Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA CIVILE EDILE E AMBIENTALE
Corso di Laurea Magistrale in Mathematical Engineering
Financial Engineering[1], Mathematical Modelling For Engineering And Sciences[2]

Scientific Computing And Object Oriented Programming

Final Project
Data Analysis Software Development

Studenti:
Vittoria Lazzarotto[1]
Matricola 1180128

Matteo Zambra[2]
Matricola 1140056

Docenti:
Emanuele Di Buccio, Ph.D.
Michele Schimd, Ph.D.

Anno Accademico 2017/2018

Page intentionally left blank

Contents

1 Problem

The data set chosen (see [**2**]) yields the problem to classify red wine quality on the basis of its chemical and physical properties (cfr. [**1**]). The Data Mining task that this kind of data represent is to learn the target function

$$f : X \longrightarrow Y$$

which maps the array $\mathbf{x} \in X \subset \mathbb{R}^d$ of $d = 11$ numerical values characterising a wine sample (in [**2**] an explanation of these is provided) in a category label $y \in Y = \{0,1,...,10\}$, which classifies that sample. Here 0 means a poor quality exemplar and 10 stands for an excellent product.

From a real world problem point of view, as pointed out in [**1**], classification of wine quality is a growing importance procedure in the certification step, as well as a warranty of safety for both human health and market necessities (the authors also cite the importance of good quality wine, since it is a non negligible amount of the Portuguese production, namely 15%). Thus the importance of predicting the product quality on the basis of easily analytical properties measurement, such as acidity, sulphates, pH and other physical and chemical features. These features are the numerical attributes $\mathbf{x}_i$ mentioned above.

To assess correctness and robustness of the code written and the algorithm used, two additional data sets are used, one regarding biomechanical features recorded from patients and the other one is the Fisher Iris data set ([**3**] and [**4**]). See Section 5 and 6 for a more detailed discussion.

2 Data Representation

The data available come as a $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ collection of $N = 1599$ records of 11 dimensional $\mathbf{x}$ vectors, y_i being the label.

A fundamental aspect of the present discussion is the way the task described in the previous Section is to be implemented. The C# Language (see Section 8 for further details about technicalities) is used, which yields the OOP-ish ease of abstraction of real world data (that is, even their behaviour). To this end the data collection has thought to be easily represented by an `DataObject` Object, designed in first instance as a wrapper for the properties

- `ItemsFeatures`, a $N \times d$ (being $N = 1599$ and $d = 11$) real matrix, storing the continuous data features. In the code, it has been modelled as a jagged array `double[N][]`, for consistency with learning algorithms later used;
- `CatIDs` $\in Y = \{0,1,...,10\}^N$ an integer vector `int[N]` containing the categories identifyiers;
- Since raw data were available as a `.csv` sheet, and I/O streams are managed via classes (built-in and from external Frameworks, See Sections 4 and 8) which read data as `string` types, the categorical labels are also stored as strings. Furthermore, the trial data sets categorical labels are indeed strings, but are eventually stored in the `int []` array of integer valued labels. This is done for all data sets.
- A `Dictionary<int,string>` to contain the feature string description, associated with the numerical label.

This `DataObject` class has no methods but the constructor, fed with dimensions of the real matrix and the labels vectors. This choice is motivated by the more clear representation of the data themselves, eventually using other classes to implement function to fill the data matrix.

The choice of a jagged array (that is, since all the entries in the data set are array with the same dimensionality, a matrix) is dictated by the natural representation of data as a real matrix. Furthermore in this ways operations of data preprocessing (such as feature-wise mean to compute the means vector to be subtracted to each row of the data matrix to perform the Principal Components Analysis – PCA, see later) are much more easy to implement. Integer values for the categorical labels are chosen to be consistent with the data as given and with the libraries used in the learning process, please see Section 3. In the to-integer translation the ordinal of the categories is not maintained, but it is irrelevant (it is shown in Section 6 that histograms of class distributions are consistent in both the ways).

An alternative could be a linked list, in which each node is a data object, i.e. an array storing the 11 features, but it could turn out ot be inefficient to random access entries, operation indeed performed while splitting the data set in training and test set (see Section 3).

3 Learning Models Deployed

More that one learning algorithm has been used. The reason for this is twofold: (1) assess correctness and (2) compare performance. The work in [**1**] has been used as a guideline, but is to be stressed that for that particular work other learning models have been utilised, but nevertheless, some affine results are here found, such as the accuracy of Support Vector Machines (SVMs) classifiers. Note also that in [**1**] SVMs prevail as classification techniques if compared with Regression and Neural Networks (NNs), while here SVMs are presented in comparison with the C4.5 Decision Trees algorithm (also here a good performance is detected).

