Gabriele Tiboni
S276241

Report for Homework #1

MACHINE LEARNING AND DEEP LEARNING COURSE

1. Introduction

In the following homework, it is asked to check how some traditional machine learning models behave in a classification task with a relatively simple dataset: the **wine** UCI dataset. In particular, it will be checked how the K-Nearest neighbors model and SVM's (both linear and kernel versions) can be hypertuned on the validation set and how their decision boundaries will change with respect to the change in hyperparameters.

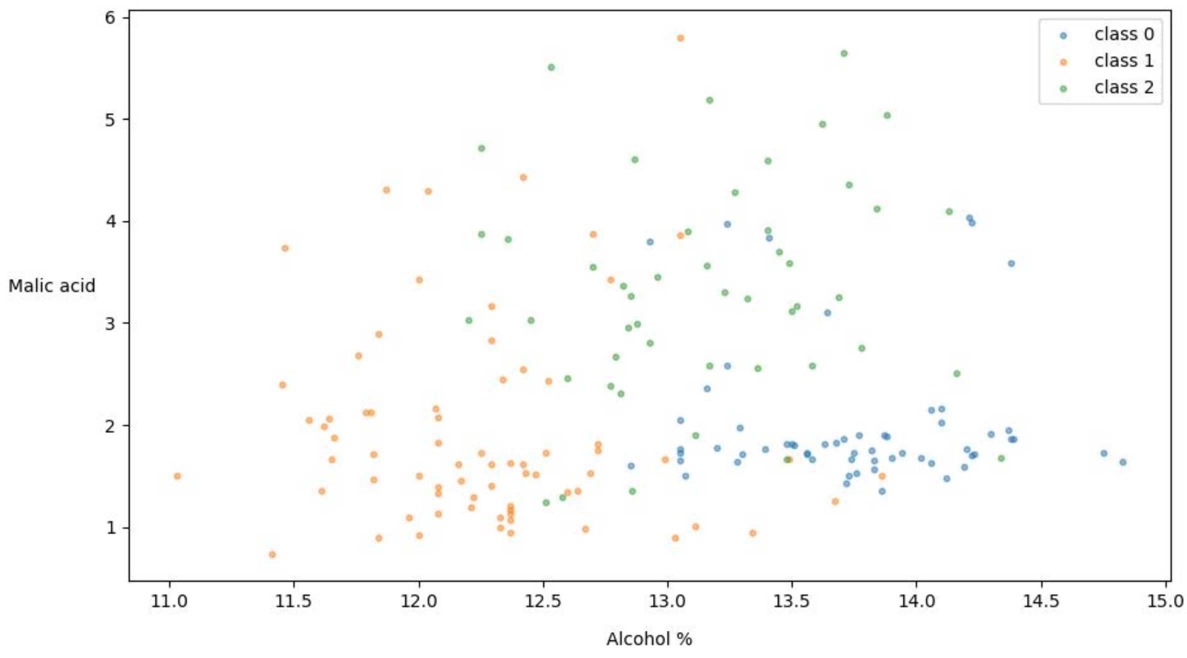
For the entire work, only the first two features of the wine dataset will be taken into account, corresponding respectively to the amount of Alcohol and Malic acid present in each observation (wine).

Before proceeding with the assignment, a brief data exploration task has been performed to better analyze the given dataset:

- Number of observations: **178**
- Number of classes: **3**
- Number of observations per class: **[59, 71, 48]**
- Feature analysis:

	Alcohol	*Malic acid*
Mean	**13.00**	**2.34**
Standard deviation	**0.81**	**1.12**
Min; Max	**11.03; 14.83**	**0.74; 5.80**

- 2D representation of the observations:



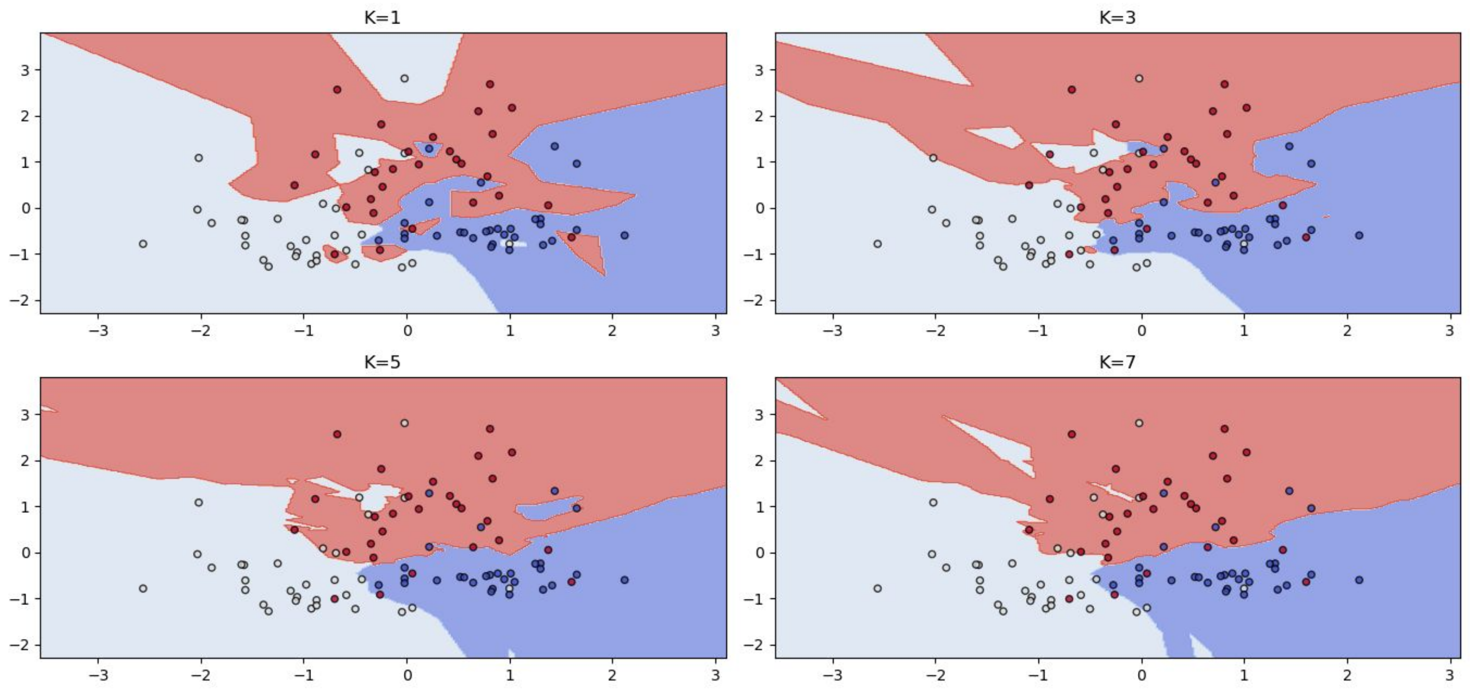
In a real world example, more features should be considered and analysed together with the first two, and other preliminary tasks such as outlier detection and correlation analysis between features should also be performed, in order to optimally preprocess the dataset for our purposes.

