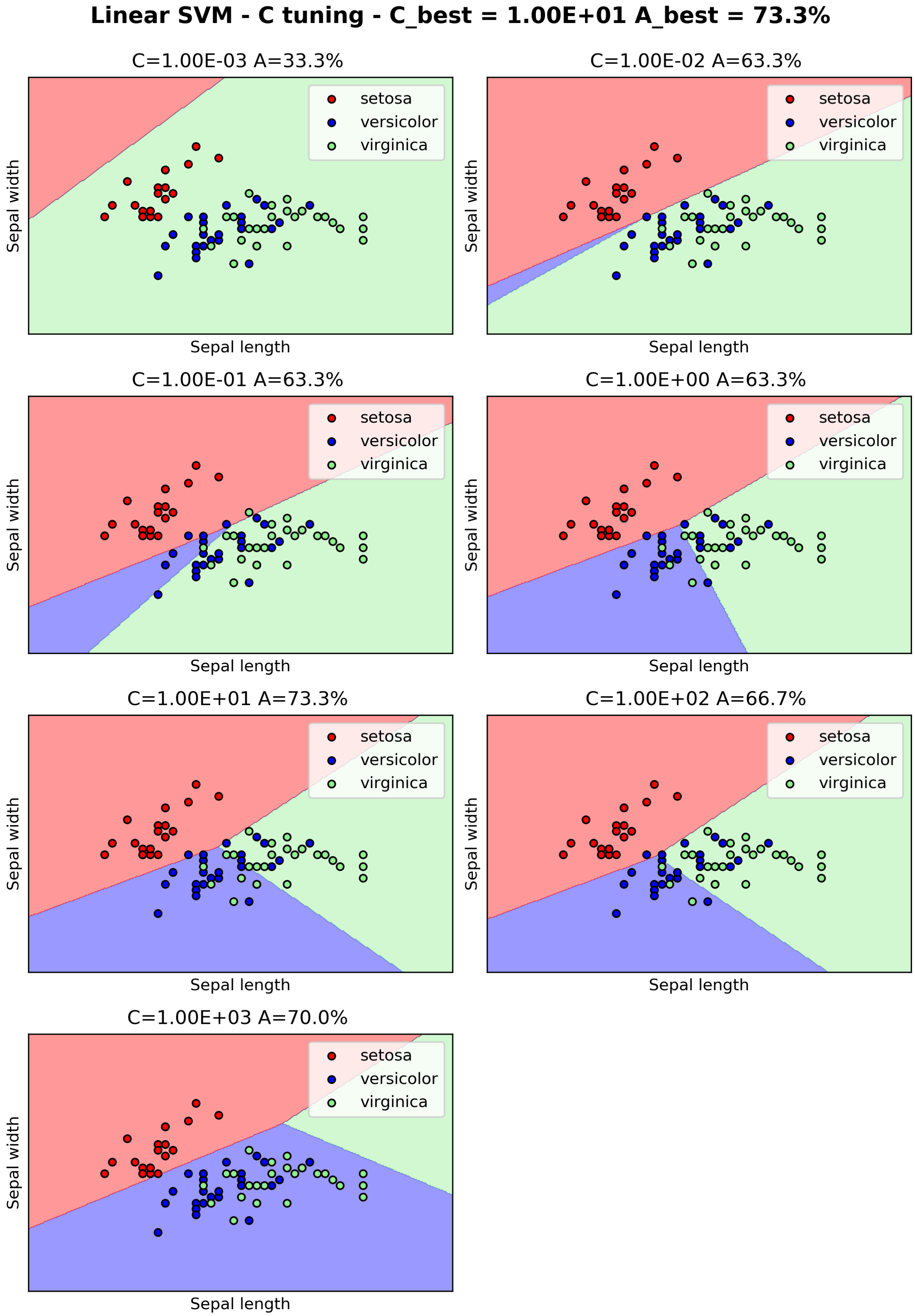

Figure 1: Decision Boundaries changing C in Linear SVM on Training set

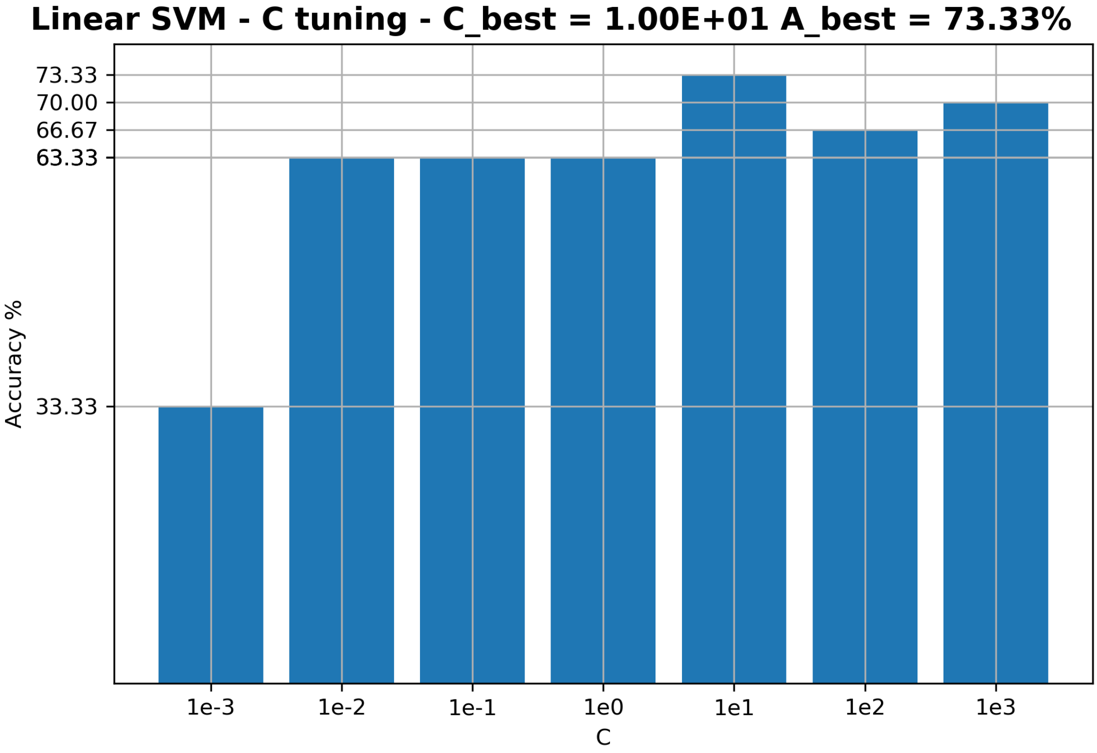

Figure 2: Accuracy changing C in Linear SVM

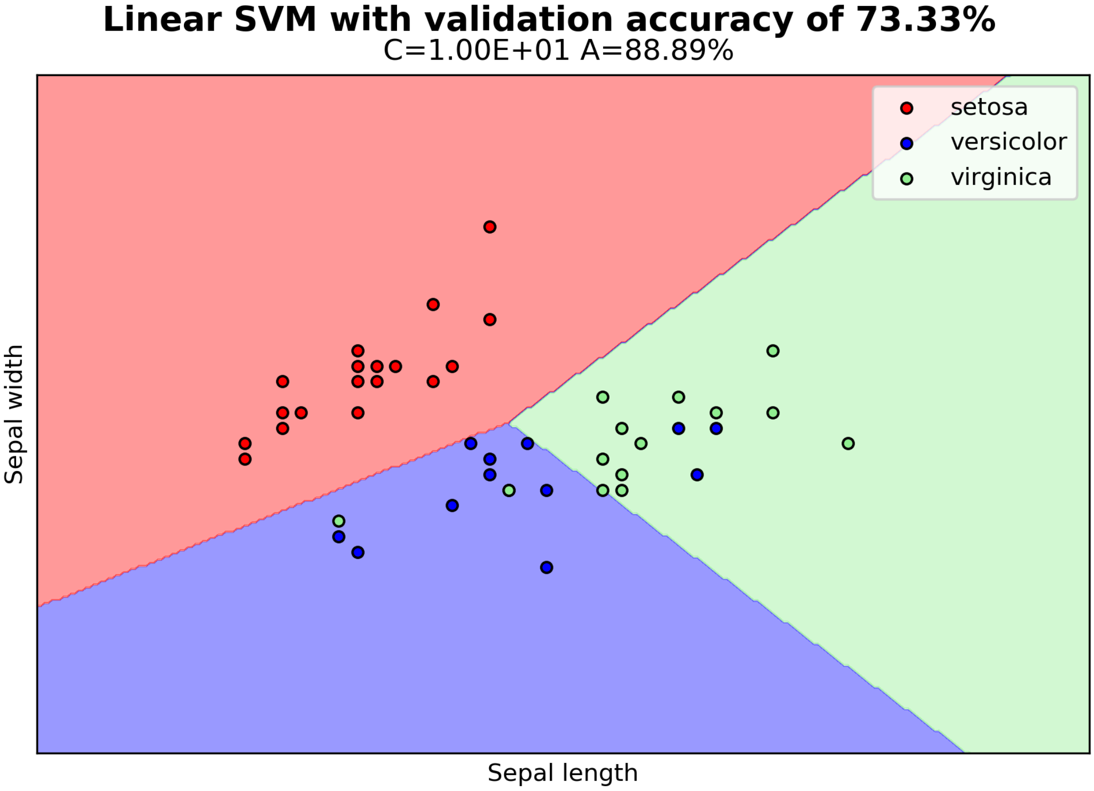

Figure 3: Results on the test set

2 RBF Kernel

In the second part of this homework, I used SVM with **Gaussian RBF kernel** eq. (1).

$$exp(-\gamma \mid\mid x - x' \mid\mid^2) \tag{1}$$

This time I had to find the best pair of C (explained in section 1 on page 1) and γ (Gamma).

Gamma parameter can be explained as the *spread* of the kernel and therefore the decision region. When gamma is **low**, the *curve* of the decision boundary is very low and thus the decision region is very broad. When gamma is **high**, the *curve* of the decision boundary is high, which creates islands of decision-boundaries around data points.