For each algorithm used to train the classification model, data were provided to it in form of *training set* and *test set*, two proper subsets of the original data set, created in a way such that $\text{TrainSet} \cap \text{TestSet} = \emptyset$ and $\text{TrainSet} \cup \text{TestSet} = \text{DataSet}$. More precisely, two choices made are

$$\text{TestSet} = \frac{1}{10}\text{DataSet}, \quad \text{TrainSet} = \frac{9}{10}\text{DataSet}$$

$$\text{TestSet} = \frac{1}{3}\text{DataSet}, \quad \text{TrainSet} = \frac{2}{3}\text{DataSet}$$

The latter choice is adopted in [**1**]. Details on the partitioning given in Section 4.

3.1 Kernel Support Vector Machines

Kernel Support Vector Machines were first conceived as binary classificators, but eventually adapted for multiclass problems, as the present (a deeper analysis of the effective distribution of the class label upon the samples is reported in Sections 5 and 6).

Appeal of SVMs lay in two reasons. First, the optimization problem of finding the hyperplane parameters $(\mathbf{w},b) \in \mathbb{R}^N \times \mathbb{R}$ such that the margin between two different data categories disposed in the features space is maximised, yields a convex quadratic problem ([**5**]), and the solution is corresponding to the only minimum of the cost function that is

$$f(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2}$$

subjected to the constrain $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1$ with $i = 1,...N$ and being $\Phi(\mathbf{x})$ the transformation from the actual features space to the transformed one. This means that the solution is optimal. The second reason is that the transformation $\Phi(\mathbf{x})$ allows to transform possibly non linearly separable data in a way to render them such in an other features space (example given in [**5**], Chapter 5). The actual functional form of Φ is not relevant, since the *kernel trick* shall be used, substituting the dot product in the hyperplane equation with the action of the kernel on the vectors $\mathbf{x}$ and $\mathbf{w}$, i.e. $\mathbf{u} \cdot \mathbf{v} \to K(\mathbf{u},\mathbf{v})$, thus requiring only to choose the most suitable kernel $K(\cdot,\cdot)$. Gaussian and Polynomial kernels are here used. These ones' functional form is respectively

$$K(\mathbf{u},\mathbf{v}) = \exp\left(\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$$

$$K(\mathbf{u},\mathbf{v}) = (a\mathbf{u} \cdot \mathbf{v} + c)^d$$

Both these need parameters tuning, and in either way, depending on the data quality and quantity, such a choice is highly impacting on the performance in terms of error and execution time. The Polynomial degree has been kept fixed to 2, since the more parameters, the more variance and consequently the danger of **overfitting** the model ([**9**], Lecture 14, Pag. 12).

The second reason reported was assumed to be relevant, because since the data available are high dimensional, a preventive transformation in higher dimensions was supposed to help. On the other hand, the high dimensionality makes data analysing also by a visual perspective a cumbersome task, so PCA was used for dimensionality reduction, in order to get a view of the disposal of data in 2 dimensions. It eventually turns out that PCA-preprocessed data yield bad results, so 2 PCs analysis is rather used for visualising data (provided that data undergone processing, even bearing the same information, appear different; moreover, for visual clarity, only 2 PCs used means that scatter plot is feasible, but presumably thus doing important features of data disposition are neglected).

3.2 Decision Tree

As previously remarked, the choice of more than one classification technique helped to assess results, furthermore, having the feeling the data being not separable, choice of Decision Trees seems a good method for treating the problem not giving the separability a potential concern. The decision rules are derived on local basis, so oppositely to SVMs, local minima may be found so rendering the solution suboptimal. However, results yielded by Decision Tree algorithm proved good in the Wines data set, and reasonably satisfactory in the other two data sets analysed. Results are exposed in Section 5. Decision rules are derived on the basis of local *information gain* maximisation, namely the C4.5 learning algorithm sets the attribute to be used as rule by searching for which of these the quantity

$$\Delta I = I_{\text{parent}} - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

is maximum, being $I = -\sum_j p_j \ln(p_j)$ the information entropy metric (p_j is the fraction of the data of category j, rather than strictly a probability), k the number of attributes, $N(v_j)$ the number of records of child node v_j. This ΔI stands for the *entropy* (that is, in the context of information theory, informativeness of data, high if information carried is vague and low if relevant) loss. We want this quantity to be as great as possible, so that the children nodes are as farther as possible from the uniform distribution. So entropy loss means impurity loss (here regarded as entropy, but more in general in Decision Trees it is an impurity measure, which can be another metric) from parent node to children.

