Glass identification: A mathematical perspective

Davide Bussone
MSc Data Science and Engineering
Politecnico di Torino

04 July 2020

Abstract

Glass identification dataset was generated to give support to criminal investigation. In a crime scene, the glass left can be used as an evidence, so it is important to correctly identify it.
The aim is to find a model able to recognize the different types of glass, exploiting physical information contained in the case study database.

1 Data exploration

The dataset [1] used to achieve the goal has 214 items, each one described by an ID and by nine attributes. In addition, each element is labelled by the class of the glass. The attributes are all float numbers and they have a physical nature. For example, refractive index is an indicator of the speed of the light through the different types of materials (glasses in this case). From these features, the model built will recognize if a glass comes,for example, from a building's window or from a container.
Regarding the numerousness of each class, histogram 1 shows the lack of a uniform distribution. There are overepresented classes, for instance "building-window", and absent classes, like "vehicle-non-float-processed". Moreover, the dataset is complete, in fact there are not missing values to handle.

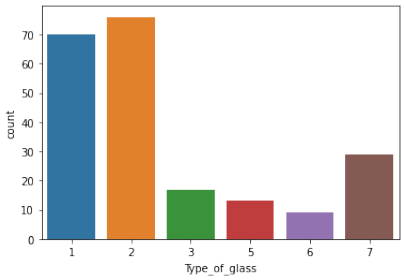

Figure 1: Classes distribution

A further step of data exploration involves checking the distribution of values for each feature in each class, in view to point out their weight in the classification task. Violin plot [5] was chosen. It is a method of plotting numeric data and it is a combination of a box plot with a kernel density plot. This type of graph shows the median value, with a white dot and the interquartile range, with a black bar in the centre of the violin. It also illustrates the lower/upper adjacent values, useful to catch outliers, which are the elements beyond the adjacent values. Finally, violin plot displays the entire distribution of data. 14 elements were removed from the dataset because they contain more than two outliers.
A distplot [3] was realized to underline the variance of the attributes. All these distributions are portrayed in the following graphs.