Finally, in the preprocessing phase, the dataset has been split in training, validation and test set, with respectively proportion of **5:2:3**.
Note that, given the low amount of samples in the dataset, a fixed random_state for splitting the points was set, in order to make the results in the report consistent among the different classifiers (same training, same validation, same test sets). However, this is not a good practice in a real world example, as cross-validation should instead be used for performing a gridsearch when dealing with small datasets. This issue will be carried on throughout the report, by showing that a different choice of training and validations sets can lead to very different results for small datasets.

2. K-Nearest neighbors model

For the application of the K-nearest neighbors model, four different values of "K" have been taken into account for the hyperparameter tuning phase: **1, 3, 5** and **7**.

The following figures show the decision boundaries obtained by training the KNN model (KNeighborsClassifier in scikit-learn) respectively with the different values of K:



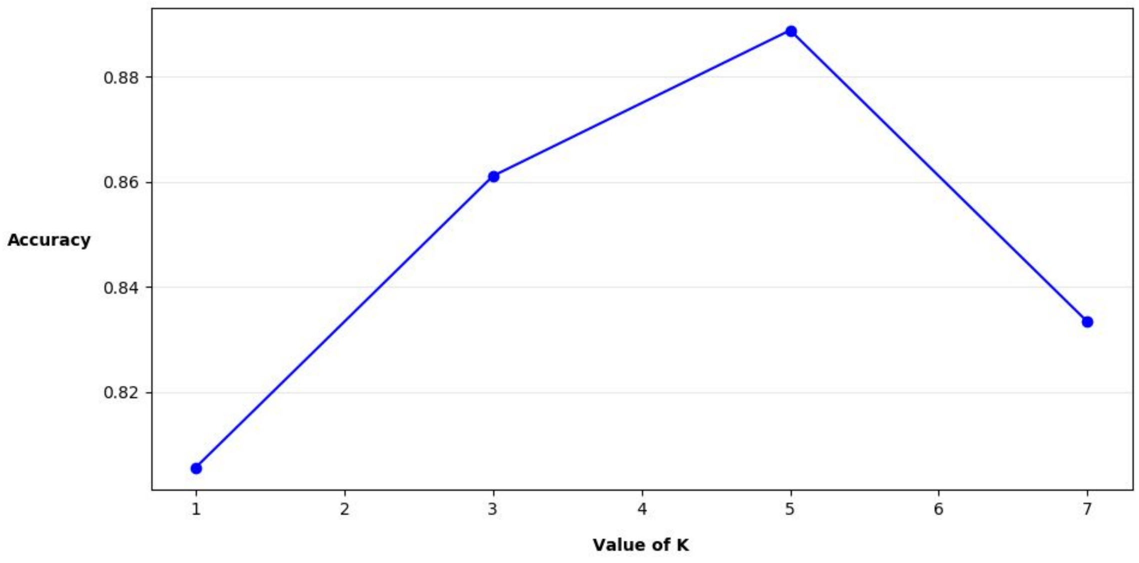
Only the training set is shown in the above figures.

It is firstly important to note that both features have been standardized (scaled to have mean=0 and std=1) as the KNN model is NOT scale-invariant (based on distances).
As shown, increasing the number of K allows for a better generalization, dividing almost smoothly the 2D space into three parts (see figures K=5 and K=7). Instead, when K is equal to one, all samples in the training set are correctly classified (even noisy observations) and the model is most likely not generalizing well for new unseen data (overfitting). The simplest model with K=1 did indeed obtain the lowest accuracy when measured on the validation set.

All the corresponding accuracies, calculated on the validation set, are as follows:

- K=1: **81%**
- K=3: **86%**
- K=5: **89% (best K found for this split)**
- K=7: **83%**

Or summarized in the graph below:

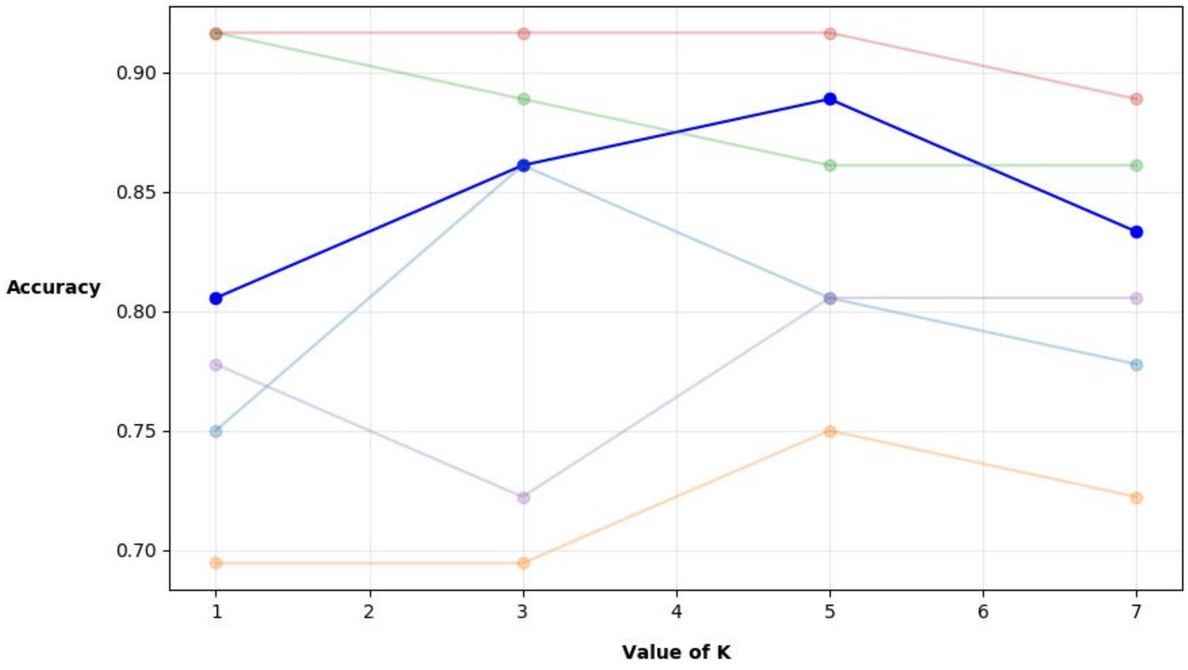


Finally, the best value found for K has been used to train and evaluate our final model. In particular, the training set and the validation set combined have been used for training, in order to use as much data as possible.
Once again, the features of this new combined set have been standardized to have zero mean and variance equal to 1.

Final accuracy obtained on the test set: **83%**

This value is somewhat below average with respect to the accuracy values we got during the hyperparameter tuning process, but still acceptable in terms of generalization. It might indicate that a little bit of overfitting occured, but it might also happen due to how the test set has been chosen. In fact, as the splitting was performed randomly, there are no guarantees that our test set will have the same distribution of the other two sets, or that it will not contain a high number of noisy observations. A cross-validation gridsearch should be performed in order to minimize overfitting during the hyperparameter tuning phase, and a stratified sampling technique can be used in order to extract a test set that is balanced in terms of class distribution. Of course, increasing the number of samples would be the best thing, if possible.

In order to show how different the results can be with respect to a different random split, a new graph has been drawn showing the accuracies obtained earlier (on the validation set) together with the accuracies obtained with 5 new random splits of the dataset:



Values of accuracy obtained on the validation set during the hyperparameter tuning process, with 6 different random splits of the initial dataset.

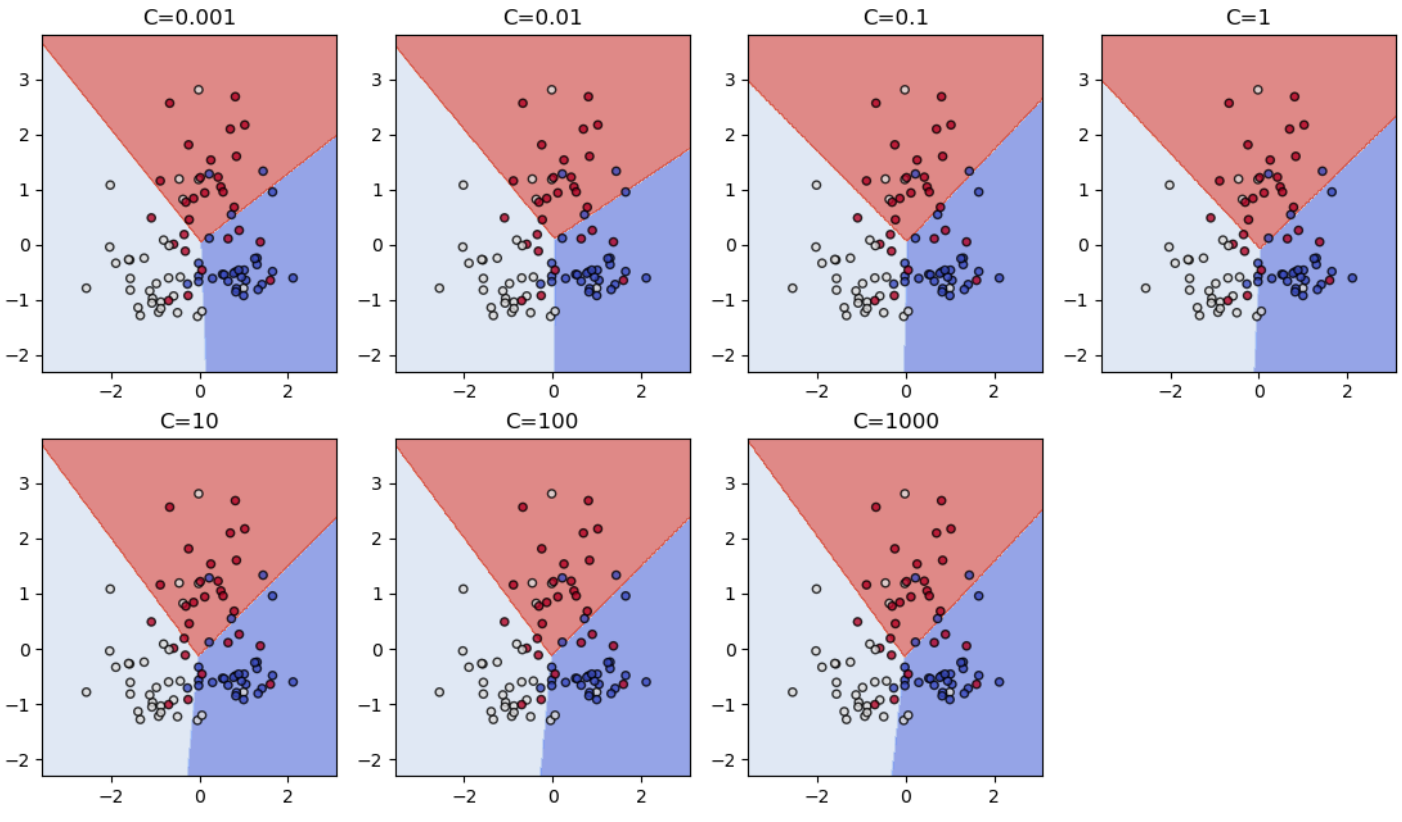
For different splits, we do get different best values of k and very different accuracies, both on the validation set (as shown in the graph above) and on the test set in the final evaluation.

3. Linear SVM model

For the Linear Support Vector Machines model, the regularization parameter C has been considered as the only hyperparameter to be tuned. In particular, the following values for C have been considered: **0.001, 0.01, 0.1, 1, 10, 100, 1000**.

For consistency, the training, validation and test sets are the same as before. Once again, both features have been standardized to mean 0 and variance 1 as it is highly recommended to do so also when dealing with SVM's (see [3]).