4 Software Development Modality

Entire software was coded by scratch, except for the Machine Learning libraries, packages installed via NuGet, the Visual Studio package manager. All the libraries here used are from the Accord.NET Framework ([**6**]).

The code has been written in C# 7.3 Language. The advantages of OOP paradigm were exploited in writing a modular and reusable code (reusable in the sense that the same code runs for different data sets). For conceptual clarity, classes were divided in different namespaces, each containing classes prone to accomplish related tasks.

Note: the program is though to be user-interactive, in such a way to dispose of an improved control on the kind of analysis performed, the data set to be used, the partition modality.

4.1 Code written from scratch

4.1.1 `MLProj2018` namespace

Two classes are here present: `InOutStreams`, which is `static` containing the `static` fields specifying the data source and reports drop destinations, with the `static void` methods `GoToFolder`, which looks for the folder where the I/O location lies, `DirectoryManager` that, given a `Dictionary<int,string>`, and the `int Flag` user choice about the data set, selects in the source folder the proper input file, and `DirectoryCreation` which accepts `int Flag` values about the user choices about test and train sets partition and data preprocessing, and accordingly creates, if not present yet, the proper folder to drop report files.

The other class is the `Program`, containing the only `static void Main()` method, from which all what needed for analysis is called.

4.1.2 `DataWorkTable` namespace

Here classes for data modelling, import and preprocessing lay. The first is the `DataObject` class, as described in Section 2. This class itself has little utility, it is solely thought for data storing. In order to fill it with actual data, methods of the following class are to be evoked.

The second class is `DataSetImport`, conceived as import methods wrapper, listed below:

- `DataObject FeaturesImport()` which reads (via `System.Data`, `Accord.IO` classes, such as `DataTable`, `StreamReader`) the data from file and stores them in the previously instance of `DataObject` representing the data set, which is returned. Category vectors are also initialised, by reading the last column of the data table and calling the following:

- `void CategorizeAsInts(DataObject data)` which translates the string entries of `Categories` in integer values. Ordinal of the former is not mantained in the latter. A phenomenological argument of the categories distribution upon data samples is given in Sections 5 and 6.

Class `DataPartition`, according to the value of the `FlagPartition` given to the constructor, separates properly the test and training set. The methods provided are

- `DataObject[] TestSetExtraction(DataObject DataSet)` which operates the split and return two `DataObjects`, containing indeed test and train set. This task is accomplshed by the generation of N_{TEST} random integers ranging in $[0, N_{TEST}-1]$. The data set entries indexed by these are kept apart forming the test set . A cycle is then performed on the orginal data set, discarding the previoulsy chosen records and storing the others in the train set `DataObject`.

- `void PrintPartition(DataObject[] SetsOfData, ...)` that prints the data set splitted on file.

Finally the class `DataProcessing` contains `static` fields about the users choice of preprocessing or not and, if the latter is positive, how many principal components are to be used. The rationale behind the `public static` choice is to allow the main program to perform a cycle to set these values if the user wants to preprocess, instead of bifurcating the program workflow according to this single decision.

- `DataObject DoThePCA()`. This returns the new data set, that is the original one on which dimensionality reduction is performed. In the execution of this code, two windows pop up: the eigenvalues *scree plot* (Values vs. the number identifying the eigenvalue itself), in such a way to let the user choose how many Principal Components use in the following (the idea is that the eigenvalues which capture the more variability are the ones related to the principal directions, i.e. their eigenvectors, are the greater ones, so the choice is made more "visual" by such a plot, see [**5**], Appendix B) and the scatter plot of the new data, if the principal components chosen to be kept are 2.

The returned type of this method is a `DataObject`, eventually submitted to splitting. The PCA transformation is done by the `PrincipalComponentsAnalysis` class, provided in the namespace `Accord.Math.Decompositions`.

4.1.3 `LearningAlgorithms` namespace

This last namespace contains the `LearningAlgorithmsClass`, which again has as fields a copy of the test/train sets (copies are nothing but references, since types passed to the constructor are `DataObjects`), the values of the choice flags and the `clock` variable, which keeps track of the time elapsed for each of the algorithms explained. The methods are, along with a `PrintReport` method (overloaded), the following:

- `void SVMGaussianKernel()` which performs a nonlinear multiclass classification using SVM with Gaussian transformation kernel. The final choice of the parameters is deeply related with the data sets being analysed and its quantitative/qualitative features. `Complexity` parameter is set egual to 3 (as in [**1**]), while whether or not setting the σ or estimating it is deeply case related, see Section 5.

- `void SVMPolynomialKernel()`. Estimation of complexity susceptible of changes during the analysis, depending on the case.