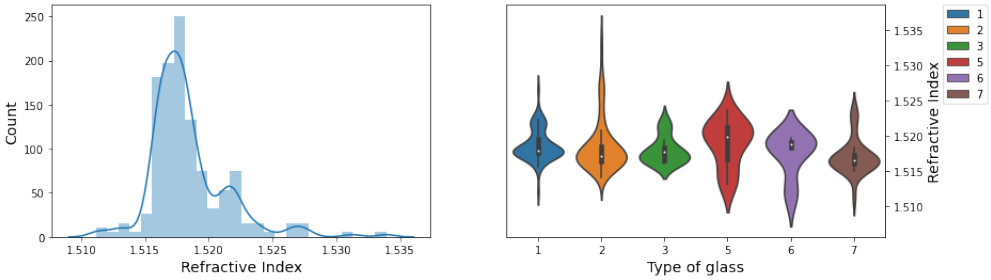

Figure 2: Reflactive index

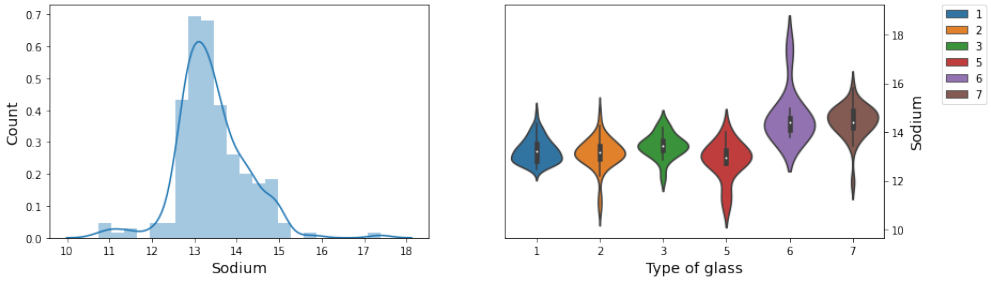

Figure 3: Sodium

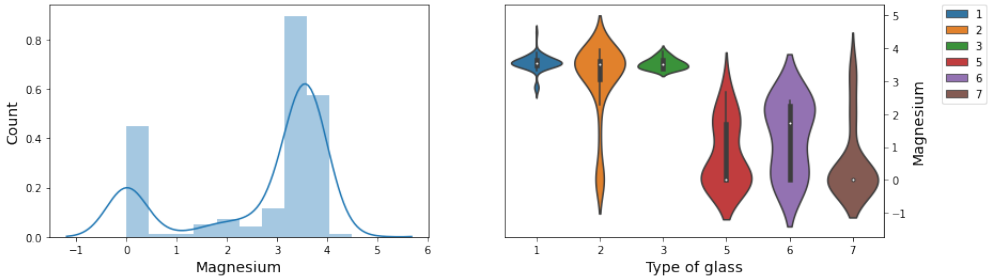

Figure 4: Magnesium

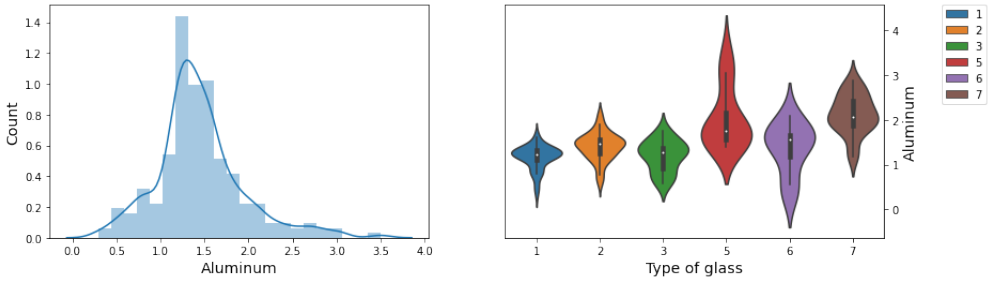

Figure 5: Aluminium

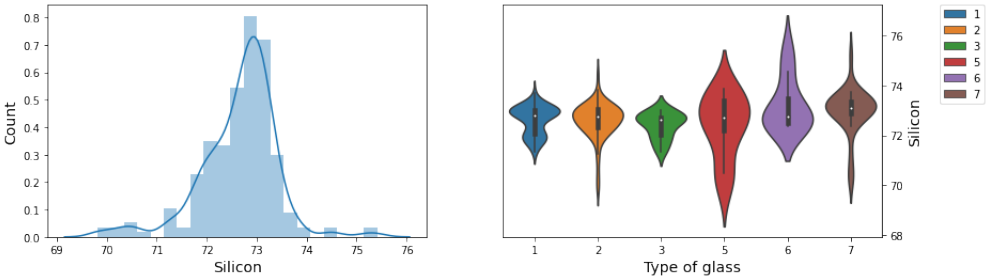

Figure 6: Silicon

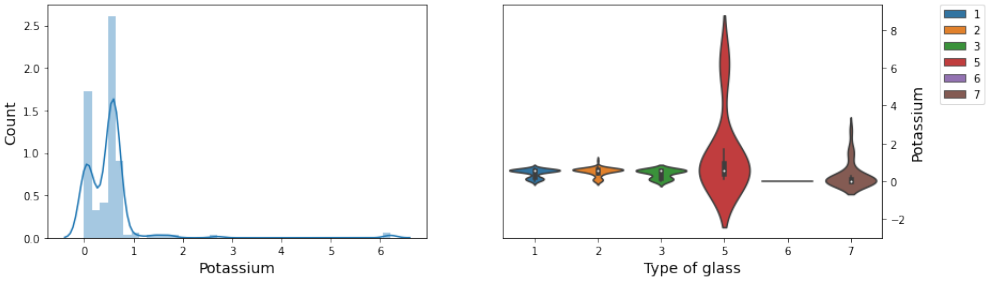

Figure 7: Potassium

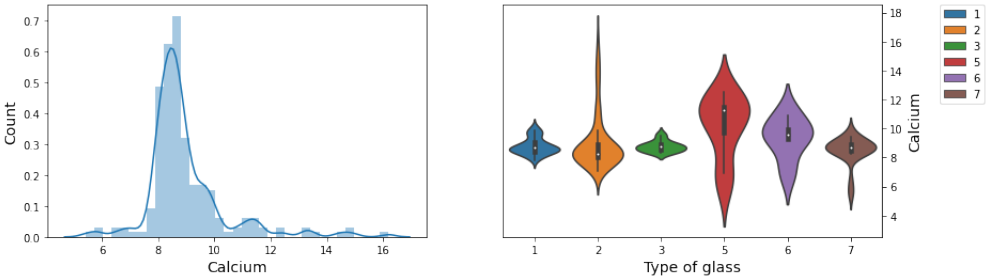

Figure 8: Calcium

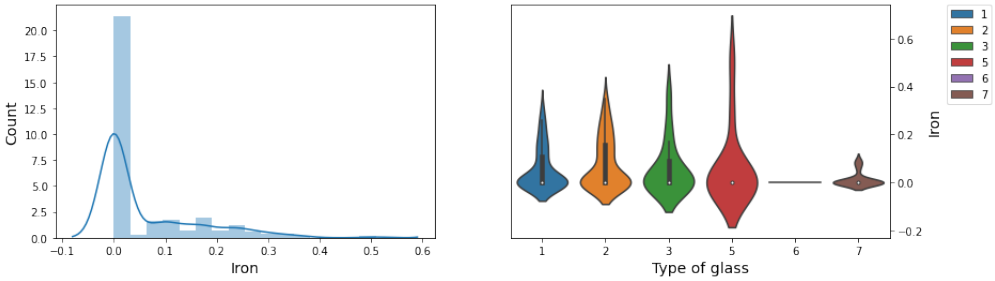

Figure 9: Iron

The last important element in this section is the correlation among features, described on matrix 10. A heatmap [4] was chosen. It is a graphical representation based on colour codification to point out different values. Each block provides an immediate impact on the relationship between features in a very intuitive way. For instance, a dark red value suggests a high correlation, on the other side dark blue is synonymous of a negative one. It is a key element because in presence of a too strong correlation between features, it will be better to apply a dimensionality reduction technique, like principal component analysis, in view to avoid overfitting in the model.