Firstly, I repeated the steps of the previous section in order to find the best accuracy modifying C and using the Gamma calculated by `sklearn` under the hood as we can see in algorithm 2. This time the boundaries are **not as linear as** the ones of Linear SVM because of the introduction of the Gaussian RBF kernel. Thanks to this approach, I found a better training accuracy (**76.67%**) compared to the Linear SVM one, and a test accuracy near to the validation one (**75.56%**) with $\boldsymbol{C = 1e{-}1}$ as we can see in figs. 4 to 6 on pages 6–7.

I used the the code of algorithm 1 on page 1 changing the classifier on line 88 with the one in algorithm 2 .

Algorithm 2: RBF Kernel SVM Classifier

```
        clf = svm.SVC(kernel='rbf', C=c_i, random_state=r_state)
```

Finally, I performed the grid search of the best pair of C and $Gamma$ with algorithm 3. This time I obtained an high accuracy in validation (**80.00%**) and also on the test set (**86.67%**) with a $\boldsymbol{C = 1e0}$ and $\boldsymbol{\gamma = 1e{-}1}$ as we can see in figs. 7 and 8 on page 8.

Algorithm 3: Searching the best value of C and Gamma in RBF Kernel SVM

```
    # Grid Search of C and Gamma
    c = np.array([1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3])
    gamma = np.array(
        [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3,
            1e4, 1e5, 1e6, 1e7, 1e8, 1e9])
    c_best = c.min()
    g_best = gamma.min()
    a_best = 0
    res = np.zeros([c.shape[0], gamma.shape[0]])
    for c_i, i in zip(c, range(0, c.shape[0])):
        for gamma_i, j in zip(gamma, range(0, gamma.shape[0])):
            clf = svm.SVC(kernel='rbf', gamma=gamma_i, C=c_i, random_state=r_state)
            res[i, j] = clf.fit(x_train, y_train).score(x_val, y_val) * 100
            if res[i, j] > a_best:
                c_best = c_i
                g_best = gamma_i
                a_best = res[i, j]
```

I repeated this code a lot of times and, even then, I found a great variation of results in validation and test scores (see section 4 on page 10).

