POLITECNICO DI TORINO

Mathematics in Machine Learning course

Tesina

Analysis of ML techniques to predict drug consumption from personality traits

Professors
prof. Francesco Vaccarino

Correlatore:
prof. Mauro Gasparini

Student
Tommaso Monopoli

July 2022

Contents

Chapter 1

Dataset analysis

In this section, I will make a brief overview of the drug consumption dataset used in the experiments and perform an exploratory data analysis (EDA) of it.

1.1 Dataset overview

The **drug consumption dataset**[1] is a dataset containing data about 1885 individuals and the frequency with which they consume central nervous system psychoactive drugs. This dataset therefore contains valuable information for the problem of evaluating an individual's risk of drug consumption and misuse.

The data was collected by means of an online survey between 2011 and 2012. There were 1885 respondents to the survey. Respondents were asked to give some demographic/biographic information (age range, gender, ethnicity, country of residence, education level) and to take a personality test which was then used to measure personality traits (which are described in section 1.2). The questionnaire consists of 120 items; each item of the questionnaire is a statement which requires the respondent to either indicate "true" or "false" or answer on a Likert scale.

On top of that, each respondent was asked about his/her frequency of consumption of 18 different (legal or illegal) drugs, plus one fictious drug ("semeron") which was introduced to identify over-claimers.

[1] http://archive.ics.uci.edu/ml/datasets/Drug+consumption+%28quantified%29

1.2 Exploratory Data Analysis

As anticipated, the dataset comprises 1885 samples, each of which is associated with a distinct participant of the survey. Each individual is described by a total of 32 features:

1. **ID** (nominal categorical): unique ID associated to each participant
2. **Age** (ordinal categorical): age range of the participant; possible values: '18-24', '25-34', '35-44', '45-54', '55-64', '65+'
3. **Gender** (nominal categorical): 'male' or 'female'
4. **Education** (ordinal categorical): the education level of the participant; possible values: 'Left school before 16 years old', 'Left school at 16 years old', 'Left school at 17 years old', 'Left school at 18 years old', 'Attended some college or university, but no certificate or degree', 'Professional certificate/diploma', 'Bachelor's degree', 'Master's degree', 'Doctorate degree'
5. **Country** (nominal categorical): country of residence of the participant; possible values: 'Australia', 'Canada', 'New Zealand', 'Republic of Ireland', 'United Kingdom', 'USA', 'Other'
6. **Ethnicity** (nominal categorical): ethnic origin of the participant; possible values: 'Asian', 'Black', 'White, 'Mixed black/asian', 'Mixed white/asian', 'Mixed white/black', 'Other'
7. **N__score** (discrete numerical): neuroticism, a long-term tendency to experience negative emotions such as nervousness, tension, anxiety and depression
8. **E__score** (discrete numerical): extraversion, manifested in outgoing, warm, active, assertive, talkative, cheerful, and in search of stimulation characteristics
9. **O__score** (discrete numerical): openness to experience, a general appreciation for art, unusual ideas, and imaginative, creative, unconventional, and wide interests
10. **A__score** (discrete numerical): agreeableness, a dimension of interpersonal relations, characterized by altruism, trust, modesty, kindness, compassion and cooperativeness

11. **C__score** (discrete numerical): conscientiousness, a tendency to be organized and dependable, strong-willed, persistent, reliable, and efficient
12. **I__score** (discrete numerical): impulsiveness, a tendency to act without thinking, have poor concentration and thought intrusions, have a lack of consideration for consequences of own actions
13. **S__score** (discrete numerical): sensation-seeking, the seeking of varied, novel, complex, and intense sensations and experiences, and the willingness to take physical, social, legal, and financial risks for the sake of such experience
14. – 32. are ordinal categorical features about the **drug consumption frequency** of 19 different drugs.

    Drugs (in alphabetical order): alcohol, amphetamines, amyl nitrite, benzodiazepines, cannabis, chocolate, cocaine, caffeine, crack, ecstasy, heroin, ketamine, legal highs, LSD, methadone, magic mushrooms, nicotine, semeron, volatile substance abuse (VSA).

    Possible values for each drug: 'Never used', 'Used over a decade ago', 'Used in last decade', 'Used in last year', 'Used in last month', 'Used in last week', 'Used in last day'.

It is evident that data does not contain an explicit definition of 'users' and 'non-users' groups. There are several possible ways to discriminate participants into groups of users and non-users for **binary classification**; I decided to distinguish between **'illegal drug user'** (in short 'user') and **'not an illegal drug user'** (in short 'non-user'), by defining as 'user' *a participant who declared to have consumed any illegal drug in the past year*. Notice that the only legal drugs in this dataset are alcohol, caffeine, chocolate and nicotine.

1.2.1 Data cleaning

We can now take a closer look at the dataset and perform some necessary **data cleaning** steps. A few considerations:

- Not all the features characterizing this dataset are useful in the task of predicting whether a certain individual is a drug user or not. In this sense, we can safely **delete the 'ID' feature**. Moreover, I also decided to **delete the 'Country' and 'Ethnicity' features**, for a number of reasons:

  - we want to avoid that the prediction given by a classification model reflects/is influenced by the ethnic origin or country of residence of the individual (which are notoriously very sensitive attributes); ideally, we want the prediction to be based only on psychological traits and some minimal biographical information;
  - most machine learning models don't work well with categorical (and especially nominal) features; a possible solution is to apply one-hot encoding, but since ethnicity and country both have 7 possible values, this would result in 14 new features, thus doubling the dimensionality of our data (with the risk of incurring in the infamous *curse of dimensionality*);
  - most participants are from UK and USA (85% of the samples), and white (91% of the samples); we do not have enough data for other countries and ethnic groups.

- Participants who declared to have consumed the fictitious drug "semeron" are dropped from the dataset, as they are considered **overclaimers** and therefore the data associated with them may be unreliable.

- There are no **missing or invalid values** in the dataset, so no samples to drop nor imputation to be performed.

- Ordinal categorical features 'Age' and 'Education' are encoded into integers in such a way that the order between values is preserved (**ordinal encoding**). For instance, Age values are encoded as follows: '18-24' $\rightarrow$ 0, '25-34' $\rightarrow$ 1, . . . , '65+' $\rightarrow$ 5;

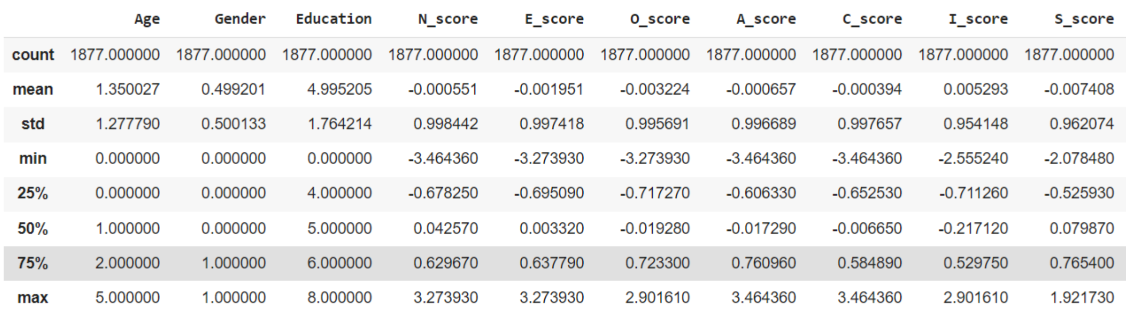
- 'Gender' values are encoded as follows: 'male' $\rightarrow$ 0, 'female' $\rightarrow$ 1

In fig. 1.1, the cleaned dataset is summarized by means of some statistical descriptors.

It is clear that features are not normalized, since they all have different means and standard deviations. Since most machine learning algorithms work better with (or even require) normalized data, we will standardize each feature as a data preprocessing step (see sect. 2).

1.2.2 Predictors and label distributions

The dataset is not balanced (fig. 1.2): in fact, 706 samples are labeled as 'Non user' (37.6% of the total) and 1171 as 'User' (62.4% of the total). Most



	Age	Gender	Education	N_score	E_score	O_score	A_score	C_score	I_score	S_score
count	1877.000000	1877.000000	1877.000000	1877.000000	1877.000000	1877.000000	1877.000000	1877.000000	1877.000000	1877.000000
mean	1.350027	0.499201	4.995205	-0.000551	-0.001951	-0.003224	-0.000657	-0.000394	0.005293	-0.007408
std	1.277790	0.500133	1.764214	0.998442	0.997418	0.995691	0.996689	0.997657	0.954148	0.962074
min	0.000000	0.000000	0.000000	-3.464360	-3.273930	-3.273930	-3.464360	-3.464360	-2.555240	-2.078480
25%	0.000000	0.000000	4.000000	-0.678250	-0.695090	-0.717270	-0.606330	-0.652530	-0.711260	-0.525930
50%	1.000000	0.000000	5.000000	0.042570	0.003320	-0.019280	-0.017290	-0.006650	-0.217120	0.079870
75%	2.000000	1.000000	6.000000	0.629670	0.637790	0.723300	0.760960	0.584890	0.529750	0.765400
max	5.000000	1.000000	8.000000	3.273930	3.273930	2.901610	3.464360	3.464360	2.901610	1.921730

Figure 1.1: Statistical descriptors of the drug consumption dataset (after data cleaning)

classification algorithms are negatively affected by imbalance in training data, because typically it will cause a classifier to have a bias towards the majority class: we will address this issue in sect. 2.3.