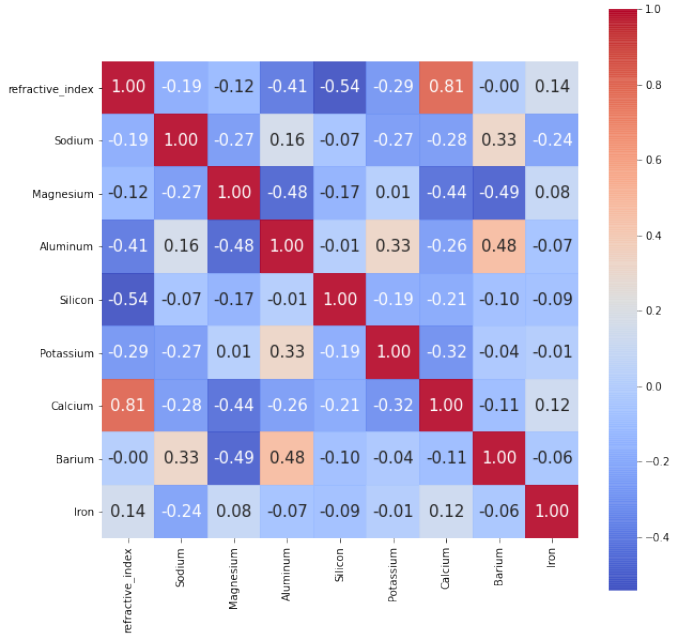

Figure 10: Correlation matrix

2 Preprocessing

2.1 Standard scaler

Standard scaler is a preprocessing algorithm that substracts the mean value and then it scales the unit variance, dividing all values by the standard deviation. The rescaled measure follows the formula $z = \frac{(x-\mu)}{\sigma}$, where μ represents the mean of the training samples and σ the standard deviation.
To normalize a dataset is a popular action for machine learning estimators, in fact some algorithms behave in a very bad way if the features are not closed to standard normally distributed data.

2.2 Dimensionality reduction

[6] After the standardization process, it usually might be useful to adopt a dimensionality reduction algorithm. This reduces the execution time, facilitates a more general learning and lets an easier interpretation of the data. The process takes data in a high dimensional space and maps it into a new space with a smaller dimension, but at the same time the aim is to lose less information as

possible.
Mathematically, the original data belongs to R^d, a reduction algorithm takes them to R^n, where n<d. So, it looks for a matrix $W \in R^d$ which brings to a mapping $x \mapsto Wx$. The natural choice of W is made so that it will lead to a reasonable recovery of the original x; the ideal case, in which x is exactly recovered, is generally impossible. The method proposed is called Principal Component Analysis (PCA). In PCA both compression and recovery are performed by linear transformations. The algorithm catches the linear transformations for which the differences between the recovered vectors and the original ones are minimal in the least squared sense. So it solves a problem having objective function reported in 1.

$$argmin \sum_{i=1}^{m} \|x_i - UWx_i\|_2^2 \tag{1}$$

In equation 1, x_i, with i=1,...,m, are m vectors $\in R^d$. W is a matrix $\in R^{n,d}$ that induces the mapping $x \mapsto Wx$. U is a matrix $\in R^{d,n}$ used to approximatively recover each original vector x from its compressed version. The optimal solution is obtained by setting $\widehat{W}=V_L$, where V_L represents a matrix containing the L eigenvectors with the largest eigenvalues of the empirical covariance matrix $\Sigma=\frac{1}{N}\sum_{i=1}^{m} x_i x_i^t$.

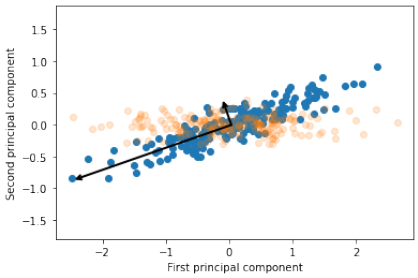

Figure 11: Orthogonality of the first two principal components in PCA algorithm

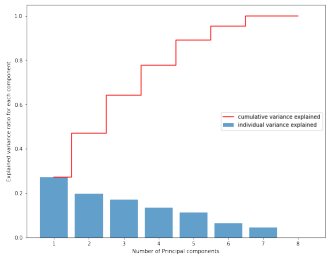

Figure 12: Explained variance between principal components

The diagonal line in figure 11 represents the first principal component. The directions of principal components are the ones along which the data have maximal variance. Each subsequent principal component is orthogonal to the previous one.
So, the n principal components considered with PCA can be seen as the largest eigenvectors extracted from the covariance matrix Σ built from the original data vectors x_i. As graph 12 infers, by taking 5

principal components, more than 90 % of variance information is kept and the data space is reduced with the mapping $\mathrm{R}^7 \mapsto \mathrm{R}^5$.

