Machine Learning and Deep Learning
01TXFSM
Politecnico di Torino

Homework 1

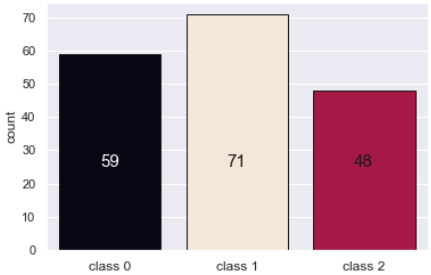
Riccardo Mereu Student ID: s277004

1 Introduction

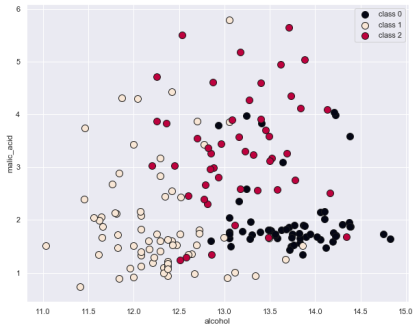
The goal of this homework is to apply on the UCI wine Dataset different types of classifiers and investigate how those different algorithms perform and how changing the validation procedure affects the hyperparameter tuning and the results.

1.1 Dataset

The wine dataset is a classic multi-class classification dataset. It is possible to load it using the `sklearn.datasets.load_wine` function. The data in this dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The dataset consists of 178 samples with 13 features each. Figure 1.a reports the distribution of the samples over the three classes which represent the different cultivars.
For this homework, we will focus on the first two features of the dataset '`alcohol`' and '`malic_acid`'. The pair-wise scatter plot obtained using these two attributes is reported in Figure 1.b.



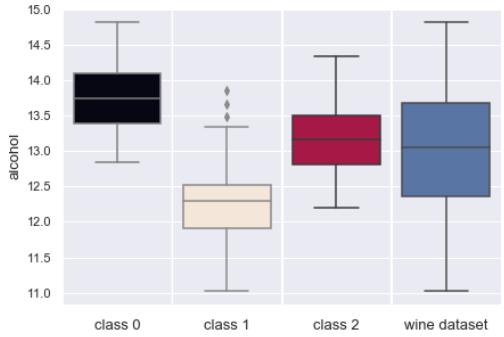
(a) Label distribution



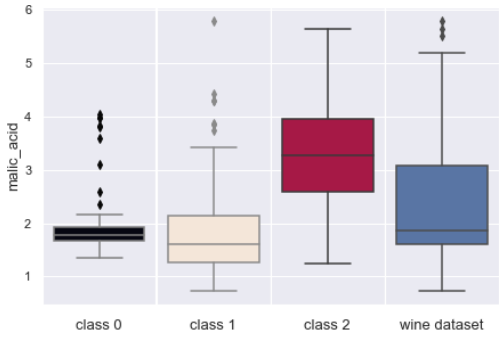
(b) Pair-wise plot 'alcohol' vs. 'malic_acid'

Figure 1

Figure 2 contains the boxplots for each feature on the whole dataset and on the single classes. The '`alcohol`' attribute has median 13.05, mean 13.00 and standard deviation of 0.81. The boxplot shows how 50% of the values are between 12.4 and 13.7. Observing the distribution class by class, the three classes seem to naturally separate as low/mid/high alcohol distribution.
The '`malic_acid`' attribute has median 1.87, mean 2.34 and standard deviation of 1.11. The boxplot shows how 50% of the values are between 1.6 and 3.1 for the whole dataset. Contrarily to the first attribute, here there is not a clear distinction between the classes. Class 2 is spread all over the attribute's domain and has 50% of the values between 2.6 and 4.0. On the other hand, class 0 IQR is between 0.3 (Q1=1.6 and Q3=1.9) and for class 1 is 0.8 (Q1=1.3 and Q3=2.1).
The different behaviour of the samples on these two attributes is clear in the pair plot (Figure 1.b). On the x-axis, the classes 0 and 1 are well separated, while class 2 sits between the others. On the y-axis, it is not possible to see a clear distinction between the first two classes (class 0 and 1). Class 2 is partially overlapped with class 1 but has the majority of the points over the other two classes.



(a) Boxplot for 'alcohol'



(b) Boxplot for 'malic_acid'

Figure 2

1.2 Splitting data into train, validation and test set

For the first part of the homework, it is necessary to split the dataset into training set, validation set and test set. To obtain this result I have implemented a function `train_validation_test_split()` which exploits `sklearn.model_selection.train_test_split`. My function first divides the dataset in two. One of the chunks will become the test set, while the other chunk will be further divided into training set and validation set. The function takes the same parameters as the sklearn function plus a `val_size` that is the proportion of dataset used as the validation set. To give the possibility to make the experiments repeatable is it possible to set a random seed and generate the same partitions of the dataset for different runs.

1.3 Data Standardization

The two selected features have different scales. Then it becomes important to standardize the data, in this way the features contribute equally to the analysis and avoid creating a bias. Standardization transforms data to have a mean of zero and a standard deviation of one. The reason that the standard deviation is changed to one specifically is that the obtained z-score is a measure of how many standard deviations we are from the mean, regardless of the size of that deviation in the original data. For this reason, standardization of a dataset is a common requirement for many

machine learning estimators. For instance, many elements used in the objective function of learning algorithms implemented in sklearn assume that all features are centered around 0 and have variance in the same order. sklearn offers a `preprocessing` module that contains the `StandarScaler` class, that I used to standardize the data before proceeding with the analysis.