The following figures show the decision boundaries obtained by training the Linear SVM (LinearSVC in scikit-learn) model respectively with the different values of C:



Only the training set is shown in the above figures.

The parameter C in the optimization problem of a SVM model indicates the strength of the regularization applied. It is inversely proportional to the regularization itself, as in the optimization problem it is applied on the minimization of the empirical risk instead of on the L2 penalty:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$
$$s.t.\ y_i\left[\langle x_i, w\rangle + b\right] \geq 1 - \xi_i \ and\ \xi_i \geq 0$$

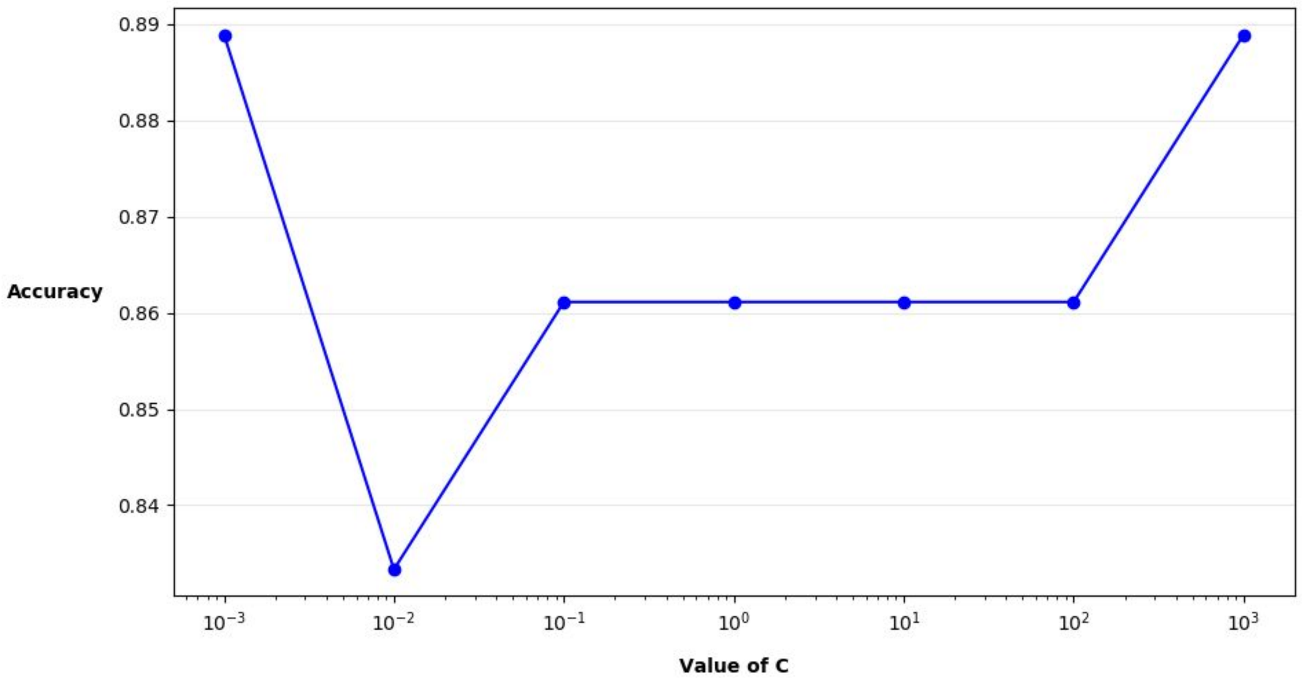
A lower value of C, then, corresponds to a relatively simpler model, by allowing a smaller norm of "**w**", a higher margin, more support vectors and more samples in the training set to be missclassified. On the other hand, increasing the value of C leads to a more complex model, with a higher norm of "**w**", smaller margin, fewer support vectors, smaller overall loss function (hinge loss) but higher risk of overfitting.
Basically, we wish to reduce C when we have a high number of noisy observation and we wish to generalize better; instead we wish to increase C when we really trust our training set to be distributed somewhat like the actual probability distribution of the data studied.

In our case, it's not very easy to tell the differences between the boundaries in the figures above, but we do notice - in the bottom side of the images - that more blu points get included into the blue class as we increase the value of C (decrease the strength of the regularization).

The accuracies corresponding to each value of C, calculated on the validation set, are as follows:

- C=0.001: **89%**
- C=0.01: **83%**
- C=0.1: **86%**
- C=1: **86%**
- C=10: **86%**
- C=100: **86%**
- C=1000: **89%**

Or summarized in the graph below:



C=0.001 and C=1000 obtained the same accuracy for our training set. Again, this might be due to the fact of having a random split of the dataset and too few samples.

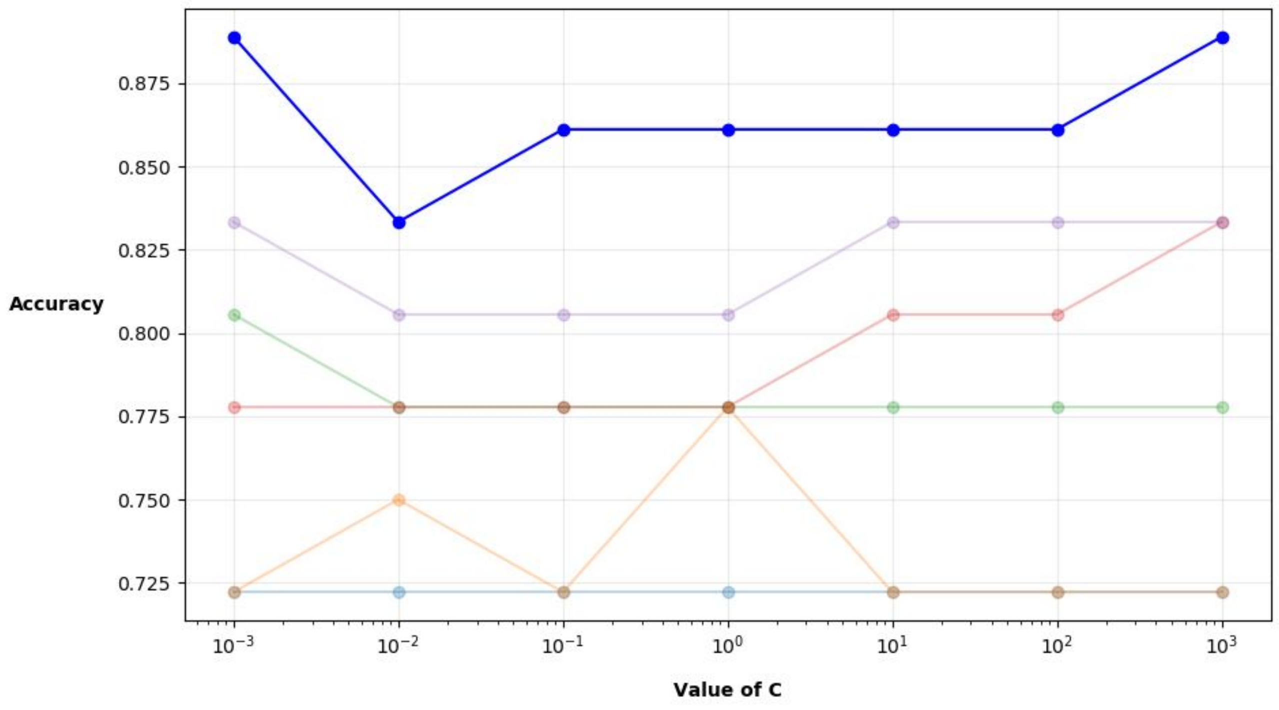
The value 1000 has been chosen as the best hyperparameter, as the graph shows an increasing trend in that direction, but no general rule is given in these cases. It is also important to note that it would not be possible to choose among the two values the one which obtains the best accuracy on the final evaluation (test set), as it would result in using the test set for tuning the hyperparameters of the model itself, which is not allowed.