- `void TreeLearning()`. This last method implements the C4.5Learning algorithm. No parameters tuning is required for this one.

The invocation of these methods does not accept as argument data because the `DataObject` containing the data set is passed once for all to the constructor of the `LearningAlgorithmsClass`, so data are available as fields of the class itself.

4.2 External libreries

All the external libraries are borrowed from the Accord.NET Framework for Machine Learning purposes as well as file stream libraries.

4.2.1 Files Streams

The `Accord.IO` namespace provides the `CsvReader` class, which allows to read numerical as well as textual data types from a `.csv` file, specifying this latter.

4.2.2 PCA

Principal components analysis can be easily implemented via the `PrincipalComponentsAnalysis` class. A property can be set to subtract the data matrix the means vector, in a way to extract the eigen values/vectors of the covariance matrix.

4.2.3 SVMs

`MulticlassSupportVectorLearning<TKernel>` class (for the Gaussian case the `Gaussian` struct has been manually fiddled in order to tune the parameter that, as in [**1**] pointed out, is to be set performing trials and checking which value yields the best result) is used to instantiate a learning model, which propetry is the optimization algorithm, kept through the whole analysis set to `SequentialMinimalOptimization<TKernel>` (this latter is itself another class of the Framework from the `Accord.MachineLearning.VectorMachines` namespace). For both Gaussian and Polynomial kernels the structure is the same, is sufficient to specify whether `Gaussian` of `Polynomial` is intended to be used, in place of `TKernel`. The other parameters are tuned to minimize the error yielded by the analysis. Is has been observed that setting `UseKernelEstimation` to `true`, for the wines data set, spoils performance in terms of error, while it gives good results for the other two data sets. This sensitivity forbids a general approach. That parameters are **not** estimated means that another method is used instead (in [**7.1**] what emerges is that with **no** σ tuning, other methods are used, for example gradient ascend optimization). The estimate, if any, could be used as starting point for the optimization algorithm (as by the second usage example given in [**7.2**]). This optimization has nothing to do with the Sequential Minimal Optimizer, that is set as a learning algorithm in the case of kSVM: the former concerns kernel parameters, such as σ, the latter is about the learning procedure itself, that is computing the **model** parameters $(\mathbf{w},b)\in\mathbb{R}^d\times\mathbb{R}$.

4.2.4 Tree

Here the class `DecisionTree` from the `Accord.MachineLearning.DecisionTrees` can be used for the decision tree induction. In particular, C4.5Learning algorithm is used, with the `C45Learning` class istantiation, which requires the class labels to be specified. No parameters tuning is required.

The error is computed as a `ZeroOneLoss` (class) function, and written in the summarising output. For this, real test set entries categorical labels are compared with how predicted by the `Decide(testset)` method, provided for each of the training algorithms used.

4.2.5 Plots

Accord provides the `Accord.Controls` namespace, through which is possible to produce scatter plots.

5 Evaluation and Results

5.1 Data Sets Comparison and Learning Algorithms Rating

As measures of performance two quantities have been considered: execution time and error, in terms of zero one loss, defined as

$$L_{0-1}(h(\mathbf{x}_i),y_i)=\begin{cases}1 & \text{if } h(\mathbf{x}_i)\neq y_i\\ 0 & \text{otherwise}\end{cases}$$

All the predicted versus real values of the categorical label are available in the `RealVsPredicted.dat` reports. Results are more easily readable in form of tables. Notice that in [**1**] the error metric used is slightly different. There the authors use the Mean Absolute Deviation, defined as

$$MAD=\frac{1}{N}\sum_{i=1}^{N}|y_i-\hat{y}_i|$$

where $\hat{y}_i$ denotes the guessed label. This discrepancy may lead to differences, but in any case in both the metrics presented the error is 0 if the guess is correct. The error there found is **0.46**, which is not far from what here found.

In the following, three data sets are compared. Even if in the last two examples results are better, it is to be kept in mind that consistency between wine data set and the two trial ones is, if any, very weak. Wine data set is one order of magnitude bigger, either in number of samples and in sample dimension. Furthermore, data disposition in features space is different (in this visualisation remark, PCA helped, see below). It does not mean that wine data set is not tractable, but it may suggest that there are techniques which perform better upon it. In fact Decision Tree has proven to be fastest, even if slightly worse than Gaussian kSVM, but in the most of the cases, a small loss in accuracy is balanced by an improved execution speed.