2 Algorithms

The algorithms considered in this homework are K-Nearest Neighbors, Linear SVM and Kernelized SVM, in particular, the RBF SVM. To evaluate the performance of these algorithms we tried different resampling methods.

2.1 k-Nearest Neighbors

The k-Nearest Neighbors is a non-parametric and instance-based classifier. Given a positive integer K and an observation $\mathbf{x}$, the KNN classifier first identifies the K points in the training set that are closest to x. Then it counts how many members of each class are in this neighbourhood and assigns by a majority vote the predicted label to $\mathbf{x}$.
The choice of K has a drastic effect on the KNN classifier obtained. Obviously, the k-NN classifier implementation of sklearn, `KNeighborsClassifier`, allows changing not only K but also different hyperparameters of the model. One of them is 'metric', that allows selecting how the distances between the examples in the dataset. The default one is the Minkowski metric with power 2, i.e. the standard Euclidean metric. It is interesting to consider the difference with other ones, so I tried the Mahalanobis and the ℓ_1 distances to see how this choice affects the boundaries of the classifier.

2.2 Support Vector Machines

Citing Bishop [1] and Cristianini et. al[6], Support Vector Machines (SVM) are decision machines, in the sense that are systems for efficiently training linear learning machines in the kernel-induced feature spaces. In the classification setting the aim is to find the best separating hyperplane that divides the training data exploiting generalization and optimization theory.
SVMs approach this problem employing the concept of margin, which is defined to be the smallest distance between the decision boundary and any of the data points. Maximizing the margin leads to a particular choice of the decision boundary that can be expressed by a subset of the data points called support vectors. For this reason, SVMs are sparse kernel machines because the predictions for new inputs depend only on the kernel function evaluated at a subset of the training points.

The simplest model of SVM is the so-called maximal margin classifier. This algorithm works only for data which is linearly separable. In the binary case, it looks for the hyperplane that separates the input data in two distinct regions in the multidimensional space. This hyperplane is generated as a linear combination of the training examples and maximizes the maximum margin. Its associated optimisation problem, given a linearly separable training sample $S = \{(x^{(i)}, y^{(i)}); i \in [m]\}$, is

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle \\ \text{s.t.} \quad & y^{(i)}(\langle \mathbf{w}, \mathbf{x}^{(i)} \rangle + b) \geq 1, \forall i \in [m] \end{aligned}$$

The hyperplane $(\mathbf{w}, b)$ that solves this problem is the maximal margin hyperplane. Then, it is possible to classify a test observation based on which side of the maximal margin hyperplane it lies. This problem can be solved using a quadratic programming solver, but considering the corresponding dual

form that leads to two advantages: the possibility to use kernels and derive an efficient algorithm to solve the problem, the SMO algorithm. The latter, in a slightly different optimized version, is used in LIBSVM, the library used by sklearn to implement `sklearn.svm.SVC` and `sklearn.svm.SVR`.

2.2.1 From hard-margin to soft-margin classifiers

The assumption of the existence of a hyperplane that linearly separates the data is very strong but unfortunately nearly impossible in the real world. In many cases, there is no maximal margin classifier, however, it is possible to extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes. This new definition allows some observations to be misclassified, i.e. to be on the wrong side of the margin or even on the incorrect side of the hyperplane. This is the so-called soft-margin classifier as opposed to the previous hard-margin classifier. To achieve such a result it is necessary to reformulate the problem adding some slack variables. Using ℓ_1 regularization we obtain the following optimization problem, primal and dual formulation:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\langle \mathbf{w}, \mathbf{w}\rangle + \lambda \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y^{(i)}(\langle \mathbf{w}, \mathbf{x}^{(i)}\rangle + b) \geq 1 - \xi_i, \forall i \in [m] \\ & \xi_i \geq 0, \forall i \in [m] \\ & \textit{with } \lambda = 1/C \end{aligned} \qquad \begin{aligned} \max_{\mathbf{w},b,\xi,\alpha} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left(\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \right) \\ \text{s.t.} \quad & \sum_{i=1}^{m} y^{(i)} \alpha_i = 0 \\ & 0 \leq \alpha_i \leq \lambda, \forall i \in [m] \end{aligned}$$

Thus, examples are now permitted to have margin less than 1, and if an example has functional margin $1 - \xi_i$ (with $\xi_i > 0$), this will affect the objective function increasing it by $C\xi_i$. The hyperparameter $\lambda = 1/C$ controls the relative weighting between the two goals of minimizing $\|\mathbf{w}\|^2 = \langle \mathbf{w}, \mathbf{w}\rangle$, that is the same as maximizing the margin, and ensuring that most examples have functional margin at least 1.

2.2.2 From binary SVM to multi-class SVM

Until now, the discussion was about binary classification but we can upgrade the SVM to the multi-class case. There are two main approaches for this upgrade: one-versus-the-rest approach also called one-vs-all (OVR or OVA) and the one-versus-one approach (OVO).
In the OVR approach, N binary classifiers are trained, where the data of n class is treated as positive examples and the rest is treated as negative. This approach is the simplest one, it can lead to inputs space regions that are ambiguously labelled and suffers from the class imbalance problem. The OVO approach consists of training $C(C-1)/2$ binary classifiers to discriminate all pairs of classes n versus n'. The data point is then classified into the class that has the largest number of votes. This approach is more computationally expensive than OVR and it may lead to ambiguous regions. In both cases, to overcome this ambiguity different heuristics are used to determine how to classify the points that belong to these inputs regions.