3 Model Selection

The dataset is now split into two subsets, the training and test dataset, with proportion 70%-30%. The training dataset is the sample of data used to fit it. So, in other words, the model learns from this data. Instead, the test dataset constitutes the set of data exploited to provide an unbiased evaluation of a final model generated by the training set. In addition, stratify's parameter was set to "y", because it manages to preserve the proportion of target as in original dataset. Furthermore, in view to generalize better the model, a portion of the training dataset is used to supply an unbiased evaluation of a model fit during the training phase, called "validation set". At the same time validation set is used also to tune model hyperparameters.
In this case study, the amount of data is quite low, so in order to avoid a waste of this information on validation, a K-fold cross validation is performed. Basically, the original training dataset is partitioned into k smaller sets (the folds) of approximatively equal size. Each fold, in turn, acts as a validation set and the remainders as a training set. The resulting error is estimated by averaging the k resulting MSE error estimates, following the equation 2.

$$CV_k = \frac{1}{k} \sum_{i=1}^{k} MSE_i \tag{2}$$

This method, as already mentioned, is often applied for model selection, which consists of selecting the combination of hyperparameters, among the proposed ones, which minimizes more the MSE error. Mathematically, as inputs there are a training set S, a list of hyperparameters Θ, a machine learning algorithm A and an integer k, representing the established number of folds. So, S is partitioned in k subgroups, denoted by $S_1, ..., S_k$. Then, for each $\theta \in \Theta$ and by iterating over each fold k, the average error is computed according to relation 2. The procedure gives as output the θ^* best combination of hyperparameters minimizing the MSE error. In the next sections, this method is applied on different machine learning algorithms A.

4 The Algorithms

4.1 Logistic regression

The first machine learning algorithm proposed is the logistic regression. This process does not pretend to model directly the class of a sample, but it takes into account the probability that a certain sample belongs to a determined category. In classification tasks, logistic regression is preferred to the simplest linear regression. To explain that, consider some attributes $X_1, ...X_n$ which concurs to classify an item while Y represents the classification response. By using linear regression, some estimates might be outside the $[0, 1]$ interval in such a way that it is complex to interpret them as probabilities. For this reason, logistic regression is preferable.
In mathematical terms, the aim is how to model the relationship between $P(X) = P(Y = 1|X)$ and X. Linear regression algorithm, supposing that we are in a binary case, describes it with the equation $P(X) = \beta_0 + \beta_1 \mathrm{X}$. In this context, it is fast to discriminate between two classes, because it is sufficient to fix a threshold (0.5 in most cases) and $P(X) < 0$ if the threshold is not exceeded and viceversa. The problem is that for values close to zero, estimates will become lower than zero itself,

but this is a numerical value not relevant in probabilistic terms. To maintain the probability of a predicted value, logistic function is introduced, following the relation 3, applied again in the binary classification case.

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} \tag{3}$$

Moreover, by taking the logarithm of both sides, the Logit function is obtained, as it is showed at the first member of equation 4.

$$log(\frac{P(X)}{1 - P(X)}) = \beta_0 + \beta_1 X \tag{4}$$

To train the model, a method called maximum likelihood estimation is performed. This process estimates the parameters of a probability distribution function by maximizing a likelihood function, so that, under the assumed statistical model, the observed data is the most probable. The computation in a binary classification context is reported in 5.

$$L(\beta_0, \beta_1) = \prod_{i:y_i=1} P(X_i) \prod_{i:y_i=0} (1 - P(X_i)) \tag{5}$$

In the case study, through K-fold cross validation, as already mentioned in section 3, the hyperparameter C was tuned. This entity represents the inverse of regularization. It is useful, because in many cases, maximum likelihood estimation, applied on β_i attributes, might lead to overfitting. As a consequence, an addition of a quantity in the cost function of logistic regression may decrease the variance of the trained model. This process is the regularization, so having lower values of C brings to a stronger regularization. With K-fold cross validation, the best value of C results 1000.
Finally, for each machine learning algorithm analyzed in this glass identification experience, the confusion matrix and the accuracy score are reported in graph 13. A confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j. In case of binary classification problems $C_{0,0}$ represents the total number of true negatives, $C_{1,1}$ the true positives, $C_{1,0}$ the false negatives and $C_{0,1}$ the false positives. Then, the accuracy score follows equation 6 and it measures how many items (in percentage) are correctly classified.

$$accuracy = \frac{C_{0,0} + C_{1,1}}{C_{0,0} + C_{1,1} + C_{0,1} + C_{1,0}} \tag{6}$$

Also f1-score metrics is performed, it is defined by the following equation.

$$precision = \frac{C_{1,1}}{C_{1,1} + C_{0,1}}$$
$$recall = \frac{C_{1,1}}{C_{1,1} + C_{1,0}}$$
$$f1score = 2 * \frac{precision * recall}{precision + recall}$$

After the tuning of hyperparameters, the accuracy score on the test set is equal to 95 % and the f1 score reaches 95.3 %.