Finally, the value of C=1000 has been used to train and evaluate the final model. Again, now the training set and the validation set combined have been used for training, and both features being standardized.

Final accuracy obtained on the test set: **83%**

The same considerations on the final result stated above for the KNN model are valid. The model gave a similar accuracy to the one with respect to the validation set but, given the small dataset and the holdout gridsearch technique, the final evaluation we obtained cannot be so precise. A cross-validation gridsearch technique (or even leave-one-out) should be performed with small datasets. Also note that the best value C should take into account the number of samples in the training set (empirical risk minimization) and change accordingly when evaluating on the test set ([1]). However, this correction is not in the scope of this work.

Finally, a graph showing also other 5 different random splits has been drawn, with accuracies w.r.t. the validation set:



Values of accuracy obtained on the validation set during the hyperparameter tuning process, with 6 different random splits of the initial dataset.

For different splits we get different best values of C and very different accuracies, both on the validation set (as shown in the graph above) and on the test set in the final evaluation.

4. SVM model (RBF kernel)

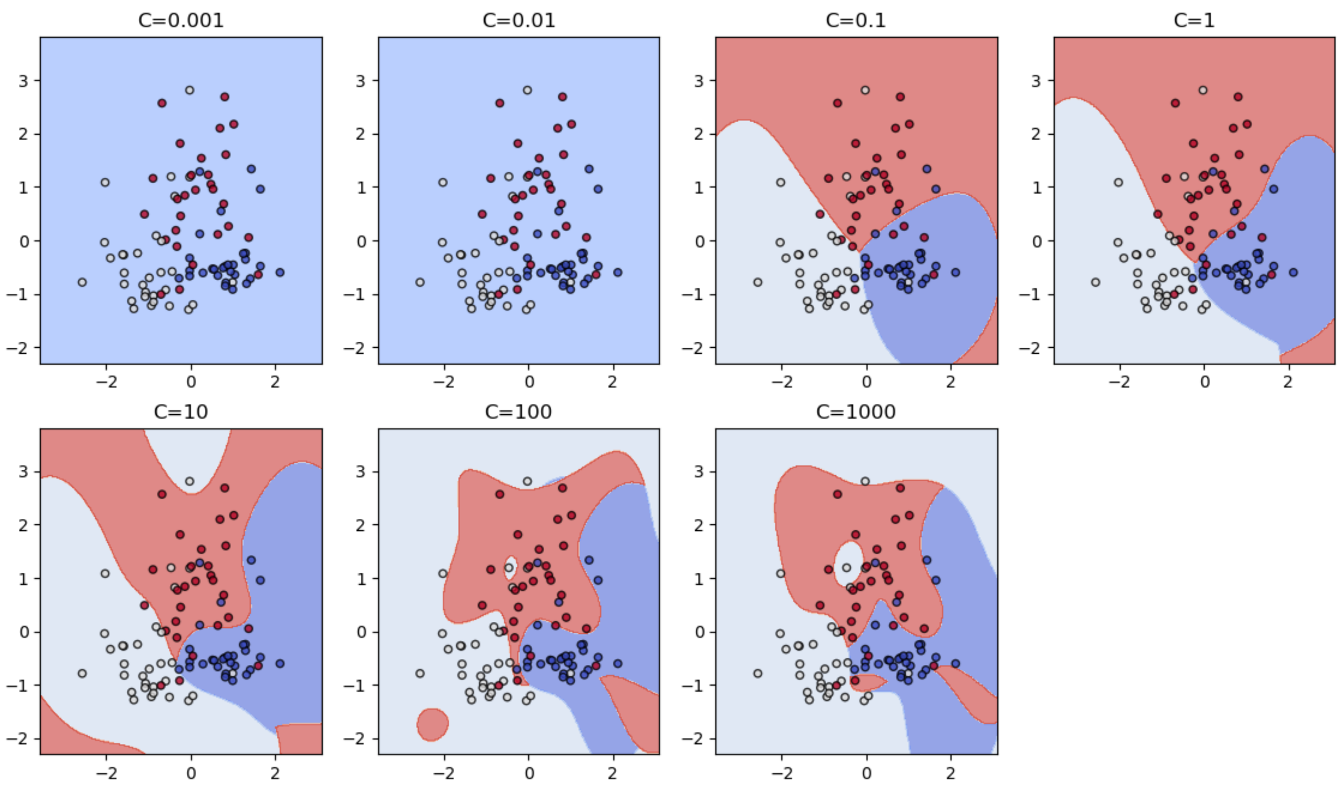
The RBF Kernel allows us to automatically project our points onto a higher-dimensional space and compute their scalar products, which end up being somewhat proportional to how close points are in our lower initial subspace.