2.2.3 Linear SVM and RBF SVM

Even with the use of a soft-margin classifier, there are problems where the unknown function that labels the input vectors is not linear and then the classifier will perform poorly. The dual form of the optimization problem is written entirely in terms of the inner products $\langle \mathbf{x}, \mathbf{x}' \rangle$. This means it is

possible to map the input attributes of the problem, in this case $\mathbf{x}$ and $\mathbf{x}'$, in a new feature space where may be possible to find a hyperplane of this new space that separates correctly the inputs.

We can replace the inner products with $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$. Since the mapping may be computationally expensive, especially considering infinite-dimensional feature spaces, it is possible to define the corresponding kernel to be:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

The kernel function $k(\mathbf{x}, \mathbf{x}')$ may be very inexpensive to calculate and that gets the SVMs to learn in the high dimensional feature space given by ϕ without the computational burden to explicitly calculate $\phi(\mathbf{x})$.

The basic scenario addressed in the definition of the previous section can be seen as the scenario where the feature space is the same as the attribute space, i.e. $\phi(\mathbf{x}) = \mathbf{x}$. Then the corresponding kernel is the linear kernel, $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$.
A more interesting kernel is the Gaussian radial basis function kernel (RBF kernel)[4][5], which implicitly maps the input vectors into an infinite-dimensional feature space. It is a particular case of the Gaussian kernel defined as

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Sigma^{-1} (\mathbf{x} - \mathbf{x}'))\right)$$

If Σ is a spherical matrix (i.e. a multiple of the identity matrix), we get the isotropic kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

where $\gamma = \frac{1}{2\sigma^2}$ is a parameter. Since this kernel is a decreasing function of the Euclidean distance between points,it can be interpreted as a measure of similarity: the larger $k(\mathbf{x}, \mathbf{x}')$, the closer the points $\mathbf{x}$ and $\mathbf{x}'$ in the feature space.

3 Experiments and results

In the first part of the homework, the goal was to use the validation set approach to select the best hyperparameters for each model fitted. Then, the models were tested on the test set to assess their performance.
The following goal consisted of tuning the RBF SVM using a grid search over two hyperparameters, C and γ, and then to evaluate the new fitted models using both validation set method and k-fold cross-validation.
The results described in the following paragraphs were obtained fixing the splits of the dataset for replicability of the results. To do this the `random_state` parameter of the `train_validation_test_split` function is set to 42. However, later I will address how this different random splits of the dataset affect the results of these approaches.

3.1 Validation Set approach

The first part of the experience consisted of applying the validation set approach to the algorithms mentioned in Section 2. The dataset is split randomly in three subsets training set, validation set and test set, with ratio 5:2:3. Each classifier is trained on the training set multiple times with different hyperparameter's values. Then, the obtained models are evaluated over the validation set. The model that achieves the best accuracy score (other scores may be used) is then fitted on the set composed of training and validation subsets. In the end, this model is tested on the test set.

3.1.1 k-Nearest Neighbors

The hyperparameter under analysis for the kNN algorithm is K, the number of nearest neighbors that must be taken into account when labeling a new example. The values suggested for this homework are 1, 3, 5 and 7. These are all odd values because, in that way, the algorithm avoids getting ties during the label voting. The metric used for the training is the sklearn `KNeighborClassifier`'s default metric, i.e. the Euclidean distance.
The decision boundaries obtained training the classifier with different K values are shown in Figure 3.

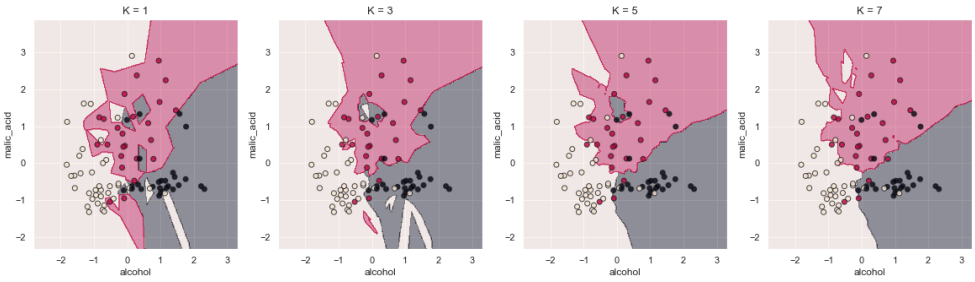
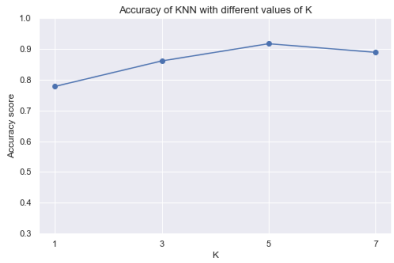


Figure 3: Decision boundaries of the kNN classifiers with different values of K. The points in the plot are from the training set.