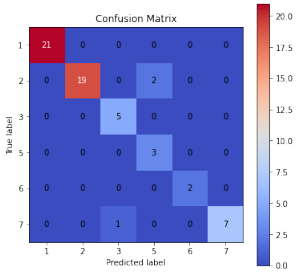

Figure 13: Confusion matrix logistic regression

4.2 Decision trees and Random forest

Decision trees are a supervised learning method used for classification and regression. Its goal is to build a model that predicts the value of a target variable by learning decision rules derived from the data items. This approach consists of segmenting the predictor space into a number of simple region, based on the features values. So, from a set of $X_1, ...X_p$ values, the predicted space will be stratified into j distinct and non-overlapping regions $R_1, ..., R_j$. Next, for every observation that falls into the region R_j, the same prediction is made. Actually, this process performs well on the training dataset, but it is usually subjected to overfit the data, leading to poor test set performance.
A better strategy is to develop, initially, a very large tree T_0 and then prune it back in order to obtain the subtree. This approach is called "pruning". Given a single subtree, it is possible to estimate the test error by exploiting the K-fold cross validation, already described in 3. The application of cross-validation is computationally expensive, because potentially the number of subtrees may be very high. It is more feasible to consider only a small set of subtrees, listed by a non-negative hyperparameter α, known as cost-complexity measure. Equation 7 illustrates it.

$$R_\alpha(T) = R(T) + \alpha T \tag{7}$$

In relation 7, $|T|$ represents the number of terminal nodes, while R(T) is traditionally defined as the total misclassification rate of the terminal nodes. R(T) presents the behaviour described in equation 8 and it represents the fraction of training observations not belonging to the most common class.

$$E(X_m) = 1 - \max_k(\widehat{p}_{mk}) \tag{8}$$

In equation 8 $\widehat{p}_{mk}$ represents the proportion of training observations in the m_{th} region that belongs to k_{th} class. It seems to be an optimum criteria, because it keeps in mind what is the most common occurring class, coherently with the classification's task. But, it was discovered that misclassification rate is not sufficiently sensitive for tree growing. In fact, other two measures are preferable. The first one is the Gini index. Its computation is described in 9. Intuitively, it pretends to measure how often a randomly chosen element from the set would be incorrectly labelled.

$$Gini = \sum_{i=1}^{k} \widehat{p}_{mk}(1 - \widehat{p}_{mk}) \tag{9}$$

Then, the other measure is the entropy, whose aim is to catch the randomness in the information

being processed. As you can infer from equation 10, entropy exploits the log operator, instead of using simple probabilities. This brings this index to be more computationally expensive, but also capable to benefit of advantageous properties, like the additive one.

$$Entropy = -\sum_{i=1}^{k} \widehat{p}_{mk} \log \widehat{p}_{mk} \tag{10}$$

Both indexes are very sensitive to node purity, in fact a small number of them attests that a node contains predominantly observations from one class. As a consequence, they are both exploited as a criteria in this case study. After the performance of k-fold cross validation with five splits, Gini's index turns out to be the best performing criteria in this experience.
Another hyperparameter set was the max depth of the decision tree. It is important to say that this value is not always the same quantity as depth of a decision tree, but it is a way to pre-prune a decision tree; so, if a tree is already as pure as possible at a depth lower than max depth, it will not continue to split until the established value. After cross-validation, max depth's best value was 5. By evaluating the model on the test set with these two hyperparameters already tuned, both the accuracy and f1 score reach 100 % and in picture 14 is showed the confusion matrix. As you can infer from the graph 15, attribute "Iron" influences a lot the classification task, decision tree performs very well when there is an attribute more important than others.

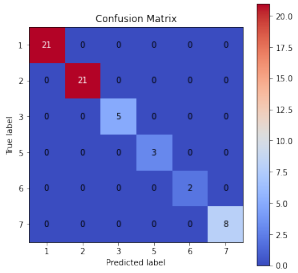

Figure 14: Confusion matrix decision tree

Decision trees have different advantages, such as a simple explanation and the possibility to handle qualitative predictors without the need to create dummy variables. Unfortunately, trees generally do not have the same level of accuracy as some other classification approaches and they are not so robust. The "glass identification" dataset is a particular case. In fact, a small change in the data can cause a large change in the final estimated tree. In view to overcome this, another approach intervenes, whose name is "Bagging" or "Bootstrap aggregation". This procedure takes many samples from a single training dataset, then it builds a separate prediction model using each training set and, finally, it averages the resulting predictions. Bagging lets solve the lack of more training sets. Mathematically, a single training set is split into B smaller sets $\widehat{f}^1(x), ..., \widehat{f}^B(x)$ and the predicted value derives from the most commonly occurring class among the B predictions. Bootstrap aggregation could be improved by decorrelating the trees. Random forest algorithm starts from this idea. As in bagging, a determined number of decision trees on bootstraped training sample is built. But during the processing of these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use