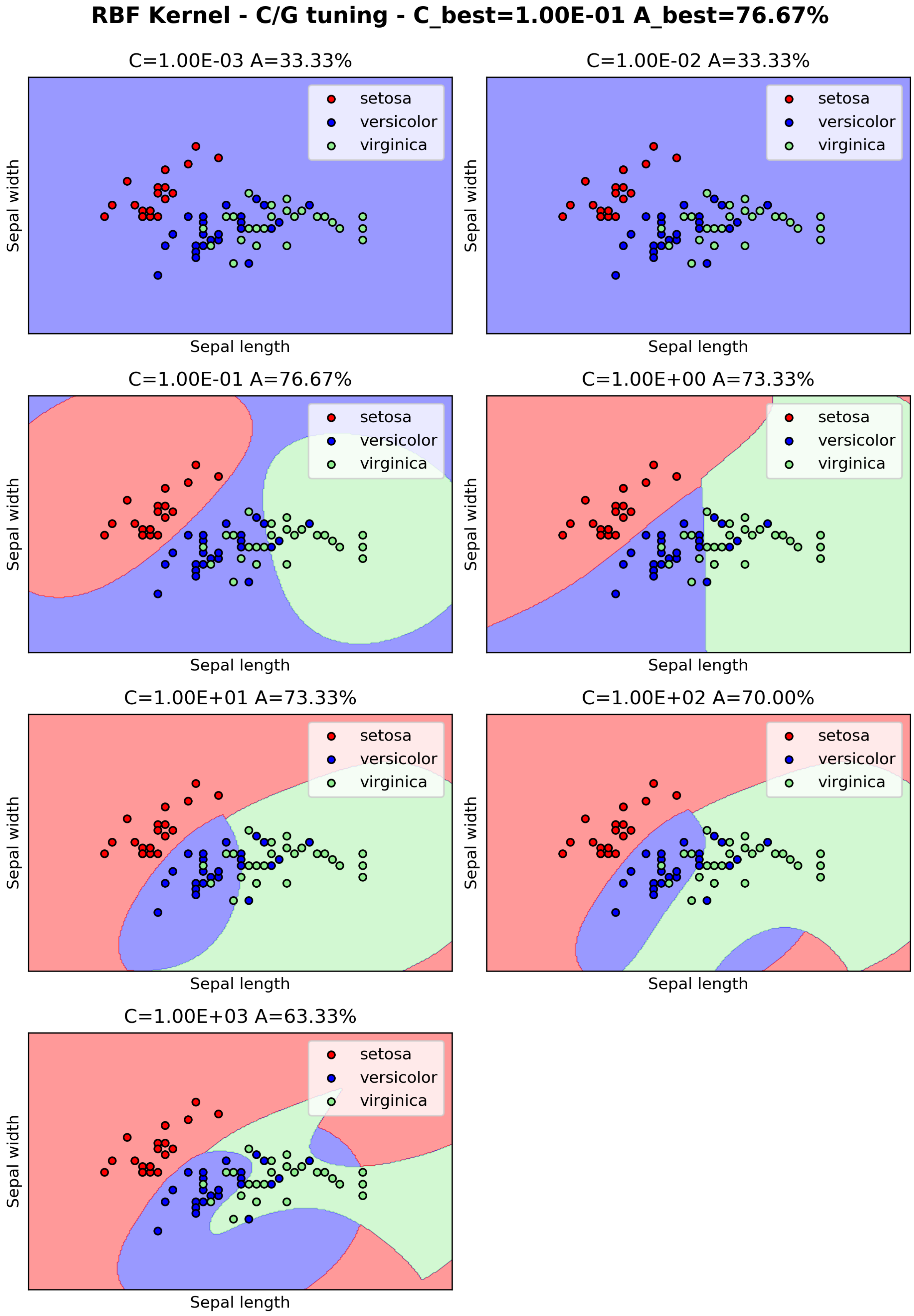

Figure 4: Decision Boundaries changing C in RBF Kernel SVM on Training set

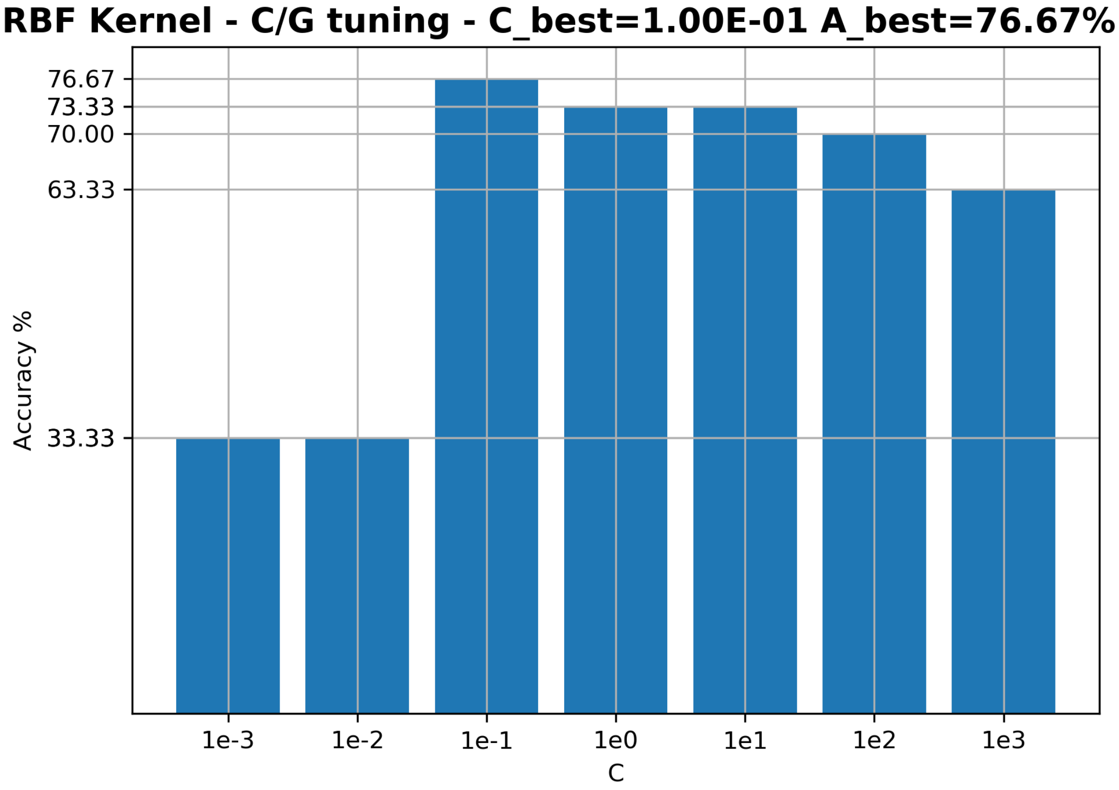

Figure 5: Accuracy changing C in RBF Kernel SVM

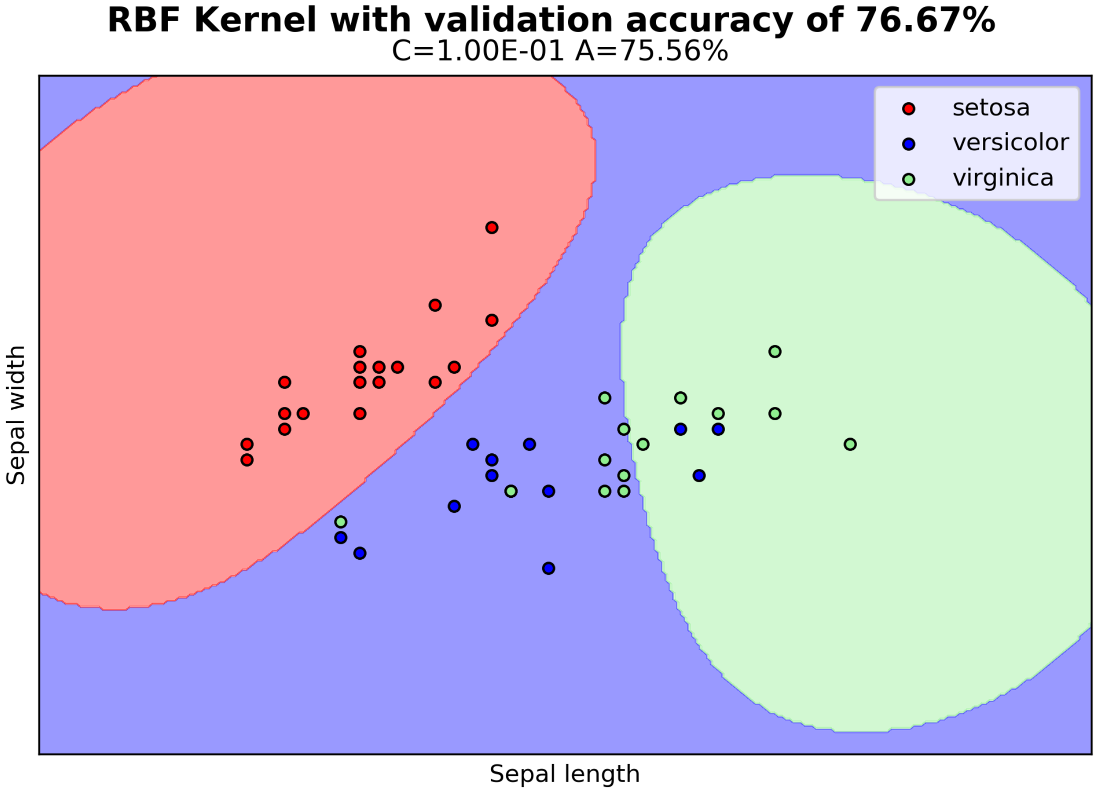

Figure 6: Results on the test set

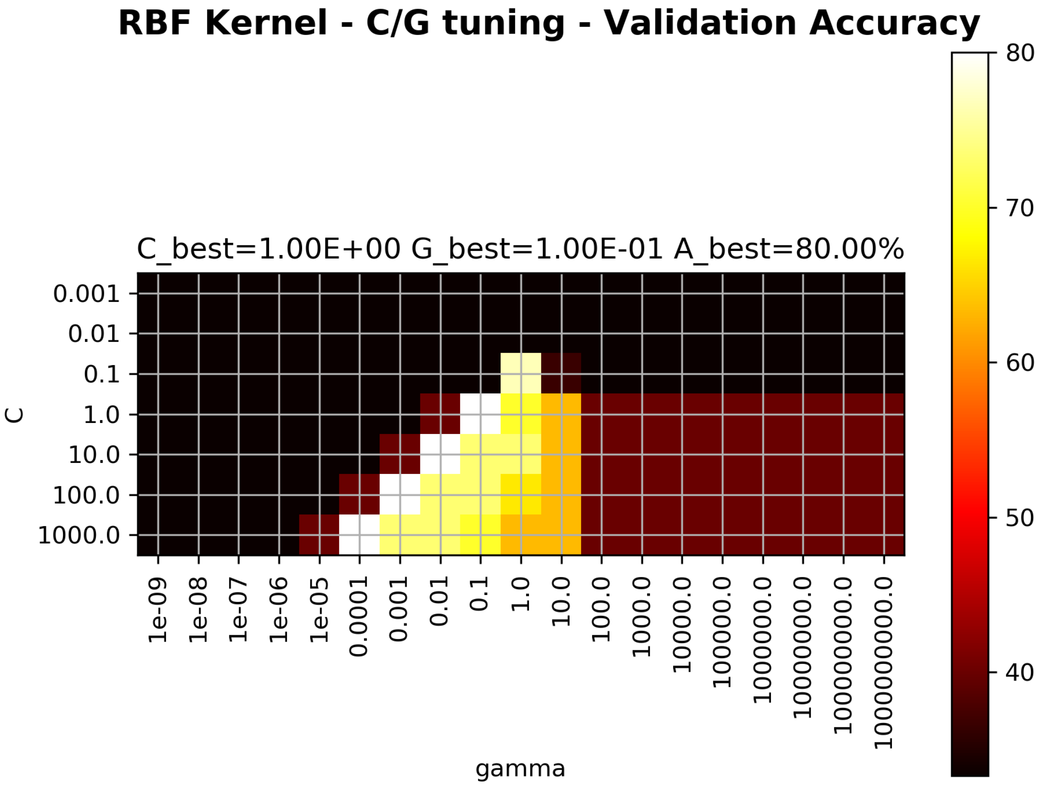

Figure 7: Grid Search of C and Gamma in RBF Kernel SVM on Training set