Inspecting Figure 3, it is possible to see that the decision boundaries ended up containing islands of labels formed because of noisy examples. In particular, we can observe this behaviour for smaller values of K. In fact, by taking more neighbors, the impact of outliers can be reduced. Larger K produces smoother boundaries because the impact of class label noises cancelled out each other. On the other hand that may lead to overly simplified models (i.e. underfitting). With smaller values of K, the training error reduces, for instance with K=1 we always reach training error equals to 0. However, this may lead to overfitting because the model is not able to generalize well. Summarising, when K increases the training error will increase (increased bias), but the test error may decrease at the same time (decrease variance). The higher the k, the lower the model complexity, and the estimation becomes more global, space partitioned into larger patches.



	K value	Accuracy	Precision	f-score
Validation set	1	0.778	0.775	0.774
	3	0.861	0.869	0.859
	5	0.917	0.932	0.918
	7	0.889	0.891	0.887
Test set	5	0.759	0.783	0.763

Figure 4 and Table 1

The scores achieved are shown in Figure 4 and Table 1. The performance on the validation set seems to be poorer with smaller values of K and, after achieving the best result for K=5, it decreases

again for larger values.
We have to say that these results are biased by the current split of the data. This is clear in Table 4, where we can see that the best model for the validation set performs poorly on the test set, with an accuracy of only 0.759, which is worse than the result achieved by the worst model on the validation set. A different value of K might lead to a better generalization but since we are choosing the hyperparameter before seeing the test cases, that was impossible to know and not good because in practice the future data on which the model will work are unknown either.

As stated in section 2.3, the Euclidean distance is not the only choice available. Choosing different metrics changes the shape of the decision boundaries. Figure 5 shows the decision boundaries obtained with the Euclidean, the ℓ_1 and Mahalanobis distance with K=3. The first and the third one have essentially the same shape, but the ℓ_1 distance leads to sharper boundaries.

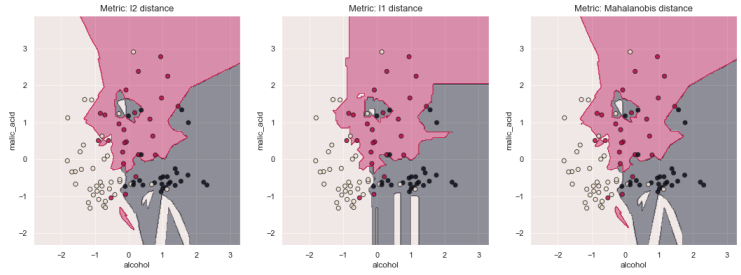


Figure 5: Decision boundaries obtained with K=3 and different metrics

3.1.2 Linear SVM

For the linear kernel SVM, the tuning hyperparameter considered is C, that is the regularization constant addressed in section 2.2.1. This parameter controls the trade-off between the slack variable penalty and the margin. If C goes to ∞, then λ goes to 0 and we obtain the large margin classifier described in section 2.2. This means that for each $i \in [m]$ $\xi_i = 0$, this is an extreme case and a maximal margin hyperplane exists only if the classes are linearly separable. As C decreases, we have $\lambda > 0$, then because any point that is misclassified has $\xi_i > 1$, it follows that $\sum_i \xi_i$ is an upper bound on the number of misclassified points. The parameter C is therefore analogous to a regularization coefficient because it controls the trade-off between minimizing training errors and controlling model complexity. The values suggested for C are 0.001, 0.01, 0.1, 1, 10, 100, 1000, that means allowing progressively less misclassified data points.

sklearn has two different implementations of the SVM classifier `svm.SVC` and `svm.LinearSVC`. The following results were obtained using SVC and later I will address the differences between these implementations.

Figure 6 shows the decision boundaries obtained fitting SVC on the training set with different values of C.
The first decision boundary, obtained with C equal to 0.001, allows many misclassified examples and as a consequence a large margin. Basically, all the points reside within the margin and are all classified with the label 1 that recalling Figure 1.a is the most numerous class among the dataset and event in the training set. Therefore, the SVM is trying to minimize the loss assigning to all the samples the class with guarantees the less number of misclassified samples, i.e. the largest one.
The plot for C equal to 0.01 can split class 1 and class 0, but the margin is again too large to distinguish even class 2.

For values equal to or greater than 0.1, there is no substantial difference in the decision boundaries. The difference between these models resides in the dimension of the margin as shown in Figure 7.

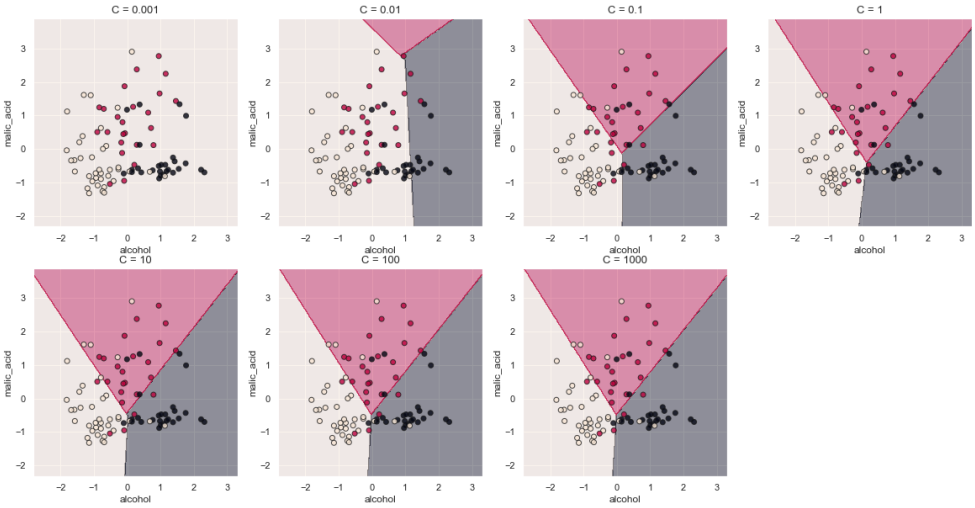


Figure 6: Decision boundaries of the linear kernel SVM obtained using different values of C