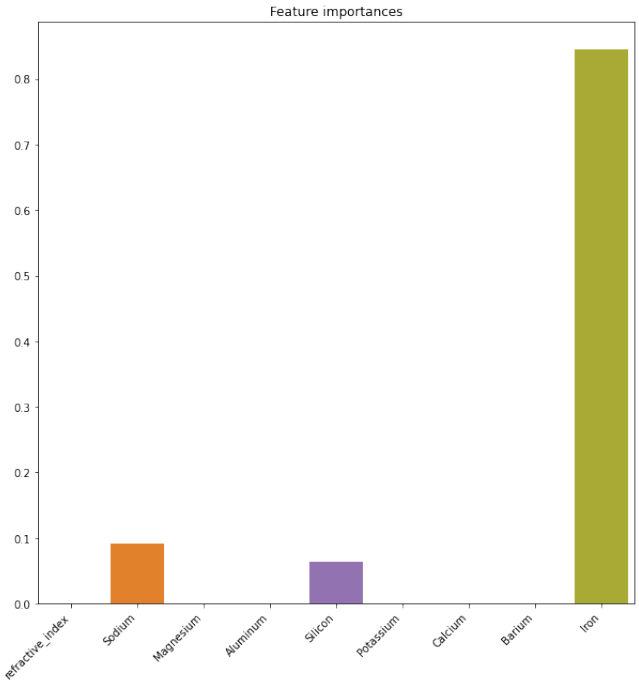

Figure 15: Features importances decision tree

only one of those m predictors (usually m $\approx \sqrt{p}$). The main difference between bagging and random forest is the choice of predictor subsets of size m. Practically, the sub-sample is controlled by max sample hyperparameter by setting bootstrap to true. Like in decision trees, the criteria and the depth are tuned: Gini index results to be the best criteria and the depth reaches a higher score with value 7. In addition to that, also the number of trees is regulated through k-fold cross validation (in Scikit learn [2] it is recognized as n-estimator). 50 was the found number. The best set of values was applied on the test set ; in figure 16 confusion matrix is reported and both the accuracy score and f1 score touch 98 %. As histogram 17 shows, random forest, in this case, performs less than decision tree because it takes in account more attributes which have not a high importance. Despite of that, "Iron" holds a primary role in terms of importance.

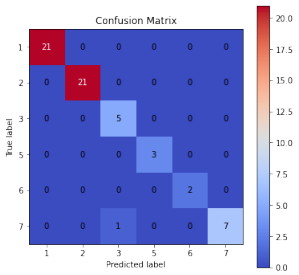

Figure 16: Confusion matrix random forest

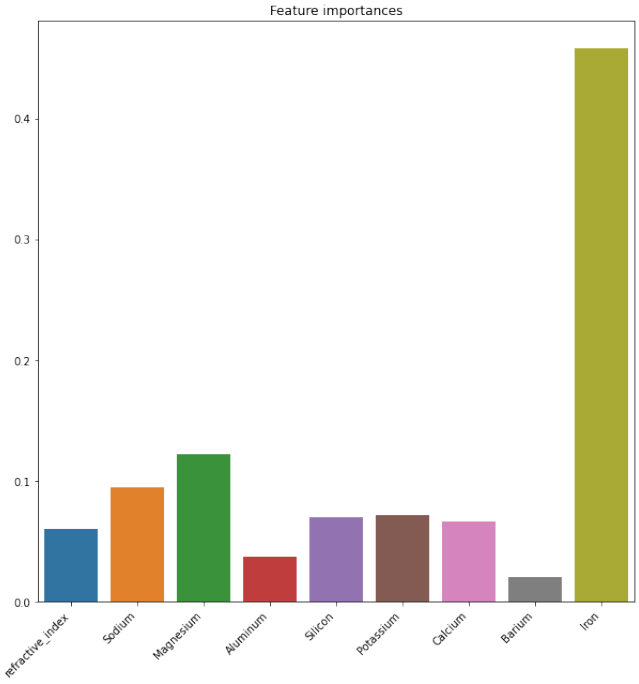

Figure 17: Feature importances random forest

4.3 K nearest neighbors classifier

The idea of k nearest neighbors classifier is to memorize the training dataset and then to predict the label of any new item on the basis of the labels of its closest neighbors in the training dataset. So, Knn finds a predefined number of training samples closest in distance to the new point and predict the label from these. The distance can be any metric measure, usually the choice falls on Euclidean distance. Mathematically, a metric function $\rho : XxX \longrightarrow R$ is a function which returns the distance between any two elements of X. By considering $X{=}R^d$, then ρ can be the Euclidian distance by following formula 11.

$$\rho(x, x') = |x - x'| = \sqrt{\sum_{i=1}^{d}(x_i - x'_i)^2} \tag{11}$$

This kind of procedure is simple to interpret, moreover it does not make any assumption, so it can be implemented in non-linear tasks. On the other side, it becomes very slow as the number of data points increases, because the model needs to store all these elements. In addition it is also sensitive to outliers.
The number of training samples to consider is regulated by hyperparameter K. After cross validation was performed, the best value of K found is 1. So, a new item is classified in the same way as its

nearest neighbor. After the tuning phase, the model was applied on the test set. In graph 18 is reported the confusion matrix, accuracy score grows until 85% and f1 score touches 86 %.