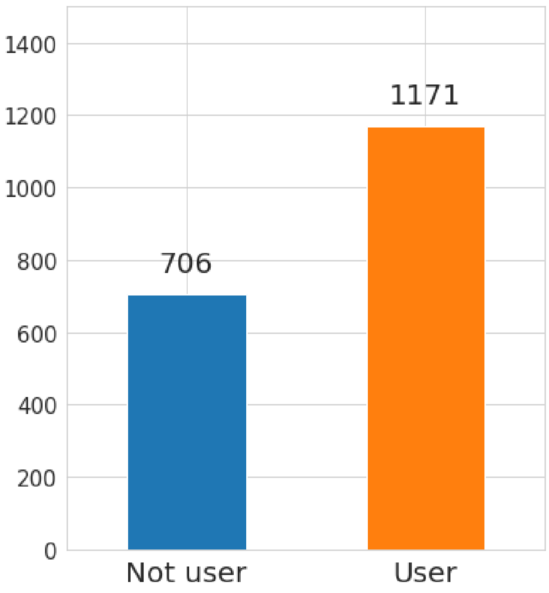


Figure 1.2: Label distribution

It is interesting to visualize the distribution of each predictor, separately for each of the two classes. In fig. 1.3a the distribution of the categorical features Age, Gender and Education is depicted as a **count plot** (a particular histogram well suited for categorical data).

Because of the class imbalance, it is better to visualize the distribution of the categorical features not with respect to their absolute frequency (i.e.

counts) but instead their **relative** one (fig. 1.3b). In this way, in spite of the class imbalance, the distributions can be visually compared.

By looking at these figures, some relevant facts can be inferred:

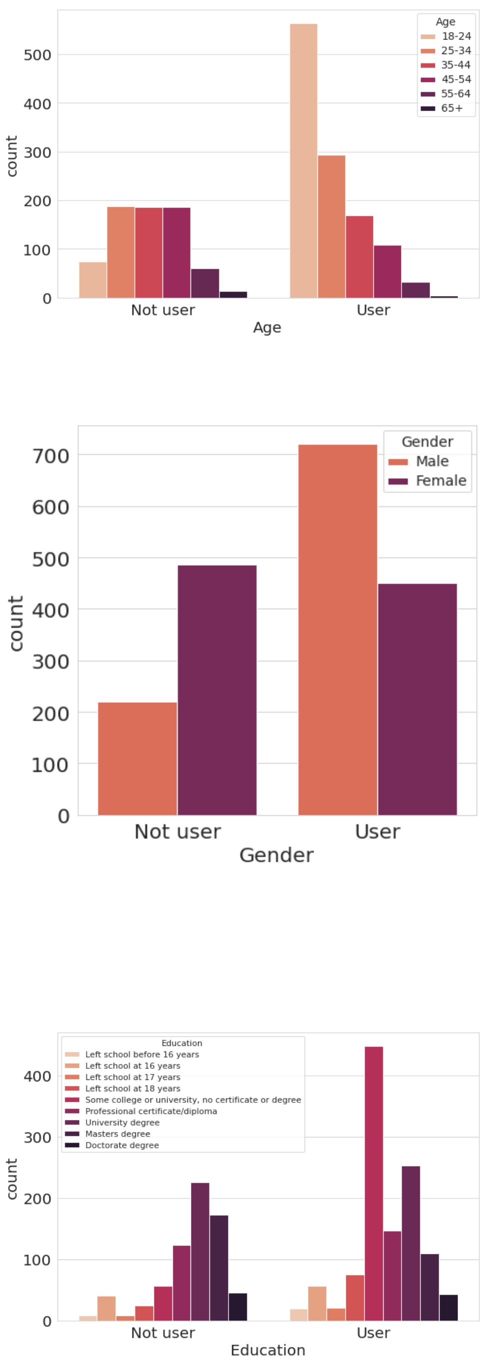
- Most drug users in the dataset are young or very young (18-24 are almost 50% of all users), whereas consumption becomes more unlikely as the person ages; we expect a (weak) negative correlation between Age and the class label.
- over 60% of the users in the dataset are male, whereas only the 30% of non-users are male; since male = 0 and female = 1, we expect a (weak) negative correlation between Gender and the class label.
- most users in the dataset attended some college or university, but didn't graduate (around 38% of the users), whereas most non-users hold at least a professional certificate or diploma; we expect a (weak) negative correlation between Education and the class label.

Overall, it seems that these "biographical" features could play a relevant role in predicting the drug consumption class of a respondent, since they appear to have quite different distributions for 'Not user' and 'User'. However, these are sensitive predictors: a classification model trained only on these three features would be very likely to predict as User a young male who dropped out of college.

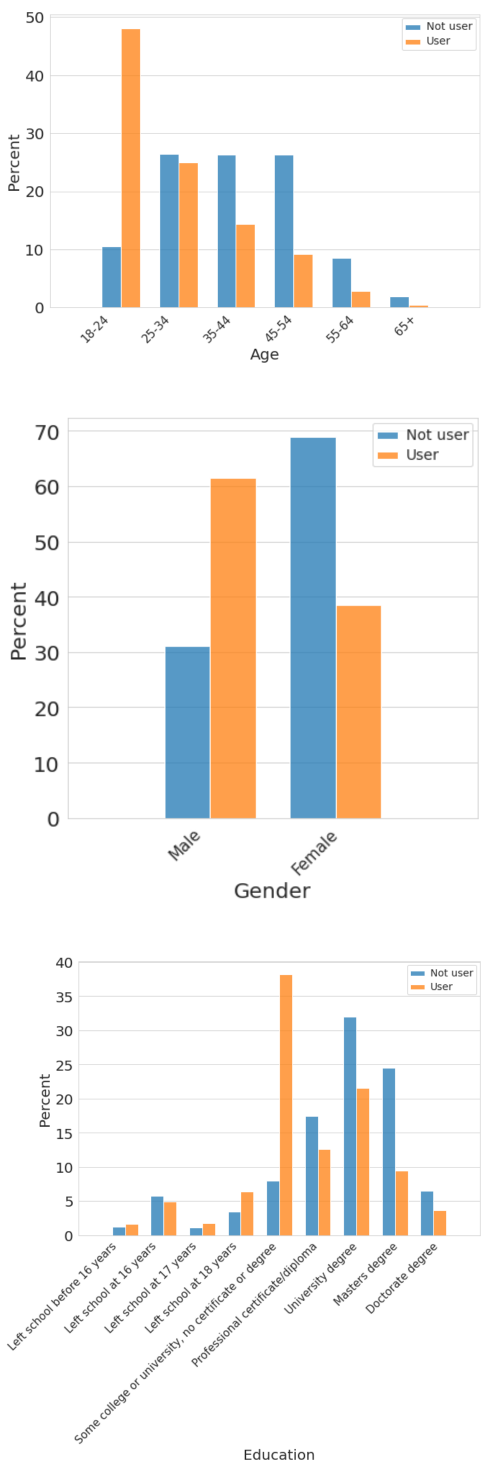
To address this discriminatory bias, we could decide to drop these features (as done with Country and Ethnicity) and rely only on personality trait features for training a classification model. However, I consider this issue to be out of the scope of this work, and have decided to keep these features.

In fig. 1.4 the distribution of the remaining numerical features is depicted as a box plot.

A **box plot** is a graph which gives a rough idea of the distribution of numerical data, through their quartiles. The first (Q_1) and third quartile (Q_3) graphically locate the lower and upper side of a "box", which is further cut by a line located at the median (second quartile, Q_2) of the data. The interquartile range is defined as IQR = $Q_3 - Q_1$, and can be used to identify outliers. The minimum and maximum data point - excluding any outliers - locate the "whiskers" of the box plot. Outliers may be defined as those points which are at a distance of at least $1.5 \times$ IQR above Q_3 or below Q_1, and are plotted as individual points beyond the whiskers.



(a)



(b)

Figure 1.3: Count plots (left) and relative frequency histograms (right) of the 'Age', 'Gender', 'Education' categorical features

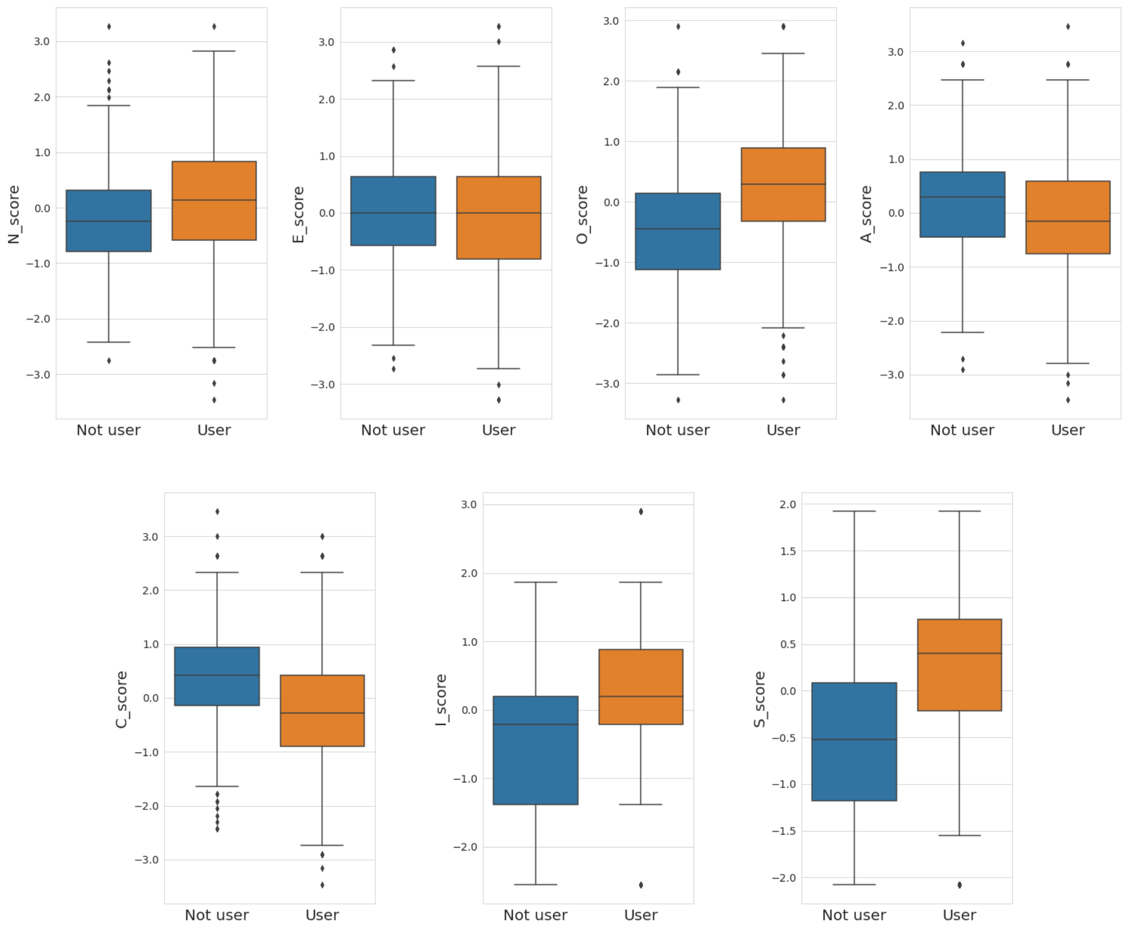


Figure 1.4: Box plots of the numerical features (personality trait measurements)

By looking at the figure, we can notice the existence of a few **outliers** (according to the definition given above), which are not likely to be measurement errors, and are not too "extreme": for these reasons, we can safely keep them.