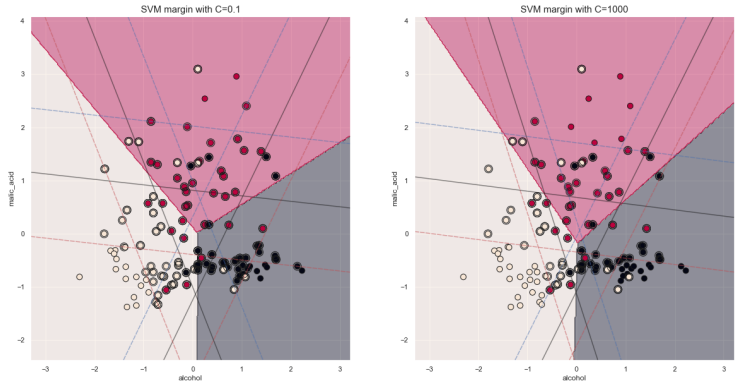
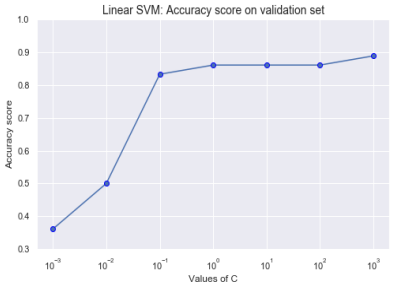


Figure 7: Decision boundaries and margins obtained with C=0.1 (on the left) and C=1000 (on the right)

Following the validation set approach, after fitting the models on the training set, they are evaluated on the validation set. The best one is then fitted again and evaluated on the test set. The obtained results are reported in Table 2 and Figure 8. The best model on the validation set is the one with C equal to 1000, which have an accuracy score of 0.891 on the validation set and, as happened for the kNN, a lower accuracy score on the test set of 0.741.



	C value	Accuracy	Precision	f-score
Validation set	10^{-3}	0.361	0.130	-
	10^{-2}	0.500	0.420	0.404
	10^{-1}	0.833	0.836	0.834
	1	0.889	0.863	0.861
	10^{1}	0.861	0.867	0.858
	10^{2}	0.861	0.867	0.858
	10^{3}	0.889	0.889	0.887
Test set	10^{3}	0.741	0.750	0.741

Figure 8 and Table 2: Results achieved by the linear kernel SVMs both on validation and test set

`SVC` uses as default value for the ' `decision\_function\_shape` parameter `ovr`, that means that for the multi-class SVM the One-Vs-Rest approach. To create the decision boundaries of the model for C=1000, the algorithm trains 3 models and merges their result as shown in Figure 9.

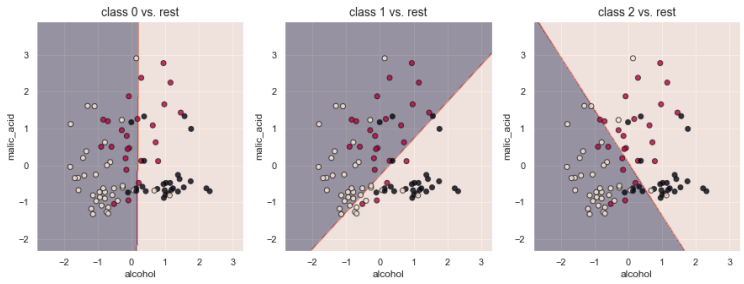


Figure 9: Decision boundaries of the 3 SVMs trained with OVR apporach with C=0.1

As mentioned above, sklearn offers two different implementations of SVM classifiers, `LinearSVC` and `SVC`. The difference lies in the libraries that actually implement the algorithm. In this homework I used `SVC`, we can observe completely different results with `LinearSVC`. This implementation relies on `liblinear`, which uses SGD to find the solution to the SVM's minimization problem. The main difference resides is due to the fact that `LinearSVC` regularizes not only the weights but also the bias [7]. In the notebook are plotted also the decison boundaries obtained using `LinearSVC` with '`hinge`' loss, which is the same used by `SVC`. Their shape is considerably different in particular for lower C, where they converge to a better accuracy but on the other hand need a higher number of iteration than the default one.