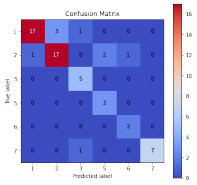

Figure 18: Confusion matrix for k nearest neighbors

4.4 Support Vector Machines

Support vector machines algorithm pretends to label the samples by catching the maximal margin hyperplane able to separate the classes. More precisely, a margin is the smallest distance from the training observation to a given separating hyperplane. It reaches its maximum value when the hyperplane has the farthest minimum distance to the training observations. Then, there is the opportunity to classify a test observation, based on which side of the maximal margin hyperplane it lies. This is the approach of a maximal margin classifier; the results will be pleasing if the classifier, which has a large margin on the training dataset, has a large margin on the test dataset. Mathematically, consider a feature space of dimension p, then $\beta_0, \beta_1, ..., \beta_p$ represent the coefficient of the maximal margin hyperplane, then the problem is to find a plane of equation 12, such that f(x) > 0 for one side of the hyperplane and f(x) < 0 for the other.

$$f(x) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p \tag{12}$$

Moreover, given a set of n training observations $X_1, X_2, ..., X_n \in R^p$ and associated to them class labels $y_1, y_2, ..., y_n \in \{-1, 1\}$, the maximal margin hyperplane results from the optimal solution of the following problem .

$$\begin{aligned} &\max_{\beta_0,\beta_1,....,\beta_p,M} M \\ &\sum_{j=1}^{p} B_j^2 = 1 \\ y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2}, ..., \beta_p X_{ip}) &\geq M \\ \forall i = 1, ..., n& \end{aligned}$$

The constraint guarantees that each observation will be on the correct side of the hyperplane. Actually, a separating hyperplane often does not exist and so the optimization problem has no solution with M > 0. A way to overcome this issue is the employment of a soft-margin. This approach allows some observations to be placed on the incorrect side of the margin or of the hyperplane (for that reason the adjective "soft"). Thanks to that, the robustness becomes greater and so the risk of

overfitting decreases. In mathematical translation, the problem turns into the following optimization problem:

$$\max_{\beta_0,\beta_1,....,\beta_p,\xi_1,...,_n,M} M$$
$$\sum_{j=1}^{p} B_j^2 = 1$$
$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2}, ..., \beta_p X_{ip}) \geq M(1 - \xi_i)$$
$$\xi_i \geq 0 \sum_{i=1}^{n} X_i \leq C$$

In the previous equation, C represents a non-negative parameter to tune. Moreover, $\xi_1, ..., \xi_n$ are slack variables having the task to allow individual observations to be on the wrong side of the hyperplane. If $\xi_i = 0$ the i_{th} observation is located in the correct side; if $\xi_i > 0$, then the i_{th} item is on the wrong side of the margin; finally if $\xi_i > 1$ sample is on the wrong side of the hyperplane. In addition, the optimization problem described before has an interesting behaviour, which is to be influenced only by samples lying on the margin or by items violating the margin. All these are defined as support vectors. In other words, support vector classifier is not affected by elements correctly labelled. Until now, only a linear boundary between two classes was analyzed, but in many applications non-linear boundaries might be considered. To face this problem, the feature space can be enlarged by using polynomial functions of the predictors. The support vector machines (SVM) represents the extension of the support vector classifier and it results from the feature space's enlargement, already described, by the exploitation of kernels. Intuitively, a kernel must be an entity favoriting the accomodation of a non-linear boundaries between classes. In the case study there are seven classes and SVM approach was to compare a K label with the remaining K$-$1. Firstly, equation 13 defines the inner product of two observations X_i, X_i'.

$$\langle x_i x_i' \rangle = \sum_{j=1}^{p} x_{ij} x_{ij}' \tag{13}$$

Then, the linear support vector machine is given by formula 14.

$$f(x) = \beta_o + \sum_{i=1}^{n} \alpha_i \langle x_i x_i' \rangle \tag{14}$$

In 14 the α_i are one for each training observations. In order to give an estimate of $\alpha_1, ...\alpha_n$ and β_0, it is requested the $\binom{n}{2}$ inner products $\langle x_i x_i' \rangle$ between all pairs of training observations. It turns out that $\alpha_i \neq 0$ only for the support vectors in the solution. By specifying S as the collection of indices of these support points, the equation 14 can be turned into relation 15.