By looking at these box plots, it is immediate to see that there exists a mild but clear correlation between these personality traits and the class label. The fact that these personality traits have very different distributions with respect to the class label clearly indicate that they can indeed be exploited to predict drug consumption. In particular, I_score and S_score seem to be the most (positively) correlated with the class label.

1.2.3 Correlation analysis

Lastly, we can validate our inferences about the correlation between the predictors and the class label by explicitly computing Pearson's correlation coefficient for each pair (predictor, predictor) and (predictor, target variable).

Pearson's correlation coefficient is a measure of linear correlation between two random variables. It is the ratio between the covariance of the two variables and the product of their standard deviations; thus it is essentially a normalized measurement of the covariance, such that the result always has a value between -1 and 1.

When working with samples instead of the entire statistical population (as it is our case), the sample Pearson's correlation coefficient between two variables x and y, r_{xy}, can be computed:

$$r_{xy} = \frac{\widehat{\text{cov}}(x,y)}{\hat{\sigma}_x \hat{\sigma}_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\widehat{\text{cov}}(x,y)$ indicates the sample covariance, $\hat{\sigma}_x$ the sample standard deviation, $\bar{x}$ the sample mean, n the sample size.

A positive correlation ($0 < r_{xy} \leq 1$) means that when the value of one variable increases, the other one tends to increase too (the closer to 1, the more linearly); viceversa for a negative correlation ($-1 \leq r_{xy} < 0$). An absolute value of exactly 1 implies that a linear equation $y = ax + b$ describes the relationship between x and y perfectly, with all the pairs (x, y) lying on a line. A value of 0 implies that there is no linear dependency between the variables. Fig. 1.5 summarizes these facts.

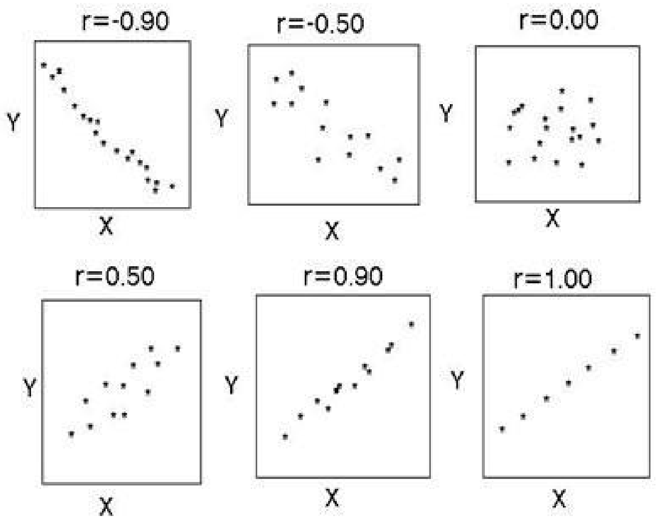


Figure 1.5: How two correlated variables behave, visually explained

Note that PCC can only reflect a linear correlation between variables, and ignores many other types of relationship or correlation (as a matter of fact, if two variables are linearly uncorrelated, i.e. $r_{xy} = 0$, they are not necessarily independent).

In fig. 1.6, the correlation coefficients between the variables of our dataset are reported (as a heat map).

The correlation analysis for our dataset confirms the inferences that we have made just by looking at the histograms and box plots of the variables.

In order to reduce the dimensionality of the dataset, one may drop those variables which are strongly correlated with other variables, because we can assume that they are redundant and therefore bring no further useful information for the prediction task (**feature selection**). However, in our case linear correlations are not too extreme (the maximum coefficient is the one between I_score and S_score, which is equal to 0.62). Therefore, we keep all the current features.

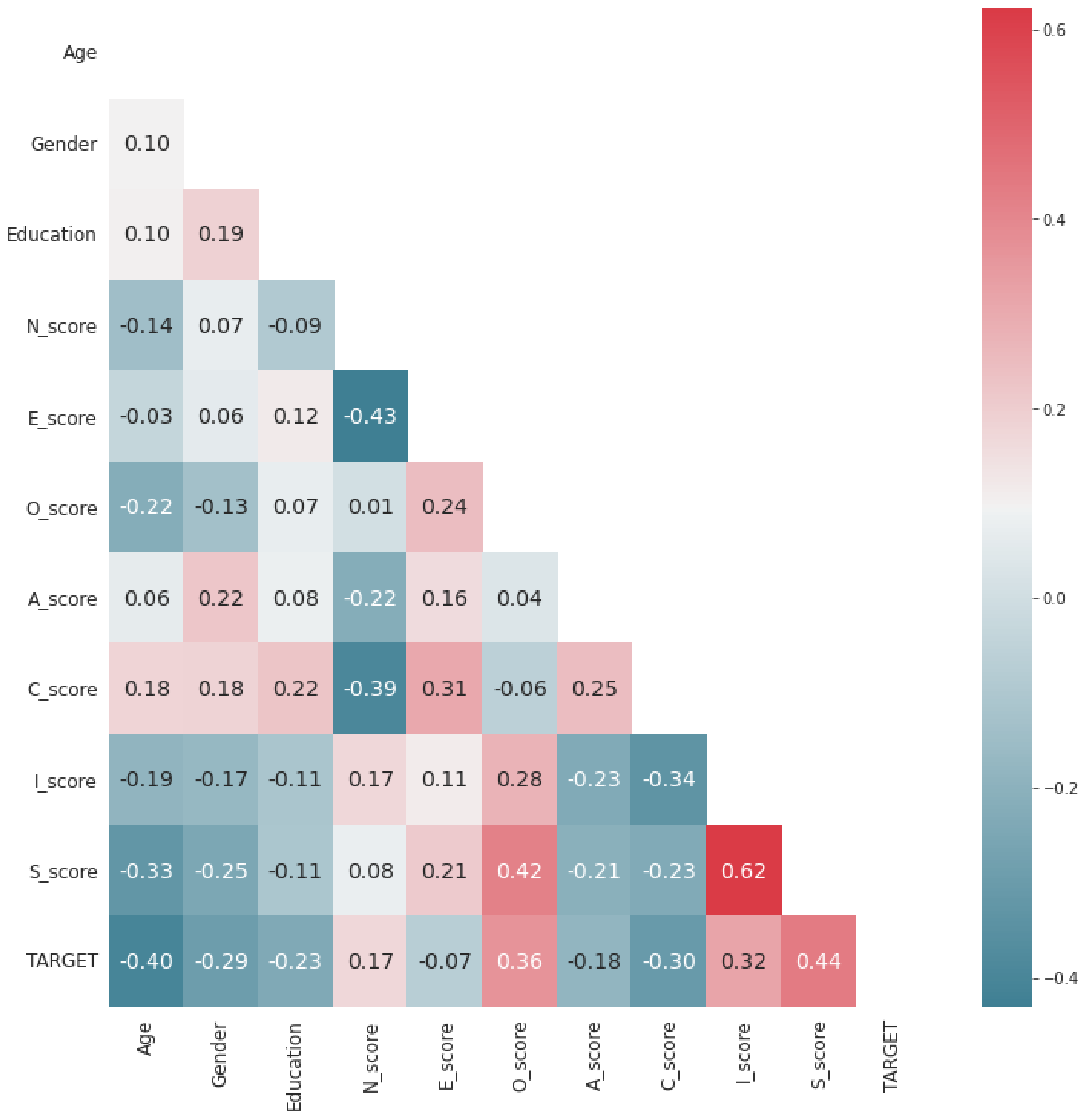


Figure 1.6: Correlation analysis for the dataset

Chapter 2

Preprocessing

Before training/fitting classification models for the task of predicting whether a respondent is a drug user or not, it is better to preprocess the data in a number of ways. In this section, different **preprocessing** techniques are reported and briefly explained, along with a motivation as to why and when to use them.

2.1 Standardization

Many machine learning algorithms don't work well with data which are at different scales; some, like PCA (sect. 2.2), even *require* that data are normalized. Since the features of our dataset have values that are at different scales (as seen in fig. 1.1), it is desirable to normalize them in some way.

Standardization is a normalization method which consists in subtracting from each feature its sample mean and dividing it by its sample standard deviation. For a feature x, with samples $x_1, \dots, x_n$, sample mean $\bar{x}$ and sample standard deviation $\hat{\sigma}_x$:

$$z_i = \frac{x_i - \bar{x}}{\hat{\sigma}_x}$$

Features normalized in this way will have the statistical properties of a standard Gaussian distribution (zero mean and unit standard deviation).