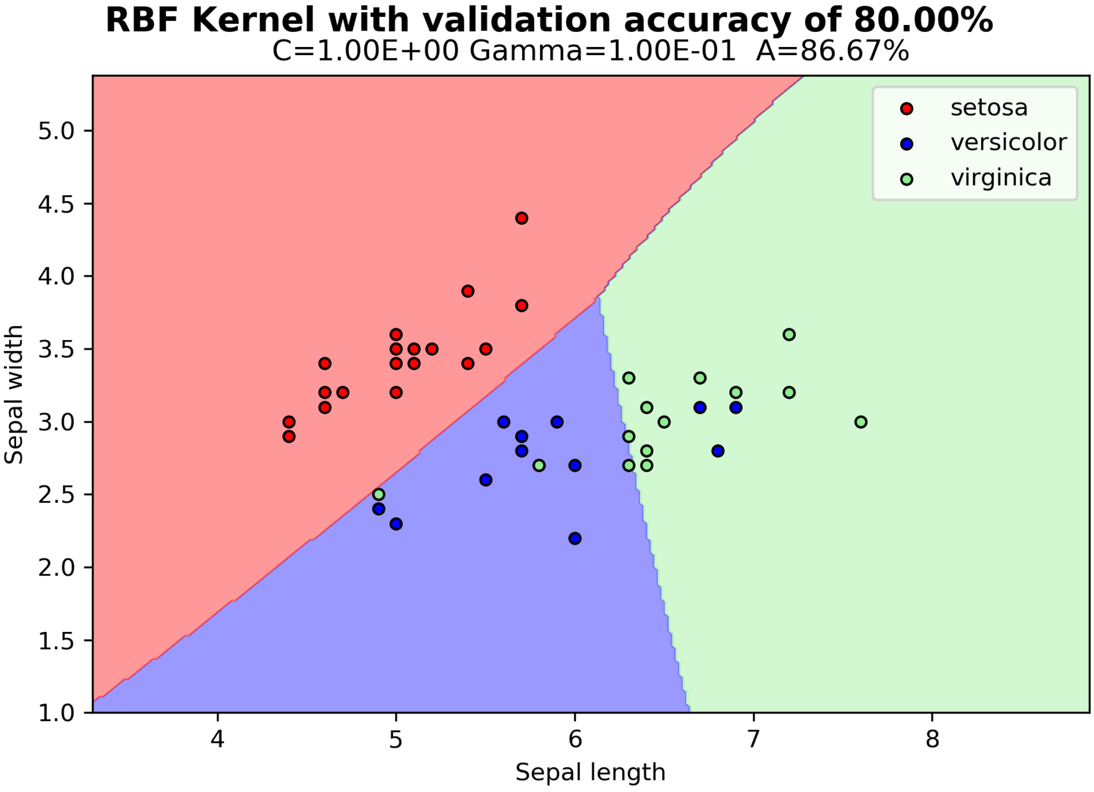

Figure 8: Results on the test set

3 K-Fold

In the last part of the homework, I merged training and validation sets and I perfomed a **k-fold crossvalidation** with $k = 5$ using algorithm 4.

Algorithm 4: Validation using k-fold with $k = 5$

```
# 5-fold validation
c_best = c.min()
g_best = gamma.min()
a_best = 0
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=r_state)
scores = np.zeros([len(gamma), len(c)])
for i, gamma_i in enumerate(gamma):
    for j, c_i in enumerate(c):
        temp = np.zeros(kf.n_splits)
        for k_i, (train_index, test_index) in enumerate(kf.split(x_train)):
            clf = svm.SVC(kernel='rbf', gamma=gamma_i, C=c_i)
            clf.fit(x_train[train_index], y_train[train_index])
            temp[k_i] = clf.score(x_train[test_index], y_train[test_index]) * 100
        acc_av = np.average(temp)
        if acc_av > a_best:
            c_best = c_i
            g_best = gamma_i
            a_best = acc_av
```

I obtained an in validation equal to **76.19%**, but a test accuracy **82.22%** with a $\boldsymbol{C = 1e2}$ and $\boldsymbol{\gamma = 1e{-}3}$ as we can see in fig. 9.