$$f(x) = \beta_o + \sum_{i \in S} \alpha_i \langle x_i x_i' \rangle \tag{15}$$

Now, inner products can be generalized following formula $K(x_i, x_i')$, where K is some function recognized as a kernel. The kernel manages to establish the similarities between two observations. By

testing $K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{ij}'$, the support vector classifier is linear. Another option might be the replacement described in 16, originating a polynomial kernel of degree d, where d is a positive integer.

$$K(x_i, x_i') = (1 + \sum_{j=1}^{p} x_{ij} x_{ij}')^d \tag{16}$$

The last option considered and studied also in "glass identification" is the radial kernel, which has the form reported in formula 17, where γ is a positive constant.

$$K(x_i, x_i') = \exp^{-\gamma \sum_{j=1}^{p} (x_{ij} x_{ij}')^2} \tag{17}$$

In view to give a general form to write an optimization problem for support vector machine, equation 18 comes to rescue. It is sufficient to change the kernel K depending on the evenience.

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x_i, x_i') \tag{18}$$

In this case study, a linear kernel and a radial kernel were considered. When radial kernel is chose, two hyperparameters were subjected to tuning. The first one was C, whose aim is to give a bound to the sum of ξ_i and, in that way, to determine the severity of the violations to the margin and to the hyperplane that will be tolerated. As the penalty C increases, the tolerance towards margin grows. K-fold cross validation attests that the best value was 100. The last element to tune was γ, which appeared in 17. Intuitively, γ is the measure of how far the influence of a single training sample reaches. It can be seen as the inverse of the radius of influence of samples selected by the model as support vector. After K-fold cross validation, its best performing value was 0.01. At the end, a support vector machine model, with radial kernel and the already described hyperparameters, is applied on the test, both the accuracy and f1 score reach 98.3%. The confusion matrix is portrayed in image 19. Linear kernel has only C as a hyperparameter to tune, the value catched was 1. The performance on the test set was lower, in fact accuracy arrives to 81%, while f1 score touches 79%. The confusion matrix is portrayed in image 20.

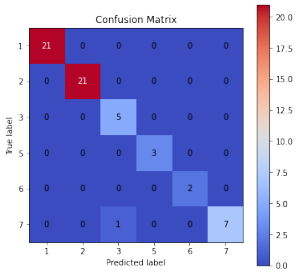

Figure 19: Confusion matrix for radial Support Vector Machines

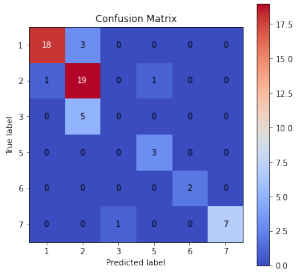

Figure 20: Confusion matrix for linear Support Vector Machines

5 Conclusions

As image 21 infers, the higher result was reached with decision tree classifier (100%). As already told, there was a feature more important respect to the others and it was crucial for the classification. In general, the scores were higher, this is probably due to class well separated,to the completeness of the studied dataset and also to a good performing validation of the proposed models.

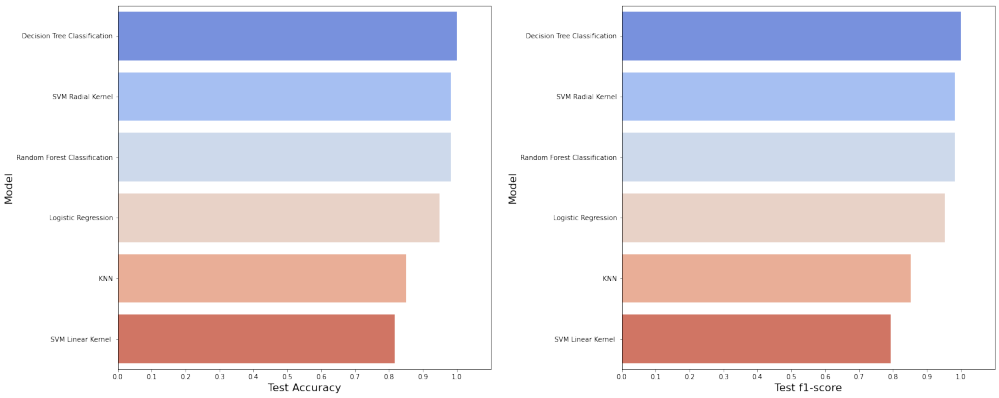

Figure 21: Algorithm comparison

References

[1] *Glass identification dataset available at.* URL: https://archive.ics.uci.edu/ml/datasets/glass+identification.

[2] *Scikit-learn library available at.* URL: https://scikit-learn.org/stable/.

[3] *Seaborn documentation distplot.* URL: https://seaborn.pydata.org/generated/seaborn.distplot.html.

[4] *Seaborn documentation heatmap.* URL: https://seaborn.pydata.org/generated/seaborn.heatmap.html.

[5] *Seaborn documentation violin-plot.* URL: https://seaborn.pydata.org/generated/seaborn.violinplot.html.

[6] Shai Shalev-Shwartz Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University, 2014.