2.2 PCA

Since there are some mildly correlated predictors, but not so strong as to perform feature selection, it may still be desirable to perform some kind of

dimensionality reduction of our data, in such a way that the maximum possible variance is preserved.

Principal Components Analysis (**PCA**) is a method for performing a particular change of basis on the data, in which the axes of the new coordinate system are the set of orthonormal eigenvectors of the scatter matrix $A = X^T X$, commonly known as "principal components" of the dataset X. PCA is tipically exploited to perform (unsupervised) dimensionality reduction, in such a way to keep the maximum possible variance in the data.

The PCA problem can be expressed in many different ways, which can be proven to be equivalent. Two of them are reported hereafter. Given a dataset $X \in \mathbb{R}^{m \times d}$, with data points $\mathbf{x_1}, \dots, \mathbf{x_m}$, and assuming that data is zero-centered ($\frac{1}{m} \sum_{i=1}^{m} \mathbf{x_i} = 0$), the PCA problem is:

- find an "encoding" matrix $W \in \mathbb{R}^{n \times d}$ and a "decoding" matrix $U \in \mathbb{R}^{d \times n}$, with $n \leq d$, such that when the data points $\mathbf{x_i}$ are mapped to their lower-dimensional representation ($W\mathbf{x_i}$), and then mapped back to the original space ($UW\mathbf{x_i}$) the total reconstruction error is minimized

$$\operatorname*{arg\,min}_{U,W} \quad \sum_{i=1}^{m} \|\mathbf{x_i} - UW\mathbf{x_i}\|^2$$

- find a set of $n \leq d$ orthonormal directions $\mathbf{w_1}, \dots, \mathbf{w_n}$ by solving consecutively the following variance maximization problems:

$$\operatorname*{arg\,max}_{\mathbf{w_1}:\|\mathbf{w_1}\|=1} \quad \frac{1}{m}\sum_{i=1}^{m}(\langle \mathbf{w_1}, \mathbf{x_i}\rangle)^2$$

$$\operatorname*{arg\,max}_{\substack{\mathbf{w_k}:\|\mathbf{w_k}\|=1 \\ \mathbf{w_k}A\mathbf{w_j}=0,\, j=1,\dots,i-1}} \quad \frac{1}{m}\sum_{i=1}^{m}(\langle \mathbf{w_k}, \mathbf{x_i}\rangle)^2, \qquad k = 2, \dots, n$$

It can be proved that the first problem is solved by choosing U as the matrix whose columns are the (unit-norm) eigenvectors $\mathbf{u_1}, \dots, \mathbf{u_n}$ associated to the largest n eigenvalues of the scatter matrix A. In this case, the minimum of the objective function is $\sum_{i=n+1}^{d} \lambda_i$, the sum of the n smallest eigenvalues.

The second problem is solved similarly by choosing $\mathbf{w_1} = \mathbf{u_1}, \dots, \mathbf{w_n} = \mathbf{u_n}$, and in this case the maximum values of the objective functions of each problem are, in order, $\lambda_1, \dots, \lambda_n$.

This second formulation is of particular interest, because of the following interpretation: $\mathbf{w_1}$ is the direction of a line onto which the variance of the projected points, $\langle \mathbf{w_1}, \mathbf{x_i}\rangle$, is maximized; $\mathbf{w_2}$, is a second direction which maximizes the variance of $\langle \mathbf{w_2}, \mathbf{x_i} \rangle$ but which is also orthogonal to $\mathbf{w_1}$ (i.e., the projected points onto $\mathbf{w_1}$ and $\mathbf{w_2}$ are uncorrelated); and so on with the remaining maximization problems, which yield other directions of variance maximization which are orthogonal to all the previously found directions. This interpretation is particularly evident in fig. 2.1

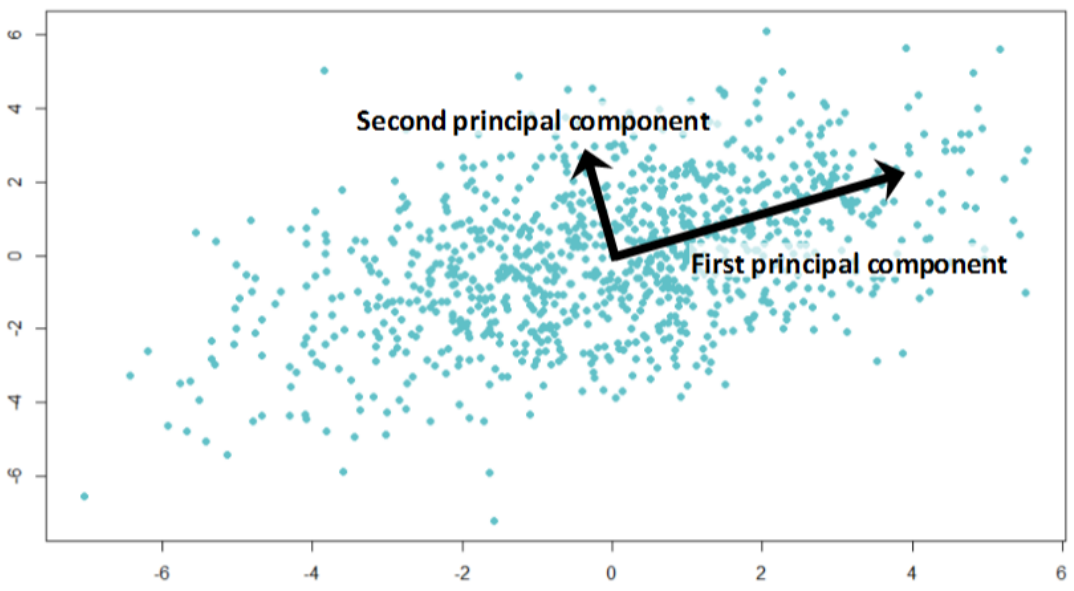


Figure 2.1: PCA on a simple 2D dataset

The proportion of the variance represented (or "*explained*") by each eigenvector can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues. This means that by reducing dimensionality with n principal components, the proportion of variance explained by these is $\sum_{i=1}^{n} \lambda_i / \sum_{i=1}^{d} \lambda_i$.

In this work, for all the experiments where PCA is used, we choose $n = 6$: based on the scree plot visualized in fig. 2.2, with $n = 6$ we are able to explain over 80% of the total variance of our dataset.

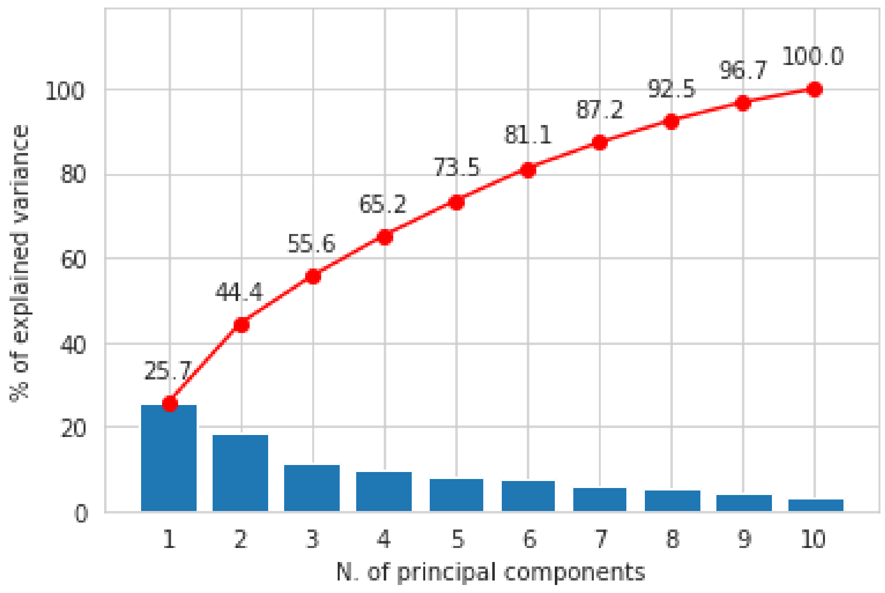


Figure 2.2: Scree plot associated to PCA on the drug consumption dataset

2.3 Rebalancing

As we displayed in fig. 1.2, the dataset is imbalanced. To solve the class imbalance, a number of **rebalancing** strategies can be employed:

- **Random oversampling**: the samples of the minority class (Not User) are randomly oversampled (without replacement), so to achieve balance between the two classes;
- **Random undersampling**: a random subset of samples from the majority class (User) is sampled (without replacement), so to achieve balance between the two classes;
- **SMOTE**: an oversampling technique; minority class is augmented with new synthetic points, computed from the original samples of the minority class

Each method has its pros and cons. While random oversampling increases the size of the dataset (which is desirable in order to have more samples on which the classifiers are trained on), it effectively just gives more "weight" to the oversampled samples of the minority class without creating new useful information, and thus increases the risk of overfitting. On the other hand, random undersampling discards potentially useful data, and this too can cause classification models to have a high generalization error (since there will be less samples to train on). The two methods are compared in fig. 2.3 for a generic 2D dataset.