Indeed, the RBF Kernel takes as input two data points and returns:

$$k(\overrightarrow{x_i}, \overrightarrow{x_j}) = \exp(-\gamma \|\overrightarrow{x_i} - \overrightarrow{x_j}\|^2)$$

This new non-linear operator, replacing the scalar product in the linear case, allows us to find a non-linear decision boundary for our classification problem, still using the same optimization problem as before. Note that we do not have any knowledge on the mapping function that projects our data onto the higher-dimensional space, so this time we will never know what the actual hyperplane in higher space looks like, nor what the perpendicular vector “w” is. The support vectors are instead take into account for computing the decision function and the classification itself (dual opt. problem).

The same values for C as before have been used for the regularization tuning on the validation set using RBF kernel version of SVM (SVC in scikit-learn), and the results are as follows:



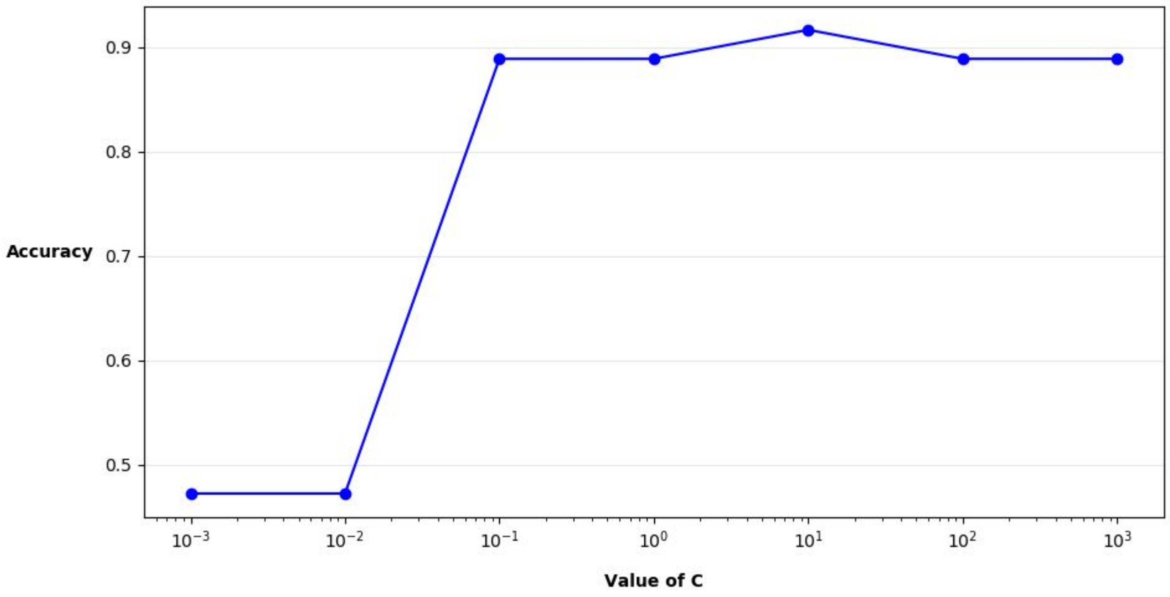
Only the training set is shown in the above figures.

As we know, choosing a small value for C results in a relatively simpler model that allows for missclassifications of the training set, and in our case it ended up predicting only a single class (the majority one in the training set) already for C=0.01. As we increase C instead, we are fitting more the model on our data, and eventually overfitting it (the white dot in the middle of the above figures greatly shows this phenomenon as C changes).

The accuracies corresponding to each value of C, calculated on the validation set, are as follows:

- C=0.001: **47%**
- C=0.01: **47%**
- C=0.1: **89%**
- C=1: **89%**
- C=10: **92% (best C found)**
- C=100: **89%**
- C=1000: **89%**

Or summarized in the graph below:



The best value for C found was 10, and it does seem to generalize fairly well if we look at the corresponding decision boundaries.

Final accuracy obtained on the test set with the best C found: **87%**

The final evaluation obtained is higher with respect to the linear version of the classifier, meaning that, given our random split, we are getting a little advantage by using a non-linear decision surface instead of a straight line between classes.

Later, also the hyperparameter “gamma” of the RBF Kernel has been taken into account for the tuning phase, in hope to find an even better model for our data.

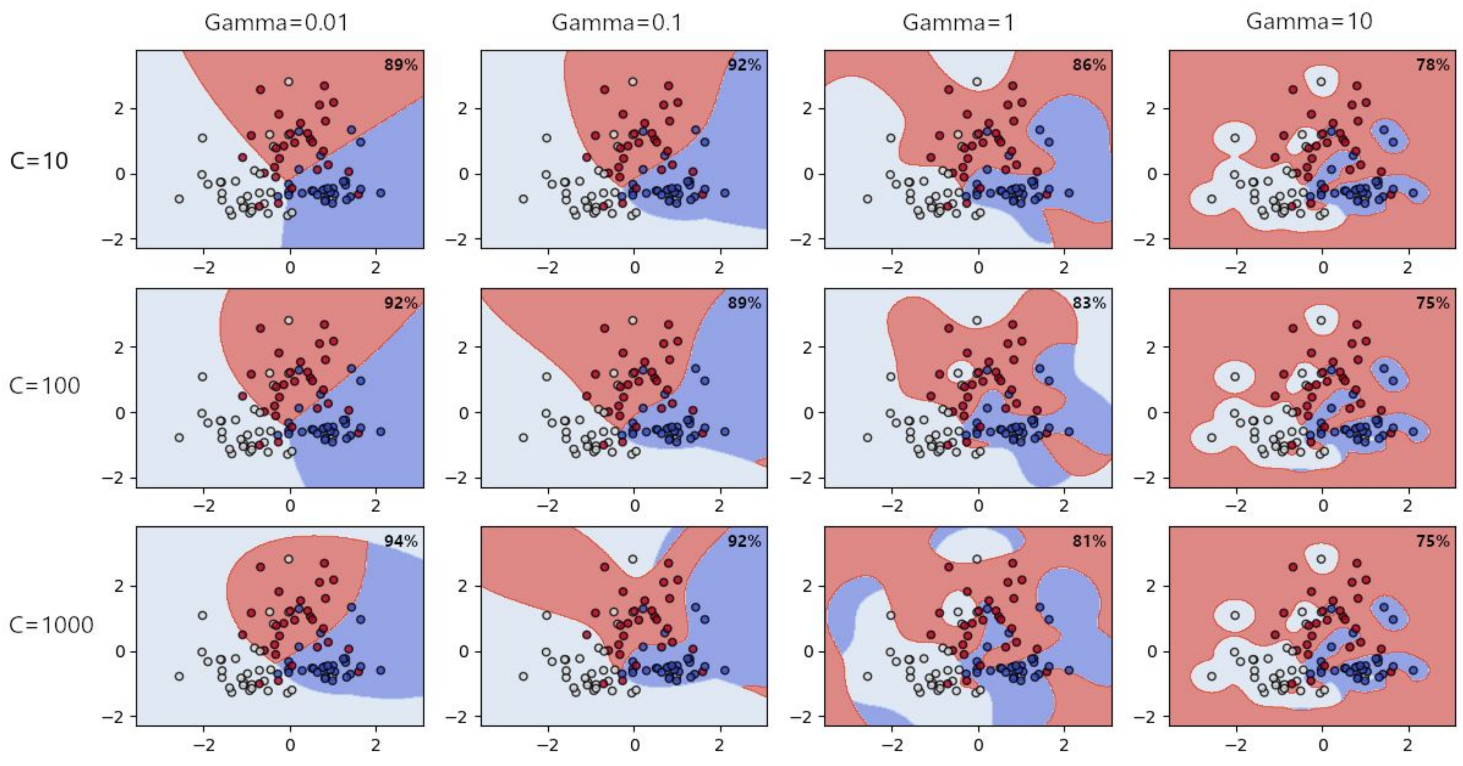
Tuning gamma allows for changing the radius of the RBF Kernel, modifying how the similarities between points are calculated. A relatively small gamma leads to all support vectors affecting the decision function, while a considerably high value of gamma makes only “close” support vectors to be important for the resulting decision function.