3.1.3 RBF SVM

The use of a non-linear kernel changes completely the shape of the decision boundaries. The hyperplanes split the data points in the features space and then to plot them, they are projected again onto the attribute space. Their shape is completely different from the linear kernel SVM.
The decision boundaries of the fitted models are shown in Figure x, note that only the hyperparameter C is tuned. The scores on the validation set are presented in Table 3, as we can see the best result is 0.861 obtained for C=$10^{-1}, 1, 10, 10^{2}$. Like for the linear kernel, for lower values of C the solution is determined entirely by the regularization term so that the bias is very high and variance very low. For this reason, in the first two plots, all data points are labeled to class 1. The

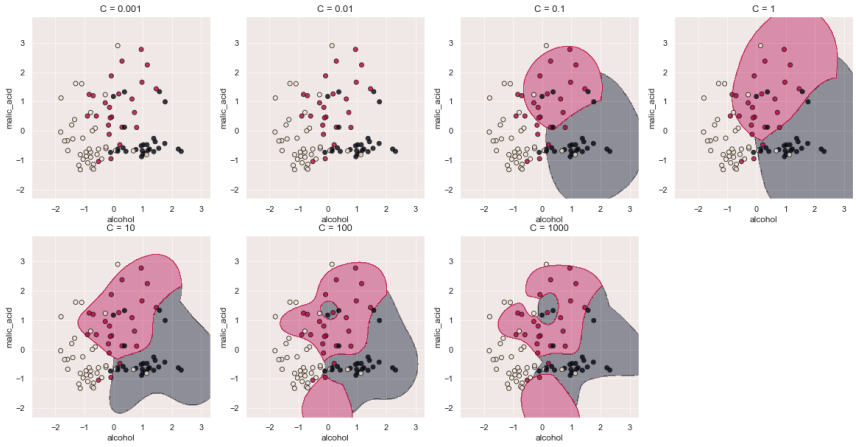
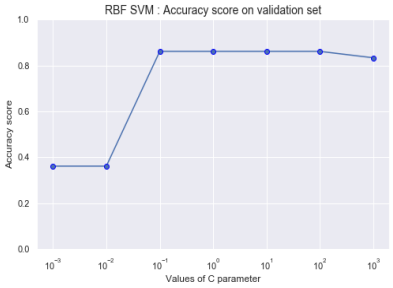


Figure 10: Decision boundaries of the kNN classifier with different values of K. The points in the plot are from the training set.

decision boundaries for C equal to 0.1, 1 and 10 are smooth and regular, that means that the models have a good bias-variance trade-off. This is also confirmed by the results on the validation set. But continuing to increase C also the variance increases and the model starts to overfit the training set, as a consequence the accuracy score on the validation decreases. Looking at the decision boundaries, I've decided to use as the best model the one with C=10, this decision is motivated by two reasons choosing the model that enforces the lower number of errors, so with higher C values, but not too high to prevent overfitting on the training set. The latter becomes evident with C=100 and C=1000, where the decision boundaries start forming islands of points into larger regions. The best model obtained an accuracy of 0.852 on the test set, which is a little lower than the results obtained in the validation set, but the gap is smaller than the one obtained with both kNN and linear SVM.



	C value	Accuracy	Precision	f-score
Validation set	10^{-3}	0.361	0.130	-
	10^{-2}	0.361	0.130	-
	10^{-1}	0.861	0.862	0.857
	1	0.861	0.862	0.857
	10^{1}	0.861	0.862	0.857
	10^{2}	0.861	0.877	0.858
	10^{3}	0.833	0.859	0.827
Test set	10^{3}	0.852	0.860	0.855

Figure 11 and Table 3: Results obtained by RBF kernel SVM on both validation and test set

3.1.4 Drawbacks of Validation Set approach

To obtain the results of the previous paragraphs, the dataset was randomly divided into three parts, a training set, a validation set and a test set. If the '`random_state`' is not fixed, then repeating the process of randomly splitting the dataset, the estimates of the accuracy score will be different. Figure 12 shows how different runs with different random split generates variable results. For this reason, the results presented before are biased by the particular split of the data chosen before. This approach is very simple and easy to implement but has two main drawbacks. First, the validation estimate of the test error, i.e. the accuracy score on the validation set, can be highly variable, depending on which observations are included in the training set and the validation set. Second, only a subset of the observations are used to fit the model, then our model will perform worse than in the case where we fit it with all the data at our disposal.

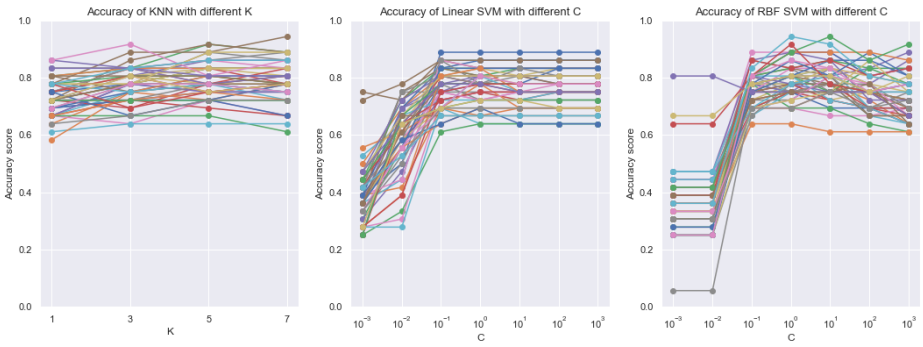


Figure 12: Accuracy of 50 runs of each model using validation set approach

3.2 Grid Search Model Tuning

3.2.1 The γ hyperparameter

Before talking about the grid search model tuning, I will focus on the hyperparameter γ. A kernel expresses a measure of similarity between vectors. The RBF represents this similaruìity as a decaying function of the distance between the vectors, the squared-norm of their distance. The discriminant function learned by an SVM using this kernel has the form