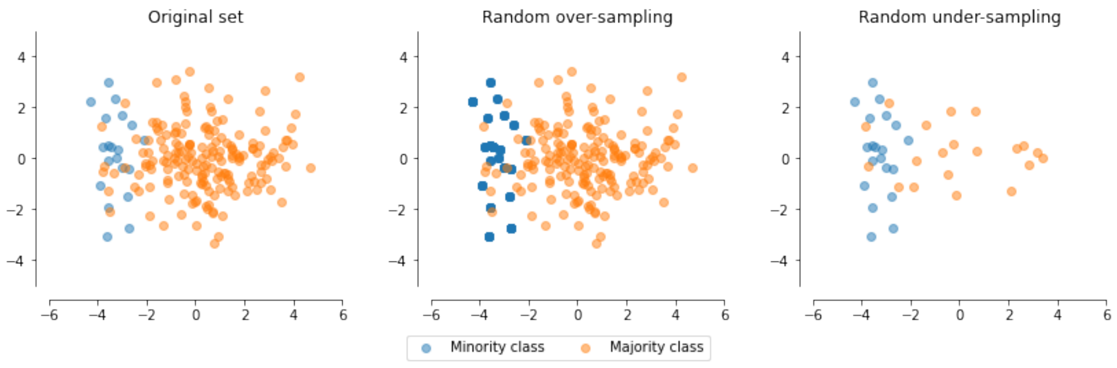


Figure 2.3: Application of random oversampling and random undersampling on a 2D dataset

Synthetic Minority Oversampling Technique (SMOTE) is an oversampling technique which tries to overcome both of these drawbacks: it doesn't lead to information loss, and the newly generated samples of the minority class are different from the original ones.

SMOTE first selects a minority class instance x_i at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors x_{neigh} at random and then computing a convex combination of the two chosen instances:

$$x_{new} = x_i + \lambda(x_{neigh} - x_i) \qquad \text{with } \lambda \in [0,1]$$

The resulting instance will lie on the (hyper-dimensional) segment connecting a and b, as shown in fig. 2.4. k is a user-defined hyperparameter; in this work it will be set to 5 for all experiments.

A variant of the SMOTE method is SMOTENC, which is more suited when also nominal categorical features are present: only $\lambda = 0$ or $\lambda = 1$ is chosen for these features, when generating the corresponding feature of x_{new}. In this work, SMOTENC is used in place of SMOTE when PCA is not applied.

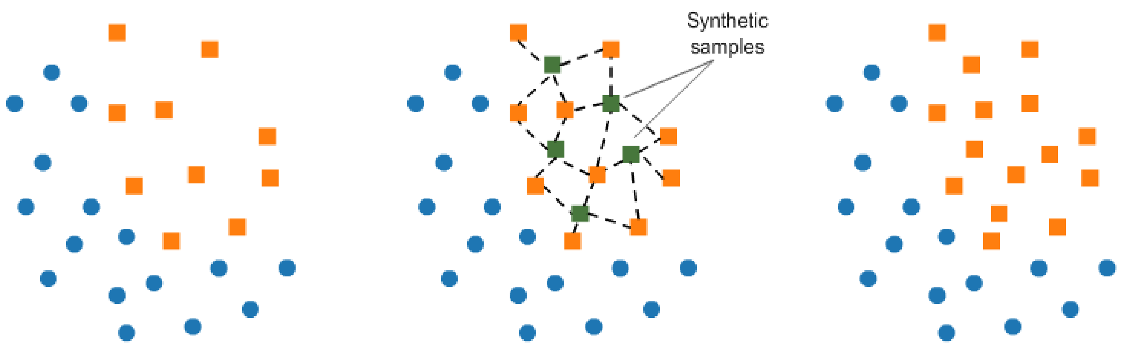


Figure 2.4: Application of SMOTE on a 2D dataset

2.4 Training and validation procedure

In this section, we delve into the methodology used to train, validate, test and compare the performances of classification models

2.4.1 Performance metrics

To evaluate the performances of a binary classification model, many different performance metrics can be used. Many of them stem from the same four different quantities, which are easily measured upon evaluating a model on a test set:

- True Positive (TP): n. of samples for which the prediction is positive (1), and the true label is positive (1)
- False Positive (FP): n. of samples for which the prediction is positive (1), but the true label is negative (0)
- True Negative (TN): n. of samples for which the prediction is negative (0), and the true label is negative (0)
- False Negative (FN): n. of samples for which the prediction is negative (0), but the true label is positive (1)

Clearly, these quantities sum up to the cardinality of the test set. In a binary classification setting, a confusion matrix is a 2×2 table which allows the visualization of the four quantities (fig. 2.5).

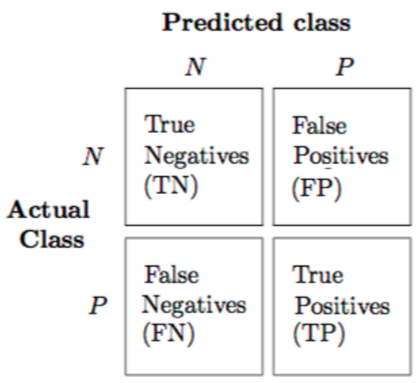


Figure 2.5: Confusion matrix

Using the values in the confusion matrix, a number of other metrics can be computed. Some examples:

- Accuracy: (TP+TN) / (TP+TN+FP+FN)
- Recall (or sensitivity, or true positive rate): TP / (TP+FN)
- Precision: TP / (TP+FP)
- Specificity (or true negative rate): TN / (TN+FP)
- F_1 score: $2 \times \text{precision} \times \text{recall}/(\text{precision} + \text{recall})$

For choosing which metrics to use to evaluate our drug consumption classification models, we have to make a few important considerations:

- if we use our model to decide whether to arrest a potential drug user or not, a false positive (i.e. non user misclassified as user) could lead to a wrongful arrest; in this case, we want to maximize true negatives while minimizing false positives.
- if we use our model to decide whether to admit a potential drug user to a rehab center or not, a false negative (i.e. user misclassified as non user) could mean preventing a drug user to seek rehabilitation; in this case, we want to maximize true positives while minimizing false negatives.

For these reasons, we decide that a good model is one which has both a high **sensitivity** (SEN) and a high **specificity** (SPC) on a test set of unseen data. When comparing two or more models, the best one is that which maximizes the quantity $\min\{\text{SEN}, \text{SPC}\}$.

2.4.2 Train-test splitting

In order to test the performance of the model on unseen data, so to evaluate its generalization error, the dataset is split in two subsets: a training set, containing 80% of the samples, and a test set, containing the remaining 20%. This splitting is performed in a **stratified** fashion, i.e. the two subsets are generated in such a way that the proportion of User vs Non User is the same as in the original dataset. The training set will be used for training the classification model, whereas the test set will be held out for the final evaluation of the model predictive performances.

2.4.3 K-fold cross-validation

To tune the hyperparameters of the model and to evaluate the performances of different predictive modeling procedures, **(stratified) K-fold cross validation** can be used. It consists in randomly splitting (in a stratified fashion) the training set into K subsets, called *folds*. One fold is held out as a validation set, while the remaining $K-1$ folds are used for training the model: each prediction pipeline is fitted and trained on the $K-1$ training folds, and tested on the remaining validation fold, on which one or more performance metrics are evaluated. This procedure is repeated K times, in order to use each fold as a validation set; therefore a total of K classifiers are trained. The resulting performance metrics are combined (e.g. averaged) over the K rounds to give an estimate of the model's predictive performance on unseen data.

In this work, we perform a **grid search** over different hyperparameters which characterize the classification models. For each set of hyperparameters, a 5-fold cross-validation is run; the set of hyperparameters which performs best, i.e. the one associated to the maximum $\min\{\text{SEN}, \text{SPC}\}$ (both computed in a **micro-average** way), is used to refit the classification pipeline on the whole training set, and this final pipeline is evaluated on the held-out test set. The cross-validation procedure is visually summarized in fig. 2.6.

For each classification model, we experiment with different preprocessing pipelines: standardization (always performed) $\longrightarrow$ PCA (optional) $\longrightarrow$ random oversampling/random undersampling/SMOTE (optional). This implies a total of $1 \times 2 \times 4 = 8$ possible preprocessing pipelines for our data.

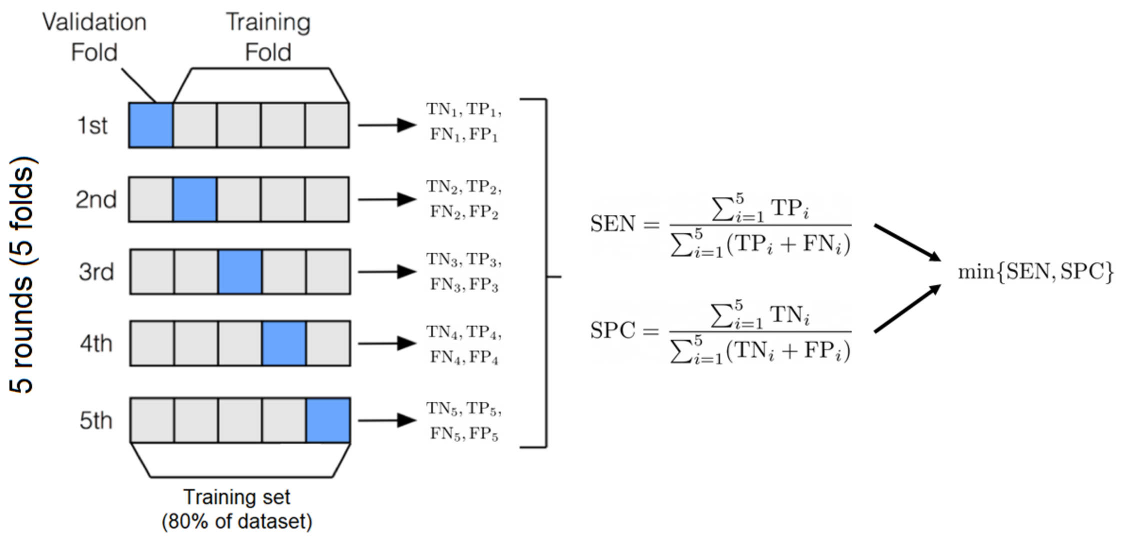

Validation Fold
Training Fold
5 rounds (5 folds)
1st
2nd
3rd
4th
5th
$\mathrm{TN}_1, \mathrm{TP}_1, \mathrm{FN}_1, \mathrm{FP}_1$
$\mathrm{TN}_2, \mathrm{TP}_2, \mathrm{FN}_2, \mathrm{FP}_2$
$\mathrm{TN}_3, \mathrm{TP}_3, \mathrm{FN}_3, \mathrm{FP}_3$
$\mathrm{TN}_4, \mathrm{TP}_4, \mathrm{FN}_4, \mathrm{FP}_4$
$\mathrm{TN}_5, \mathrm{TP}_5, \mathrm{FN}_5, \mathrm{FP}_5$
$\mathrm{SEN} = \frac{\sum_{i=1}^{5} \mathrm{TP}_i}{\sum_{i=1}^{5}(\mathrm{TP}_i + \mathrm{FN}_i)}$
$\mathrm{SPC} = \frac{\sum_{i=1}^{5} \mathrm{TN}_i}{\sum_{i=1}^{5}(\mathrm{TN}_i + \mathrm{FP}_i)}$
$\min\{\mathrm{SEN}, \mathrm{SPC}\}$
Training set
(80% of dataset)

Figure 2.6: 5-fold cross-validation procedure adopted in this work

It is critical to notice that the preprocessing pipeline is directly integrated in the cross-validation procedure: the fitting of such preprocessing steps is performed within each CV round, so as to avoid **data leakage** from the validation fold to the training folds.