The final score is good, although it is lower than some of the final scores found in previous sections. I tried to run the code multiple times changing `random_state` and I have always found values from 75% to 85% (see section 4 on the next page).

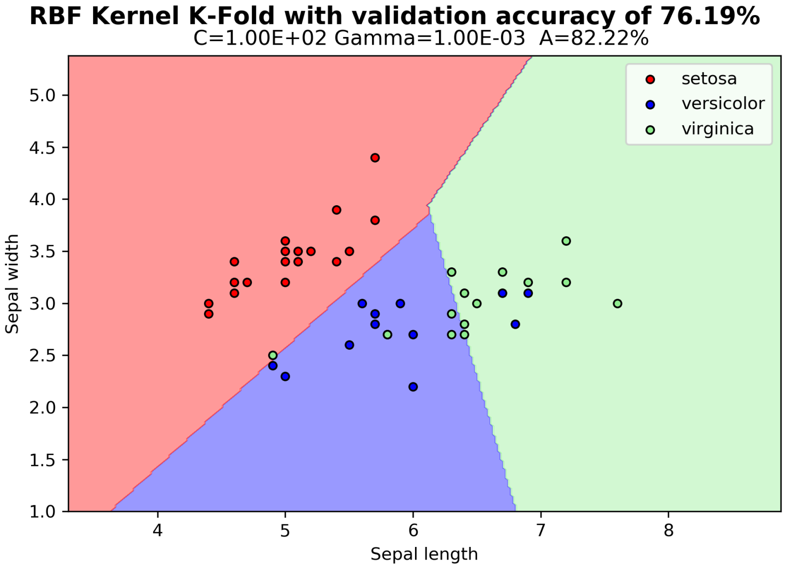

Figure 9: Results on the test set

4 Final Comments

Because of the continuous variation of results for different `random_state`, I decided to implement a simple Python script (`testing.py`) to calculate the average of the results of the three main type of parameter tuning in order to find something remarkable. The result obtained over 200 repetitions can be found in tables 1 and 2.

Table 1: Accuracy distribution

Accuracy (%) (200 rep.)	AVG Validation	STD Validation	AVG Test	STD Test
Linear SVM	≈ 80	≈ 7	≈ 73	≈ 7
RBF Kernel SVM	≈ 82	≈ 6	≈ 76	≈ 6
RBF Kernel SVM (k-fold)	≈ 81	≈ 2	≈ 78	≈ 5

Table 2: Min and Max

Accuracy (%) (200 rep.)	MIN Validation	MAX Validation	MIN Test	MAX Test
Linear SVM	≈ 60	≈ 95	≈ 50	≈ 95
RBF Kernel SVM	≈ 60	≈ 95	≈ 60	≈ 95
RBF Kernel SVM (k-fold)	≈ 75	≈ 87	≈ 60	≈ 95

Now we have something that could be noteworthy: the first two type of parameter tuning algorithms have a larger variance than the last one. Therefore, the validation averages are almost the same, but the test ones no. Indeed, we can find an higher test accuracy on the third type.

From these results, we can conclude that the validation done with k-fold crossvalidation is more stable than the other ones and it is useful to avoid overfitting problem.

5 Code Execution

5.1 Requirements

- Python 3
- All dependencies in `requirements.txt`.

  `$ pip install -r requirements.txt` to install them

5.2 Usage

- `$ python main.py`

  Execute the code

5.3 Reproducibility

In order to reproduce the same data for this experiment you have to change the global variable `r_state` (line) from `None` to 252894 which is my badge number.

Attachments

- Source Code:
  - `main.py`
  - `requirements.txt`
  - `testing.py`