In general, Gaussian SVM attains good results, but taking more time than the other two models used. Polynomial SVM is used keeping the polynomial degree fixed at 2. Different degrees are observed not to render a better performance, and are avoided because of the overfitting threat. C4.5Learning Decision Tree proved to be a good compromise between accuracy (on average satisfactory) and time (non negligibly faster). In the particular case of the wine data set, Gaussian SVM shows a high sensitivity to parameters estimation (accuracy is higher if parameters are not estimated), while for Polynomial SVM such a choice is almost immaterial.

5.1.1 Wine DataSet

Analysis is performed with and with no PCA; in the former case results are less precise, thus preprocessing is rather used only as a visualisation technique. Entries of the following tables are averages over errors and times of 30 program runs (`StreamWriter` with the `append : true` clause has been set, so that each result has been added to the output file). It is important to remark that results are sensitive with respect to the kernels parameters and complexity heuristics. In the following two tables, averages of error and execution time are reported, related to the choice: $\sigma = 0.1$, `Complexity = 3` for Gaussian SVM, estimations for both kernel and complexity `true` for Polynomial SVM (**but if** `false` is set for Polynomial kernel estimation, things do not really change for this latter).

No PCA		Error	Time (seconds)
10% Partition	Gaussian	0.4031446	3.518457
	Polynomial	0.504822	3.713693
	Decision Tree	0.426834	2.221279
33% Partition	Gaussian	0.429643	2.7000838
	Polynomial	0.522514	2.412476
	Decision Tree	0.447905	1.598985

If `true` is set for the kernel estimtion in Gaussian kernel, one obtains

No PCA		Error	Time (seconds)
10% Partition	Gaussian	0.579245	6.711302
	Polynomial	0.513208	3.113191
	Decision Tree	0.422468	2.251699
33% Partition	Gaussian	0.571857	5.349196
	Polynomial	0.530019	3.44943
	Decision Tree	0.44384	1.596546

In [**1**] one of the main points is the SVM outperforming other methods, and here the same is found **but** these data are relative to the above mentioned choice of the parameters estimation flag (and the fact that there Regression and NNs are compared to SVMs). In general, it is observed, even for the other datasets, that Decision Trees is outperformed by Gaussian SVM by few in terms of error, but keeping the best classification method in terms of time. However, some variability is also observed. This is believed to be related with the random data set partition. The thing that is more to be pointed out is that results showed in [**1**] are partially reproduced, but the use of Decision Trees puts SVM on a lower quality level.

5.1.2 Biomechanical features DataSet

This data set is dimensionally smaller, in terms of data sets records and array dimensions. So the classification task is considerably easier, and much faster to run. Nevertheless, results here found suggest the correctness of the approach described above for the wine data set, and a sharp sensitivity to the partitioning is detected. The next table reports

values related to the setting kernel estimation set to `true` for Gaussian kernel (otherwise results are far worse: tuning σ with numbers ranging from 0.01 to 100 situation does neither get better or worse, but stationarily not satisfactory) and kernel estimation `false`, complexity heuristics `true` for Polynomial kernel, because in this case setting manually σ leads to a very poor performance (high errors).

No PCA		Error	Time (seconds)
10% Partition	Gaussian	0.322581	0.294578
	Polynomial	0.170968	0.307171
	Decision Tree	0.217204	0.210096
33% Partition	Gaussian	0.168932	0.2083
	Polynomial	0.191909	0.229719
	Decision Tree	0.217159	0.115921

Then, leaving unchanged the flags for the Gaussian kernel and tuning to `true` the complexity heuristics for the Polynomial the following is obtained

No PCA		Error	Time (seconds)
10% Partition	Gaussian	0.358065	0.32899
	Polynomial	0.206452	0.325725
	Decision Tree	0.23333	0.19607
33% Partition	Gaussian	0.175728	0.202437
	Polynomial	0.193204	0.250407
	Decision Tree	0.21165	0.11792

5.1.3 Iris DataSet

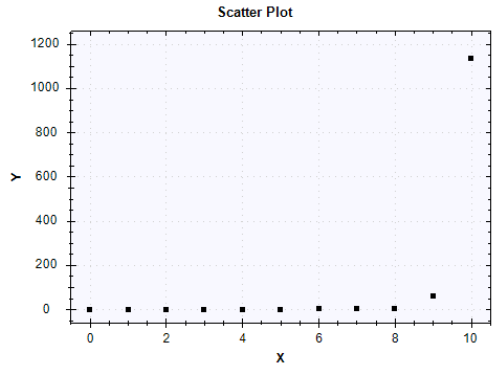
The two tables report results respectively for Polynomial kernel estimation set to `false` and `true`. By tuning $\sigma = 0.1$ for Gaussian or estimating it, it appears not really impacting, yielding the same results as follows. This is believed to be due to the more separability of the Iris data and the lower dimensionality.