Chapter 3

Model selection

In this final section, we will choose different classification models, and see how they perform on the binary classification task at hand.

3.1 Decision Tree and Random Forest

Decision tree learning is a simple supervised learning approach that creates a decision tree to use as a classification model. A decision tree predicts the class associated with a sample $\mathbf{x}$ by traveling from its "root" node to one of its leaves. At each node on the root-to-leaf path, the successor node is chosen on the basis of a splitting of the input space. Usually, the splitting is a binary "test" on one of the features of $\mathbf{x}$. The sample will be assigned the class associated to the leaf reached.

In this way, a decision tree tesselates the input space with non-overlapping hyper-rectangles ("boxes") associated to a single class (the most commonly occurring one among the training samples in that region). An example is shown in fig. 3.1.

Although many different decision tree learning algorithms exist, most of them follow the same approach, known as recursive binary splitting. This approach is:

- top-down: the tree is built recursively starting from the root and going down to the leaves

- greedy: at each recursive step the local best split is performed, rather than a locally suboptimal one which might lead to the globally optimal tree

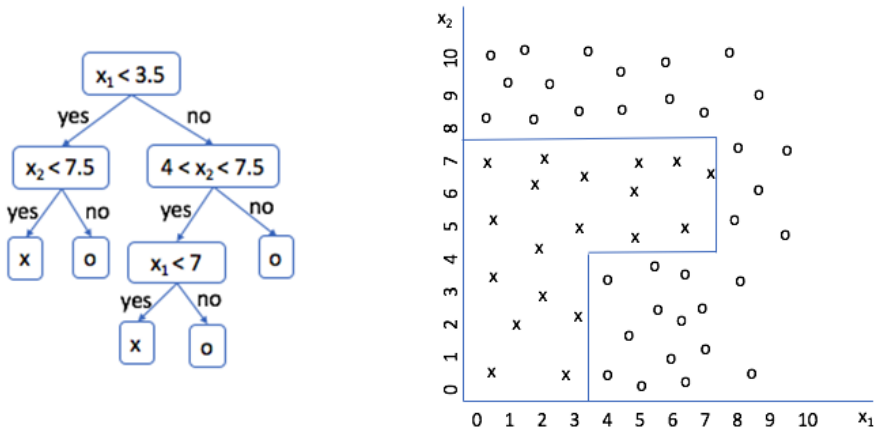


Figure 3.1: A decision tree trained on a 2D dataset (left), and corresponding decision boundaries (right)

Different learning algorithms use different metrics for defining the "best" predictor on which to split the training samples, and the associated test to perform on it. These metrics generally measure the homogeneity of the target variable within a set of samples (aka the "purity" of the nodes). Some of the most commonly used are:

- Gini index: $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
- Cross-entropy: $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

where K is the number of classes and $\hat{p}_{mk}$ is the proportion of samples in region m associated to class k.

This training process would continue to expand nodes by minimizing the chosen metric, until all leaves are pure. Clearly, this may lead to overfitting the training data. To prevent it, we may decide to build the tree until a stopping criterion is reached: for example, we could fix the maximum depth of the tree as a hyperparameter of the learning algorithm.

Decision trees are simple, highly interpretable, can be displayed graphically and can easily handle categorical predictors without the need to create dummy variables. However, they typically can't compete with other well-known classification approaches in terms of prediction accuracy. To reduce variance, prevent overfitting and hence increase performances, we may use **Random Forest**, an ensemble learning technique that consists in growing multiple trees and then combining them to produce a prediction based on majority voting. It is based on two statistical techniques:

- Bagging (bootstrap aggregating): train an ensemble of B models on B different training sets, each obtained by randomly sampling $m = |X|$ instances with replacement from the original training set X; used to reduce the variance of a learning algorithm.
- Feature bagging (random subspace method): at each split, the features on which to split are selected within a random subset of only $d' < d$ of the total set of d predictors; used to reduce the correlation between estimators in an ensemble

Combining many trees usually results in dramatic improvements in prediction accuracy, at the expense of some loss of interpretability. For this reason, in this work only random forest is used.

Results are reported in the following tables. The best performing configuration is highlighted in bold.

Table 3.1: Random Forest - grid search of hyperparameters

Hyperparameter	Values
Forest population	8, **40**, 120, 240, 360, 480
Split criterion	Gini index, **cross-entropy**
Maximum depth	no limit, **5**, 10, 20, 30, 50
Maximum features at each split	all, $\lceil\sqrt{\mathbf{d}}\rceil$
Bootstrap	yes, **no**

Table 3.2: Random Forest - results on test set

Pipeline	sensitivity	specificity	accuracy
Standardization	0.885	0.660	0.801
Std. + random oversampling	0.885	0.624	0.787
Std. + random undersampling	**0.783**	**0.773**	0.779
Std. + SMOTENC	0.872	0.702	0.809
Std. + PCA	0.851	0.631	0.769
Std. + PCA + random oversampling	0.834	0.631	0.758
Std. + PCA + random undersampling	0.766	0.787	0.774
Std. + PCA + SMOTE	0.826	0.674	0.769

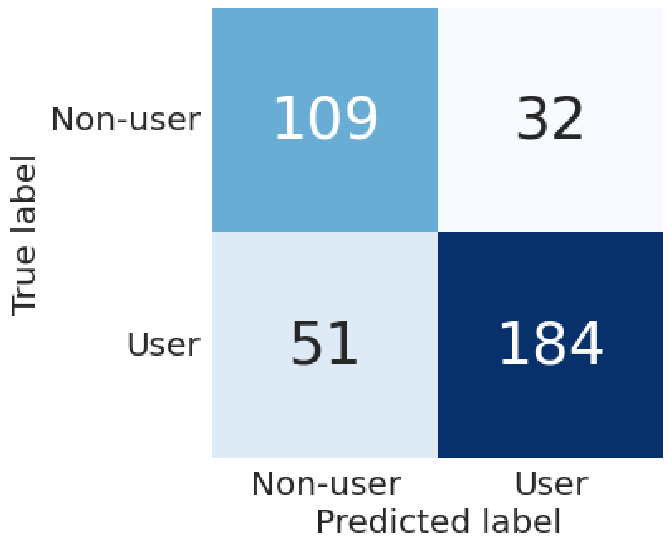


Figure 3.2: Random Forest - confusion matrix associated to the best performing configuration

3.2 SVM

Support-vector machine (SVM) is a supervised learning binary classification model. The original SVM learning problem consists in finding the hyperplane which best separates the two classes in the data space.

Let $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m)$ a d-dimensional dataset. If the dataset is linearly separable (i.e. there exists a hyperplane $\pi : \langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ which separates the two classes, in formula: $y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) > 0 \ \forall i$) then the problem is to find that separating hyperplane which maximizes the minimum distance between itself and the data points):

$$\begin{aligned} \max_{\mathbf{w}, b \,:\, \|w\|=1} \quad & \min \{ d(\pi, \mathbf{x_i}) \mid i = 1, \ldots, m \} \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) > 0 \qquad \forall i \end{aligned} \tag{3.1}$$

The "margin" of a separating hyperplane is the distance of the closest training points to it ($\min_i \frac{|\langle \mathbf{w}, \mathbf{x_i} \rangle + b|}{\|\mathbf{w}\|}$), so the best separating hyperplane is that which maximizes the margin. The training points lying on the margin are called the "support vectors", because it can be shown that in the dual of the Hard-SVM problem only the Lagrange multipliers associated to said vectors are non-zero. This means that among all the training samples, only support vectors contribute to fixing the best separating hyperplane. Fig. 3.3 shows all these concepts.

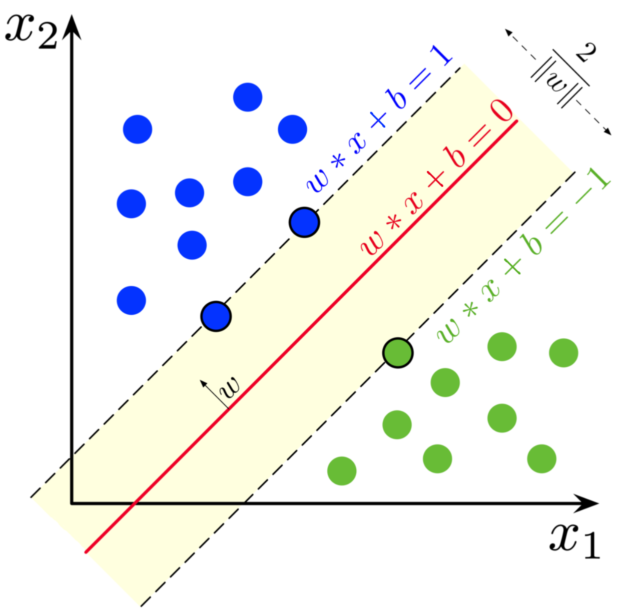


Figure 3.3: An SVM trained on a (linearly separable) 2D dataset

It can be proved that problem 3.1 can be rewritten as a quadratic programming problem (much easier to solve than a non-linear one):

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 \qquad \forall i \end{aligned} \tag{3.2}$$

This formulation is known as **Hard-SVM** learning algorithm. The resulting optimal hyperplane $(\mathbf{w}^*, b^*)$ is then normalized to $(\mathbf{w}^*/\|\mathbf{w}^*\|, b^*/\|\mathbf{w}^*\|)$, and this tuple solves the original problem 3.1

However, assuming that the dataset is linearly separable is a rather strong assumption. A natural relaxation is to allow the constraints to be violated for some of the examples in the training set. This can be modeled by introducing nonnegative slack variables:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \lambda \|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \qquad \forall i \end{aligned} \tag{3.3}$$

This formulation is known as **Soft-SVM** learning algorithm. The hyperparameter $\lambda > 0$ determines the trade-off between increasing the margin size and ensuring that the $\mathbf{x_i}$ lie on the correct side of the margin.