Note that also gamma acts as a regularization parameter, in a way ([2]).

The following hyperparameters have been tried, during the tuning phase on the validation set:

C: [**0.1**, **1**, **10**, **100**, **1000**]
Gamma: [**0.01**, **0.1**, **1.0**, **10.0**, **100.0**]

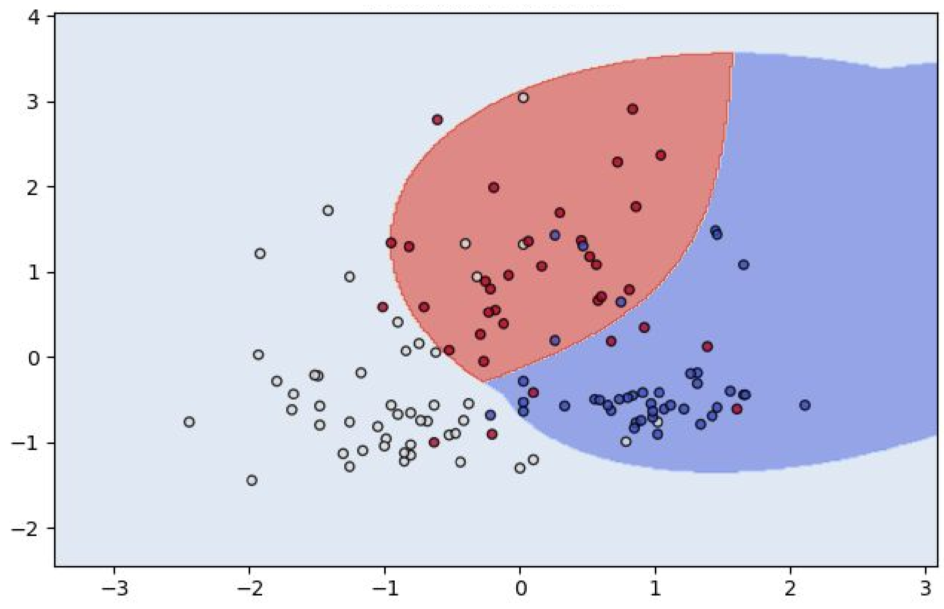
Some of the models created during the gridsearch are summarized in the following figures showing their corresponding decision boundaries:



Only the training set is shown in the above figures.

As shown, both the hyperparameters are affecting the capability of the model to fit our data or generalize. In particular, really high values of gamma lead to high overfitting as only the closest support vector has weight on the decision function.

- The gridsearch gave the following best configuration: **{C=1000, Gamma=0.01}**
- Accuracy on the validation set: **94%**
- Final evaluation of this model on the test set (with training+validation): **83%**
- Decision boundaries of the final model (training+validation):

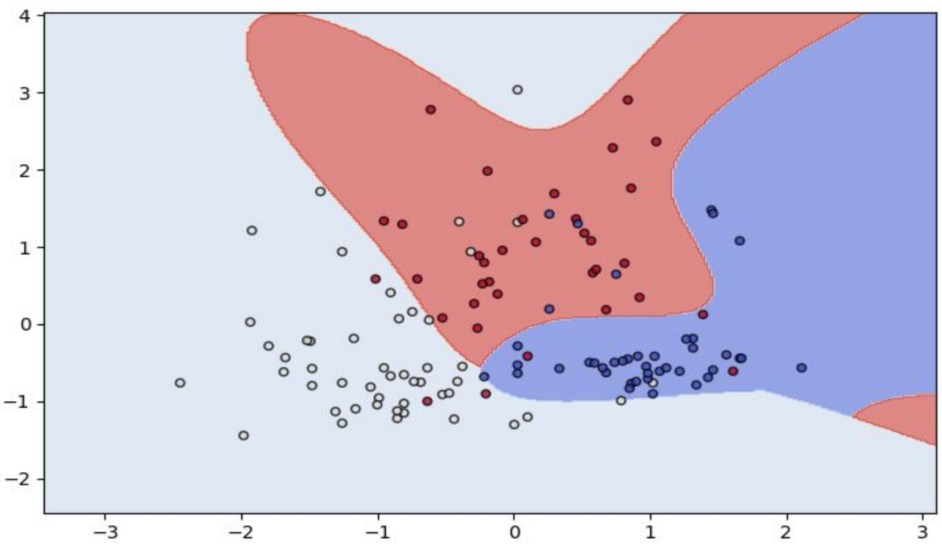


Both the training set and the validation set are shown in the above figure.