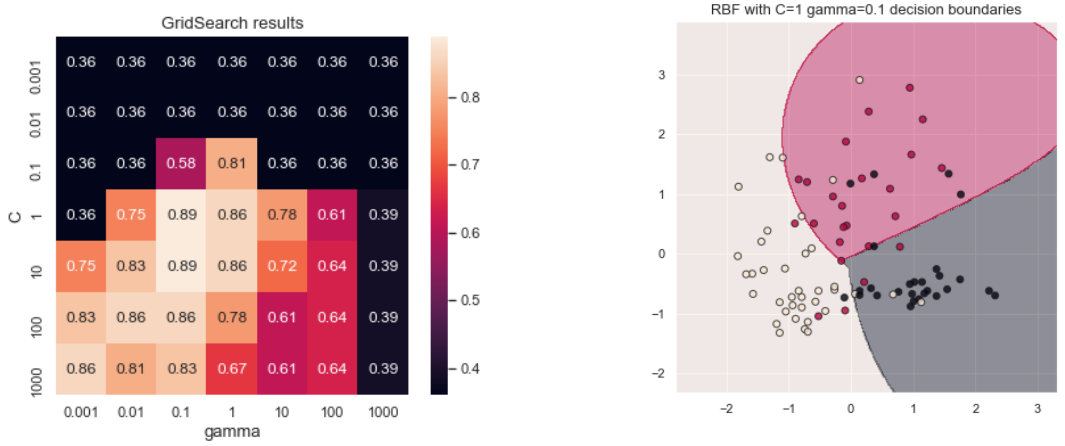
$$f(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}\|^2\right)$$

and therefore is a linear combination of Gaussian centered on the support vectors.
Using this kernel almost any decision boundary can be obtained. The larger the value of γ, the more peaked the Gaussian are around the support vectors allowing to create more complex decision boundaries. This peaked curves can lead to the creation of islands of decision boundaries around data points. On the other hand, smaller γ corresponds to a smoother decision boundary.
As suggested by the authors of LIBSVM in [3], trying exponentially growing sequences of C and γ is a practical method to identify good parameters. For this reason I decided to use the same values used for C even for γ.

3.2.2 Grid Search

The grid search approach is a method to tune the model hyperparameters. It is an exhaustive trial and error approach in which a discrete grid with all possible combination of hyperparameters value is created. After that, the algorithm fits a model for each possible pair and evaluates its performance. The combination of hyperparameters in the grid that performs better is chosen to train the final model on all the data. The performance evaluation can be done or by the validation set approach or by cross-validation.
In this section, the former approach is used. sklearn provides only the class `GridSearchCV`, which performs the grid search assessing the model through cross-validation. To perform a grid search that fits the models on a training set and validate their performance on a validation set I modified `GridSearchCV` creating the class `GridSearch`. This class has the same methods of the sklearn's one, but for the `fit()` method takes as input data not only `X` and `y` but `X_train`, `y_train`, `X_validation` and `y_validation`. The internal mechanisms are the same as the `GridSeachCV` without the cross-validation part, which is replaced with a simpler validation set approach.

Figure 13.a contains the result of the grid search over the hyperparmeters using `GridSearch`, there is a tie between $(C = 10, \gamma = 0.1)$ and $(C = 1, \gamma = 0.1)$ both with accuracy score of 0.89 on the validation set. As for `GridSearchCV`, the class `GridSearch` takes as the best hyperparameter the `argmin` of the dictionary of the results. Then in case of a tie like this one, the choice is arbitrary. Since this approach is already biased by the choice of the split, I have decided to stick with the values returned by the `GridSearch` instance. The accuracy score on the test set of this model is lower than the accuracy achieved in the validation set and is equal to 0.796.



(a) Average accuracy score of the models trained with GridSearch

(b) Decision boundary of the best model obtained with GridSearch

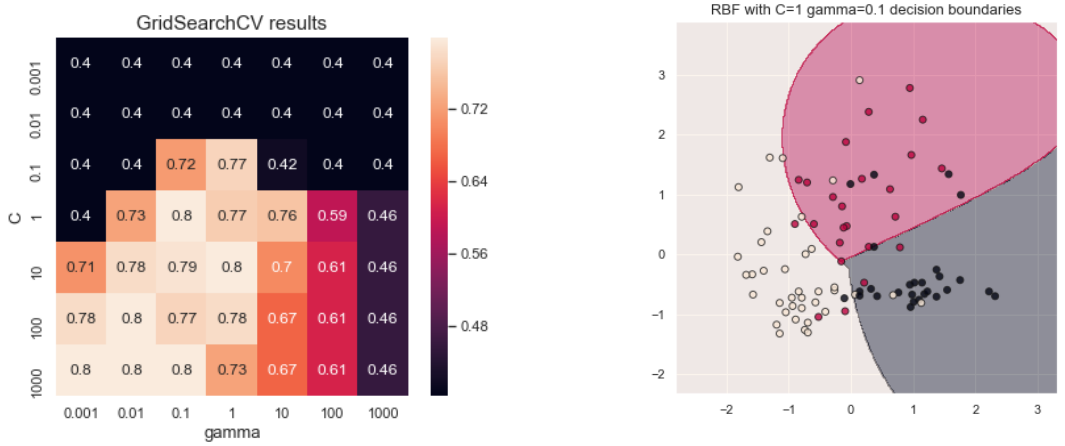
Figure 13

3.3 k-Fold Cross-Validation approach

In this section are discussed the results obtained tuning the RBF SVM model with a grid search cross-validation. The k-fold cross-validation approach consists of randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and