In *scikit-learn*, a hyperparameter C is used instead of λ, which weights the sum of the slack variables instead of the norm of $\mathbf{w}$ in the objective function: higher values of C imply a more accurate separation at the cost of a smaller margin, and viceversa.

C = 1/(2*lambda*m)

Sometimes data which is not linearly separable in the original space may be linearly separable when mapped to a higher dimensional feature space, defined by means of a mapping $\psi : \mathcal{X} \to \mathcal{F}$ (see fig. 3.4).

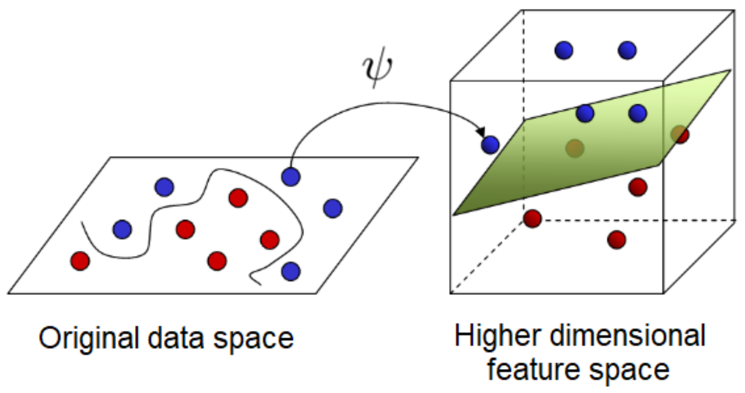


Figure 3.4: This 2D dataset is not linearly separable in $\mathbb{R}^2$, but it becomes so when mapped onto $\mathbb{R}^3$ by means of a mapping function ψ

It can be proved that the **dual Hard-SVM problem** can be written as:

$$\begin{aligned} \max_{\boldsymbol{\alpha}\in\mathbb{R}^m} \quad & \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle \right) \\ \text{s.t.} \quad & \boldsymbol{\alpha} \geq 0 \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned} \tag{3.4}$$

and that $\mathbf{w}$ is in the linear span of the training samples: $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i}$. As anticipated, it can be shown that the solution of the dual problem, $\boldsymbol{\alpha}^*$, has non-zero multipliers only for those $\mathbf{x_i}$ lying on the margin of the best separating hyperplane $\langle \mathbf{w}^*, \mathbf{x} \rangle + b = 0$, i.e. at a distance $1/\left\|\mathbf{w}^*\right\|$ from it.

Note that the dual problem only involves inner products between training samples. This property allows us to substitute the samples $\mathbf{x_i}$ with their higher dimensional mappings $\psi(\mathbf{x_i})$, so that the Hard-SVM problem can be solved in the new feature space $\mathcal{F}$. Note also that knowing the analytical expression of ψ is not really needed: we just need to know how to compute the inner product in $\mathcal{F}$. In other words, we need to know the kernel function associated to the feature space, $K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$. This is known as the **kernel trick**.

A commonly used kernel is the RBF (or Gaussian) kernel:

$$K(\mathbf{x}, \mathbf{x}') = e^{\frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$

Intuitively, the Gaussian kernel is close to 0 if the two points are far away from each other (in the original domain) and close to 1 if they are close. σ is a hyperparameter that controls the scale determining what we mean by "close".

In *scikit-learn*, a hyperparameter $\gamma = \frac{1}{2\sigma^2}$ is used instead of λ. High values of γ make the kernel close to 0 even when $\mathbf{x}$ and $\mathbf{x}'$ are very close; the effect is that meaningful differences in the kernel only take place very close to a single support vector: a single support vector almost entirely defines the decision boundary in its proximity, leading to a more complex model (i.e. a model with more support vectors), highly non-linear decision boundaries, low bias and high variance (and thus risk of overfitting). On the other hand, low values of γ have the opposite effect of reducing the number of support vectors and making decision boundaries more linear, implying high bias and low variance (and thus risk of poor classification accuracy).

At test time, a sample $\mathbf{x}$ is classified as -1 or 1 with the decision rule $\hat{y} = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$.

On a final note, it is also possible to kernelize the Soft-SVM problem, so that the dataset doesn't have to be linearly separable in the new, higher-dimensional feature space. The dual Soft-SVM problem is similar to the dual Hard-SVM problem, with the additional constraint that $\boldsymbol{\alpha} \leq \frac{1}{2m\lambda}$.

Results are reported in the following tables. The best performing configuration is highlighted in bold.

Table 3.3: SVM - grid search of hyperparameters

Hyperparameter	Values
Kernel	linear, **RBF**
C	0.001, 0.01, 0.1, **1**, 10, 100, 1000
γ (only if kernel is RBF)	(1/#features), 2e-9, 2e-7, 2e-5, 2e-3, 2e-1, **2**

Table 3.4: SVM - results on test set

Pipeline	sensitivity	specificity	accuracy
Standardization	0.911	0.546	0.774
Std. + random oversampling	**0.817**	**0.766**	0.798
Std. + random undersampling	0.804	0.766	0.790
Std. + SMOTENC	0.838	0.667	0.774
Std. + PCA	0.915	0.546	0.777
Std. + PCA + random oversampling	0.898	0.404	0.713
Std. + PCA + random undersampling	0.796	0.716	0.766
Std. + PCA + SMOTE	0.962	0.284	0.707

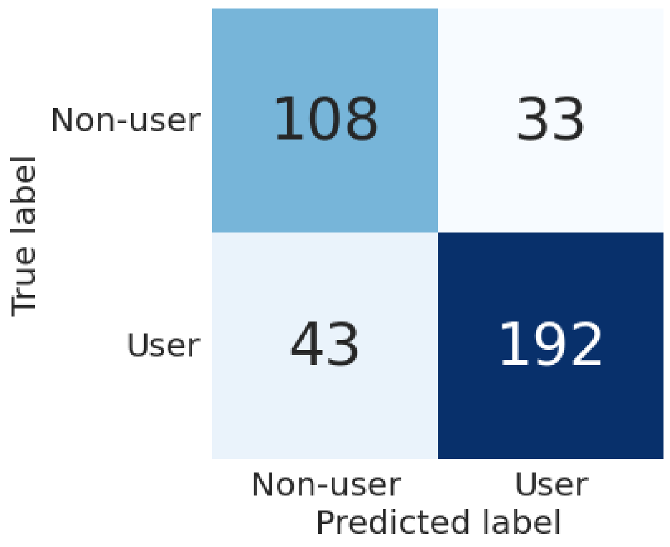


Figure 3.5: SVM - confusion matrix associated to the best performing configuration

3.3 KNN

K-nearest neighbors (KNN) is a simple classification algorithm. The class prediction of a sample is done by computing the distances between the sample and each point in the training set, then finding its K nearest neighbors and finally the predicted class is decided by majority voting between the K neighbors. The votes of each neighbor may be weighted by their distance from the sample to classify.

It is a non-parametric algorithm, in the sense that it does not make any assumption on the particular mathematical relationship between the predictors and the target variable (there are no parameters to learn): the training phase of the algorithm consists only of storing the training samples and corresponding class labels in memory.

K is an important hyperparameter of the method: when K is small, only a few neighbors contribute to the prediction, so decision boundaries of the method may look very sharp and noisy (high variance); larger values of K reduce the effect of noise on the classification, and make boundaries between classes smoother (low variance). This behavior is shown in fig. 3.6

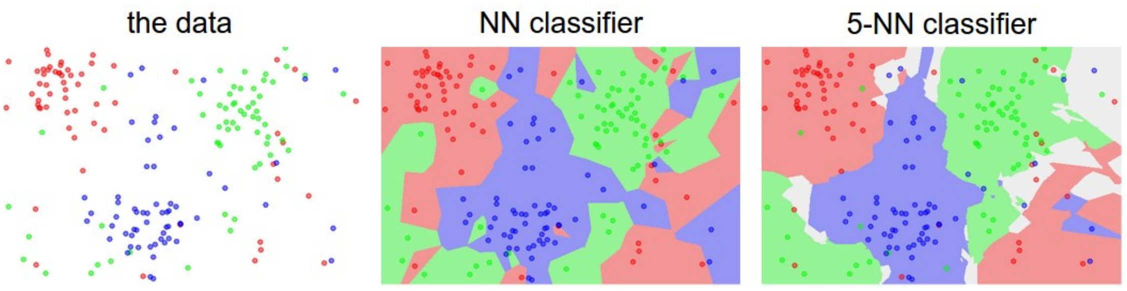


Figure 3.6: Decision boundaries of KNN with $K = 1$ and $K = 5$, fitted on a simple 2D dataset with 3 classes. Grey areas are indecision regions.

Results are reported in the following tables. The best performing configuration is highlighted in bold.