The model performed slightly worse with respect to when gamma was automatically set by the classifier. This might be due to the fact of doing a gridsearch on such a small dataset with a holdout technique that can lead to overfitting the hyperparameters on the validation set. As a result, the model probably did not learn how to generalize best from the given data.

To try overcome for this, a cross-validation technique has been tried for the hyperparameter tuning of C and gamma together:

- Number of folds in cross-validation: **5**
- Best configuration found: **{C=1000, gamma=0.1}**
- Average accuracy on the 5-folds during cross-validation: **80%**
- Final evaluation of the model on the test set (with the best configuration): **83%**
- Decision boundaries of the final model (with the best configuration):



Both the training set and the validation set are shown in the above figure.

The best configuration found was just slightly different as before, but lead to a considerably different model by looking at the decision boundaries. Unfortunately though, no improvement has been shown on our final model on the test set. This could be due to the fact of having still a low number of samples in our training set for the K-fold gridsearch. However, increasing the number of folds (or even a "leave-one-out" technique) might increase performances for small datasets as more data is used to train the classifier at each fold.
Indeed, the model does behave differently if 20 folds are used during the gridsearch:

- Number of folds in cross-validation: **20** *(5% of samples are left out at each fold)*
- Best configuration found: **{C=1, gamma=1.0}**
- Average accuracy on the 20-folds during cross-validation: **81%**
- Final evaluation of the model on the test set (with the best configuration): **85%**

When using 20 folds, the model is able to generalize better, obtaining higher accuracies on both sets.

5. Extra

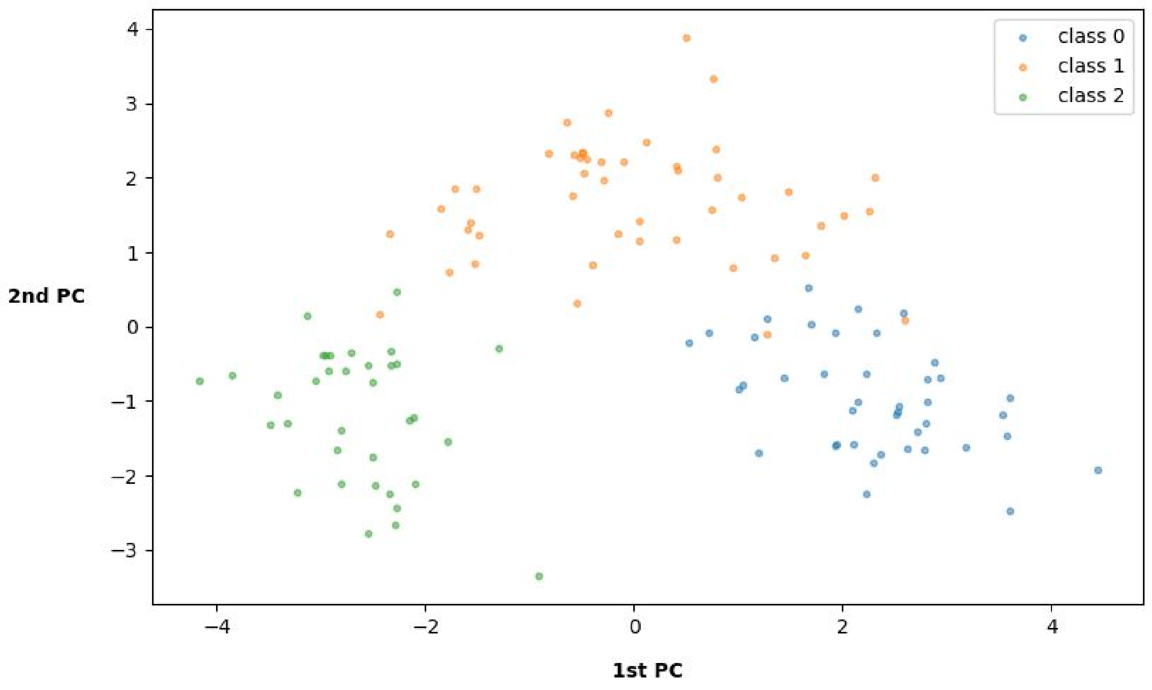
When compared, KNN and SVM models behave very differently in terms of decision boundaries and classification functions. The K-nearest neighbor classifier is more naive, as it does not rely on the solution of an optimization problem (non-parametric model) and it is very sensitive to the local structure of the data. In addition, we can only do so much by tuning the hyperparameter "K" and no other regularization techniques are easily accessible.
SVM models generally allow for better hypertuning, by acting on different hyperparameters in order to select the model that works best for our given dataset. Furthermore, they allow the use of the Kernel trick to learn non-linear decision boundaries and further adapt to the training set distribution.

Even though the results were not stable due to the random splits and a low number of samples in the wine dataset, the SVM's model performed slightly better w.r.t. the KNN model in our case.

Finally, different pairs of attributes can be chosen and the entire analysis can be performed from scratch. Hopefully, new trends and underlying data patterns will be captured in order to increase the efficiency of our classification task.

A dimensionality reduction technique can also be applied to more effectively select a new pair of features. For example, PCA can be performed and only the first two principal components kept for the analysis, or LDA can be applied resulting in 2 new features that best discriminate between the three class distributions.

If PCA is performed, the resulting dataset when only the first two principal components are kept is as follows:



Both the training set and the validation set are shown in the above figure.

It is immediately noticeable how these two new attributes better discriminate between the three labels, even through a linear separation.

The expectations were confirmed by the empirical results, where the new transformed dataset effectively outperformed by a considerable amount all the previously described models:

- **KNN:** 96% final accuracy on test set with best K=7
- **Linear SVM:** 96% final accuracy on test set with best C=1000
- **RBF-kernel SVM:** 94% final accuracy on test set with best C=100
- **RBF-kernel SVM with both gamma and C hypertuned:** 96% final accuracy on test set with best C=10 and best gamma=0.01
- **RBF-kernel SVM KFold:** 96% final accuracy on test set with best C=1000 and best gamma=0.001

References

[1] Scaling the regularization parameter in SVM's
https://scikit-learn.org/stable/auto_examples/svm/plot_svm_scale_c.html

[2] Gamma and C as regularization hyperparameters
https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[3] Support vector machines (SVM's) for classification
https://scikit-learn.org/stable/modules/svm.html

[4] K-Nearest Neighbor model for classification
https://scikit-learn.org/stable/modules/neighbors.html