No PCA		Error	Time (seconds)
10% Partition	Gaussian	0.037778	0.185715
	Polynomial	0.077778	0.178252
	Decision Tree	0.04444	0.02002
33% Partition	Gaussian	0.045333	0.0119374
	Polynomial	0.045333	0.093229
	Decision Tree	0.06	0.011979

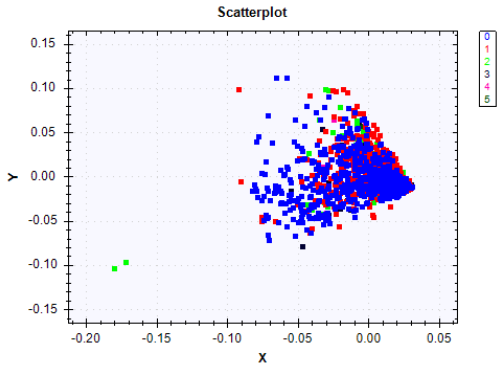
No PCA		Error	Time (seconds)
10% Partition	Gaussian	0.035556	0.139583
	Polynomial	0.086667	0.081771
	Decision Tree	0.055556	0.032787
33% Partition	Gaussian	0.044667	0.162905
	Polynomial	0.084667	0.057813
	Decision Tree	0.054667	0.020312

5.2 Graphical results

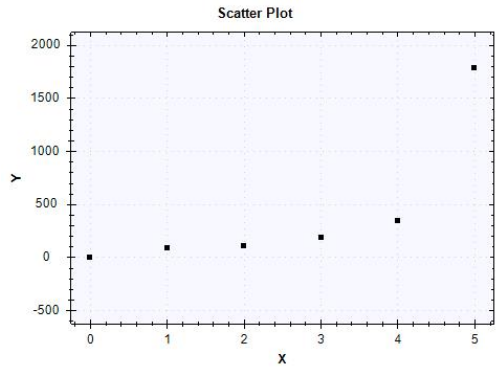
In Figure 1 some graphical outputs are reported, to give a face to the data. However, as previously stresses, 2 PCs are thought to be relevant by observing the scree plots in Figure 1: there the first two eigenvalues are assumed to be the more relevant, but it could be argued that the quantitative difference between the second and the third is not big, so there is no reason to take only the first two rather than the first three. This dispositions in reduced dimensions are **not** believed to be the real data behaviour, it is only for sake of visualisation. Scatter plots however do not render information about separability or not in 11 dimensions.



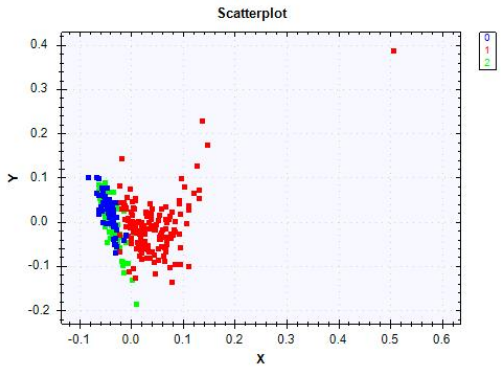
(a) Wines Scree Plot, reporting covariance eigenvalues magnitude.



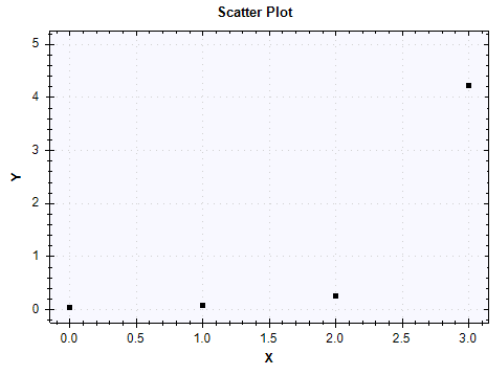
(b) Scatter plot of the features in the reduced features space.



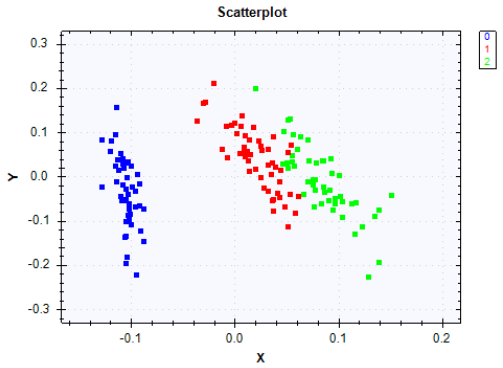
(c) Biomechanical features Scree Plot, reporting covariance eigenvalues magnitude.