the method is fit on the remaining $k-1$ folds. The score of the model is then computed on the observations in the held-out fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. The score estimate is computed by averaging the values obtained at each iteration. sklearn's `GridSearchCV`, together with sklearn `KFold`, can be used to perform a grid search on the hyperparameters C and γ estimating the accuracy score with the k-fold method. Figure 14.a contains the results of this search. The estimator that performed better on validation set had parameters (C=1, γ=0.1) with accuracy 0.80 ± 0.05, where 0.05 is the sample standard deviation over the 5 folds. To guarantee replicability of the result even in this case the `KFold` had `random_state` fixed to 42.

The score obtained on the test set is 0.778, contrary to the results obtained in the previous section, this time the difference between the results of the grid search and on the test set is smaller. This is because the previous approach tends to overestimate the accuracy since it always looks at the same data, while with the k-fold cross-validation the algorithm leads to a less biased estimate.



(a) Average accuracy score of the models trained with GridSearchCV

(b) Decision boundary of the best model obtained with GridSearchCV

Figure 14

The k-fold cross-validation approach is a good compromise between the validation set approach inspected in the previous section and the Leave One Out Cross Validation approach. It avoids the computational burden of LOOCV but, at the same time, it could lead to a better bias-variance tradeoff.

The latter point could be evaluated examining the results of this homework. The validation set approach can lead to overestimates of the test error rate since with this approach the training set contains only half of the observations of the entire data (at least with the ratio used in this case). The accuracy achieved with the validation set approach is always higher than the test accuracy for this reason.

On the other hand, the LOOCV will give approximately unbiased estimates of the test error, since each training set covers almost the entire data available for training. The best bias reduction could be achieved with it but LOOCV has higher variance than the k-fold CV. This is because the LOOCV approach consists of averaging the outputs of n fitted models each of which trained on an almost identical set of samples. The outputs are then highly correlated with each other. In contrast, using

k-fold CV we are averaging the outputs of k fitted models that are somewhat less correlated since the overlap between the training sets in each model is smaller. The test error estimate resulting from LOOCV tends to have a higher variance than does the test error estimate resulting from k-fold CV.
Another further improvement to the approach used in the HW could have been to use a higher number of folds. For instance, using 10 folds we train the model ten times each time using 90% of the training data and the remaining 10% for the validation part, as a result, this leads to a less biased estimation of the test error rate.

4 Extra points

4.1 Point 19: difference between KNN and SVM

Section 2 describes the main characteristics of the two classifiers. A few remarks on their difference may be that the k-NN is a simpler model compared to the SVM. This one is considerably more complex since it can cover a wider range of possible decision boundaries, changing the values of C we can have a large or soft margin classifier, but this is not the only possible hyperparameter to tune. Using different kernels the possible decision functions of the model are even broader. SVM considers much more information when classifying a target instance and for this reason, they usually tend to have better accuracy than the k-NN. Since the k-NN has to keep in memory all the examples of the training set it is considerably slower than the SVM in the validation and test phase.

4.2 Point 20: other features

Point 20 required to use other attributes to train the model, I have decided to reduce the dimensionality of the dataset using sklearn PCA and take the scores on the two principal components. The results of this analysis are presented in a separate notebook (HW1_pt2). Since the data is well separated compared to the first two attributes, the classifiers achieved higher accuracy scores.

	Algorithm	**Hyperparameters**	**Training Accuracy**	**Test Accuracy**
Validation Set Approach	k-NN	K = 7	0.972	0.981
	SVM linear kernel	C = 1000	0.972	0.981
	SVM RBF kernel	C = 1 gamma = 'scale'	0.972	0.981
Grid Search	SVM RBF kernel	C = 0.1 gamma = 1	0.972	0.981
k-fold + GridSearch	SVM RBF kernel	C = 1 gamma = 0.1	0.968	0.981

Table 4

Table 4 contains the accuracy scores obtained by the best models obtained on the these new features. All the models reached an accuracy of 0.981 on the test set, which is a great improvement with respect to the results obtained with the first two features of the dataset.

5 Conclusions

In this homework, we focused on k-NN and SVM classifiers, different algorithms with different advantages and disadvantages. To train these models different approaches have been considered, the validation set approach, grid search and k-fold cross-validation. In that way, we examined many aspects of model tuning and model assessment.

References

[1] Christopher M. Bishop. Pattern Recognition and Machine Learning.

[2] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Leraning . URL: https://stackoverflow.com/q/33843981/7739581.

[3] Hsu, Chih-Wei et. al. A Practical Guide to Support Vector Classification. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

[4] B. Schölkopf J. Vert, K. Tsuda. A primer on kernel methods. URL: http://members.cbio.mines-paristech.fr/~jvert/publi/04kmcbbook/kernelprimer.pdf.

[5] Kevin P. Murphy. Machine Learning: A Probabilistic Perspective.

[6] Nello Cristianini, John Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning.

[7] Post on StackOverflow: "Under what parameters are SVC and LinearSVC in scikit-learn equivalent?". URL: https://stackoverflow.com/q/33843981/7739581.