Table 3.5: KNN - grid search of hyperparameters

Hyperparameter	Values
K	1,3,5,7,10, **20**
Distance	Euclidean, **Manhattan**
Weights	uniform, **distance**

Table 3.6: KNN - results on test set

Pipeline	sensitivity	specificity	accuracy
Standardization	0.821	0.645	0.755
Std. + random oversampling	0.783	0.596	0.713
Std. + random undersampling	0.740	0.759	0.747
Std. + SMOTENC	0.753	0.752	0.753
Std. + PCA	0.843	0.624	0.761
Std. + PCA + random oversampling	0.740	0.780	0.755
Std. + PCA + random undersampling	0.740	0.773	0.753
Std. + PCA + SMOTE	**0.762**	**0.766**	0.763

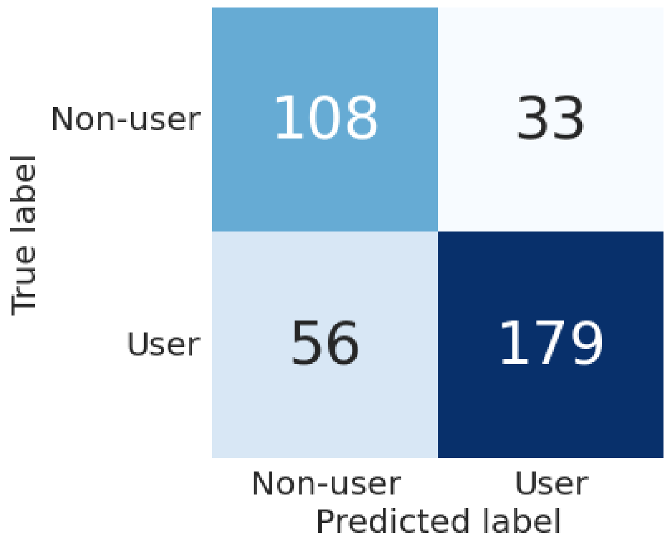


Figure 3.7: KNN - confusion matrix associated to the best performing configuration

3.4 Logistic Regression

Logistic regression is a particular form of regression analysis which is commonly used in classification problems. The (multivariable) logistic regression model is the following:

$$p(\mathbf{x}; \boldsymbol{\beta}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d)}}$$

It is easy to see that this function has values between 0 and 1, for any $\boldsymbol{\beta}$ and $\mathbf{x}$. Note that when the linear combination $\beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$ is very large (positive) then $p(\mathbf{x};\boldsymbol{\beta})$ is close to 1, whereas when it is very small (negative) then $p(\mathbf{x};\boldsymbol{\beta})$ is close to 0. Therefore, a logistic regression model can naturally be fitted to model the probability that a sample $\mathbf{x}$ belongs to a certain class $y \in [K]$.

We talk of binary logistic regression when the target variable is binary ($y \in \{0,1\}$), as in our case. In this setting, $p(\mathbf{x};\boldsymbol{\beta})$ models the probability that $\mathbf{x}$ belongs to class 1:

$$\begin{aligned} Pr(y = 1|\mathbf{x}) &= p(\mathbf{x};\boldsymbol{\beta}) \\ Pr(y = 0|\mathbf{x}) &= 1 - p(\mathbf{x};\boldsymbol{\beta}) \end{aligned} \tag{3.5}$$

or, in other words, this model assumes that $y|_{\mathbf{x}} \sim Bernoulli(p(\mathbf{x};\boldsymbol{\beta}))$, and so its probability mass function is $f_{y|\mathbf{x}}(y|\mathbf{x};\boldsymbol{\beta}) = p(\mathbf{x};\boldsymbol{\beta})^y(1 - p(\mathbf{x};\boldsymbol{\beta}))^{1-y}$. An example of a logistic model fitted to a single-predictor dataset with two classes is shown in fig. 3.8.

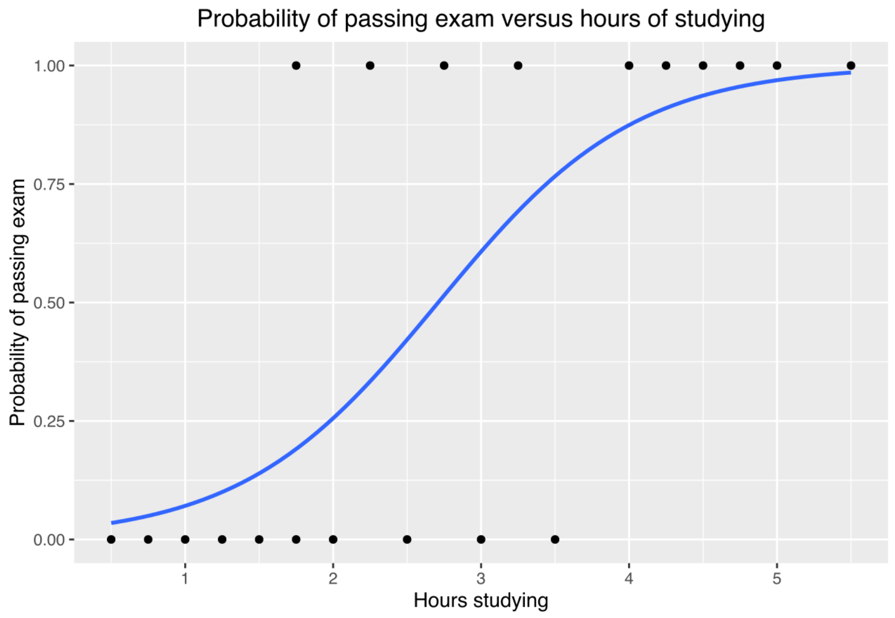


Figure 3.8: Logistic regression model fitted to a binary dataset with a single predictor.

The parameters of the logistic regression model are most commonly estimated via maximum likelihood estimation (MLE). Upon having observed a dataset $(\mathbf{x_1}, y_1), \dots, (\mathbf{x_m}, y_m)$, we want to find the parameters $\boldsymbol{\beta}$ which maximize the (conditional) log likelihood function:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\beta}; y_1|\mathbf{x_1}, \dots, y_m|\mathbf{x_m}) &= \log \prod_{i=1}^{m} f_{y_i|\mathbf{x_i}}(y_i|\mathbf{x_i}; \boldsymbol{\beta}) = \\ &= \sum_{i=1}^{m} y_i \log p_i + (1 - y_i) \log(1 - p_i)\end{aligned} \tag{3.6}$$

where for simplicity we have written $p(\mathbf{x_i}; \boldsymbol{\beta}) = p_i$.

Unfortunately, it is not possible to find a closed-form expression for the coefficients $\boldsymbol{\beta}^*$ which maximize the above function; however, we can solve this convex optimization problem with an iterative method (for example, Newton's method).

Usually, to prevent coefficients being too high or too small (which could lead to overfitting), the logistic model is parametrized by solving instead the following regularized logistic regression problem:

$$\arg\min_{\boldsymbol{\beta}} \; -\mathcal{L}(\boldsymbol{\beta}; y_1|\mathbf{x_1}, \dots, y_m|\mathbf{x_m}) + \lambda \sum_{i=1}^{d} \beta_i^2 \tag{3.7}$$

In *scikit-learn*, a hyperparameter $C = \frac{1}{\lambda}$ is used instead of λ: smaller values of C imply a stronger regularization.

Finally, to perform binary classification with a binary logistic model, it suffices to set a threshold value for the probability of belonging to class 1 (typically 0.5), so that the following decision rule applies:

$$\hat{y} = \begin{cases} 1 & \text{if } p(\mathbf{x}; \boldsymbol{\beta}^*) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Notice that, similarly to the SVM model, also the logistic model is a linear classifier: in fact the decision boundary is the hyperplane defined by the linear combination of predictors $\beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d = 0$.

Results are reported in the following tables. The best performing configuration is highlighted in bold.

Table 3.7: Logistic Regression - grid search of hyperparameters

Hyperparameter	Values
C	$0.001, 0.01, 0.1, \mathbf{1}, 10, 100, 1000$

Table 3.8: Logistic Regression - results on test set

Pipeline	sensitivity	specificity	accuracy
Standardization	0.872	0.617	0.777
Std. + random oversampling	0.817	0.738	0.787
Std. + random undersampling	**0.830**	**0.745**	0.798
Std. + SMOTENC	0.813	0.716	0.777
Std. + PCA	0.881	0.596	0.774
Std. + PCA + random oversampling	0.804	0.730	0.777
Std. + PCA + random undersampling	0.800	0.702	0.763
Std. + PCA + SMOTE	0.804	0.716	0.771

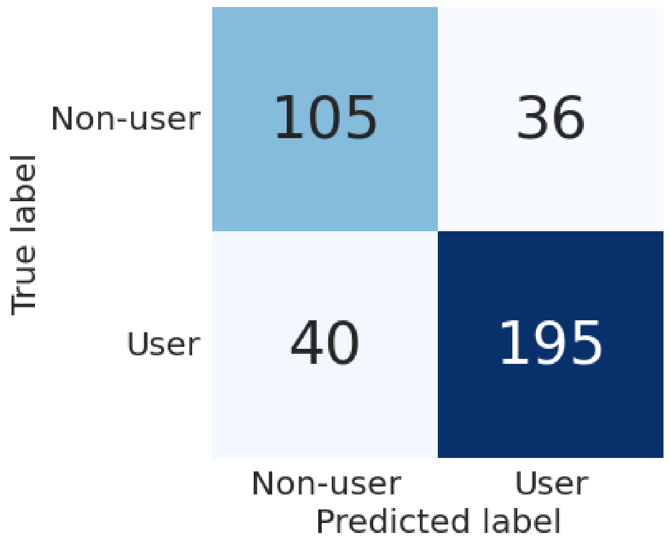


Figure 3.9: Logistic Regression - confusion matrix associated to the best performing configuration