(d) Scatter plot of the features in the reduced features space.



(e) Iris Scree Plot, reporting covariance eigenvalues magnitude.



(f) Scatter plot of the features in the reduced features space.

Figure 1: Using only two PCs data are clearly not linearly separable. This representation is believed to be far from reality though, but nevertheless non linear SVM were used to prevent the problem of trying to classify samples indeed non separable.

6 Conclusion

Different data sets have been set under analysis. Results for the wines data set are not fully satisfactory, rendering a classification error too sensitive to whether parameters are estimated or not. Recall that estimation, if performed, is a starting point of another method that produces a suitable kernel parameter. Of course goodness of results is strongly dependent by properly selected kernel parameters, and, also important, is not to be forgotten that no general procedure to mine data is available, so it is also reasonable that for any data set, and for any algorithm, changes are to be made in order to achieve good and reliable results (*No Free Lunch* Theorem).

A further reason of error in the wine data sets case is strongly believed to be uneven distribution of class labels among the data sets entries, see wine data set histograms Figure 2. This leads to **underfitting** of the models. A dominant presence of 0 and 1 class labels is observed in the training set, so the model learns too well to recognise 0 and 1 labels, some of the 2 labels test entries are guessed, but 3, 4, and 5 labels are almost unknown to the trained model, then such categories are not recognised. As a clue, histograms of class distribution among data are reported, both for the entire data set and for the training set. It is also worth noting that in the original data sets the categories spectrum does not span values but [3,8].

In fact, in the Iris data set (as well as in the Biomechanical features one) such an uneven distribution is not present, instead classes are almost uniformly (in the former, for the BM data set more nonuniformity is observed) distributed among samples. In this case underfitting is not occurring, yielding a lesser error.

7 Further Improvements

Some things of the presented work can be improved. For example, effort has been put on the consistency and robustness of the code with respect of more possible input files available (provided that the format of these is compatible, so the label column shall be put as last one; only data sets consisting in a numerical matrix and a label column have been taken into account) as well as the capability to self organize the stream flow from and to the proper folder, in response of the users instructions.

But another important point shall be implemented: to give the user the possibility to fiddle more actively and rapidly the parameters of the classification model, as well as the class of algorithms to use. This rises as a first importance necessity when data are actually to be mined, so one cannot say with certainty which model is the most suitable with which parameters. Multiple run are requested, performing trial error procedure with different models and parameters setting.

In addition, few considerations have been done about the use of PCA, presented exclusively as a visualisation tool. It could be instead deployed for a wider scope, by keeping more PCs and, even being not able to plot the data, trying to perform a *Cluster Analysis* on the reduced dimension samples.

8 Tools Utilised

- Integrated Development Environment Microsoft Visual Studio Community 2017 Version 15.7.2 to develop C#, Version 7.3 code, Framework .NET 4.6.1;
- Accord Framework libraries;
- TeXstudio ver. 2.11.0 for LaTeXtypesetting.

References

[1] Cortez, P., Cerderia, A., Almeida, F., Matos, T., Reis, J., *Modeling wine preferences by data mining physiochemical properties*, Elsevier, 2009, doi: 10.1016/j.dss.2009.05.016.

[2] Red Wine Quality Data Set on Kaggle.

[3] Biomechanical features of orthopaedic patients, idib.

[4] Iris Data Set on UCI Machine Learning Repository.

[5] *Introduction to Data Mining*, Tan P., *et al.*, website.

[6] Accord.NET Framework.

[7] Souza, C. R., *A Tutorial on Principal Components Analysis with the Accord.NET Framework*. Department of Computing, Federal University of Sao Carlo, Technical Report 2012. Webpage.

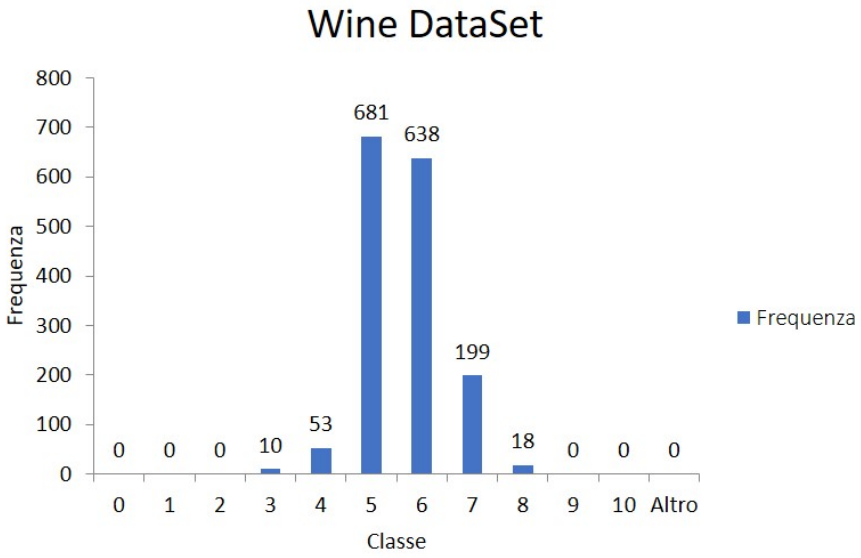
[7.1] Gaussian kernel struct source code on GitHub, webpage.

[7.2] Overview and examples of `MulticlassSupportVectorLearning` class in the Accord.NET Framework, webpage.

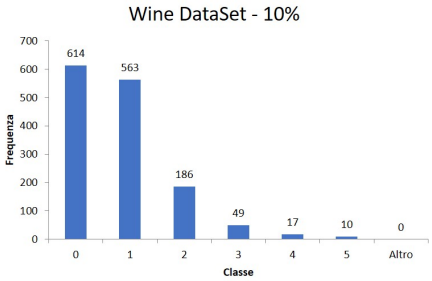
[7.3] C. Souza (Accord.NET Original author) post on his webpage about Decision Trees.

[8] Smith, L. I., *A tutorial on Principal Components Analysis*, 2002. Webpage.

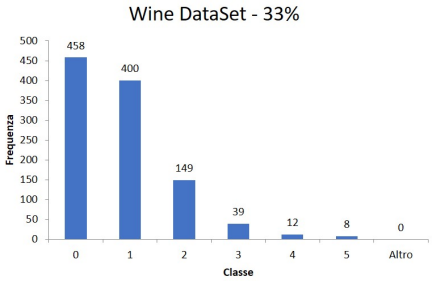
[9] Di Buccio, E., Schimd, M., Material for the course *Scientific Computing and Object Oriented Programming*, held at Università degli Studi di Padova within the Master Degree *Mathematical Engineering* courses, A.Y. 2017/2018, Spring Semester.



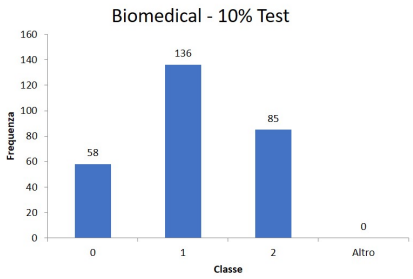
(a) The distribution of the classes is peaked on the classes 5 and 6, that means 0 and 1, after processing.



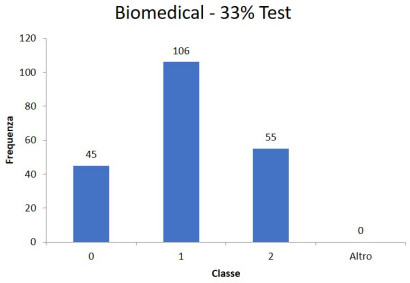
(b) Classes distribution on the training set, partitioned 0.1 TestSet and 0.9 TrainSet.



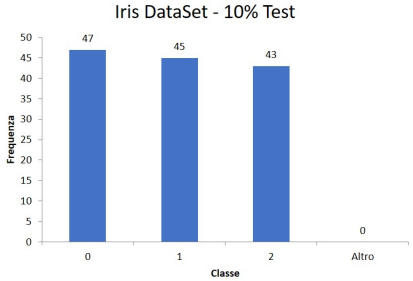
(c) Classes distribution on the training set, partitioned 1/3 TestSet and 2/3 TrainSet.



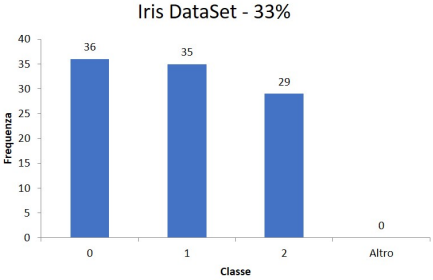
(d) Classes distribution on the training set, partitioned 0.1 TestSet and 0.9 TrainSet.



(e) Classes distribution on the training set, partitioned 1/3 TestSet and 2/3 TrainSet.



(f) Classes distribution on the training set, partitioned 0.1 TestSet and 0.9 TrainSet.



(g) Classes distribution on the training set, partitioned 1/3 TestSet and 2/3 TrainSet.

Figure 2: The uneven distribution of classes among wine samples is clear. While even in the Biomechanical features data set distribution is far to be uniform, in this latter the reduced dimensionality of the sample space prevented the models to underfit.