**DATA RECUPERATION:**

**smote = SMOTE(random\_state=42)**

**X\_train\_balanced, y\_train\_balanced = smote.fit\_resample(X\_train, y\_train)**

This function adds synthetic samples for the minority class – and adds them to X train and Y train. This function is made to improve the imbalanced class. 50% of -1 and 50% of -1 : it generates new synthetic samples through interpolation.

**# removing a lot of features - we have 30 000 features now**

**file2\_new = file2.loc[:, file2.var() > 0.01]**

In this part we remove all the features where the variance is smaller than 0.01.

**Method 1: Logistic regression with penalty Lasso**

We cannot use directly the Lasso method because it is made with linear regression – but however we can use the logistic regression with the penalty L1 (Lasso penality). Lasso considered all the variables to be independent

**L1 regularization** adds a penalty term proportional to the absolute values of the coefficients to the model's cost function

Cost Function=Loss Function+α∑∣βi​∣

It shrinks the coefficient to 0 – the non important ones.

So we use this function :

**model = LogisticRegression(penalty='l1', solver='saga', C=alpha, max\_iter=10000, tol=1e-3, random\_state=42,class\_weight='balanced')**

solver saga : it determines the best algorithm to minimize the the cost function – good for huge data set

penalty = l1 : lasso penaly

C= it is 1/lambda

(small C – high penalty, simplify the model but underfitting)

(big C – small penalty – overfitting)

Max\_iter = maximum of iteration that the solver can do before crashing – default is 5000

Tol = the solver stops when the change in the loss function is smaller than this threshold.

Class\_weight = balanced – helph to improve for imbalanced class

Loss fct for log :

A black text on a white background

Description automatically generated

Loss fct for log + penalty

A math equation with black text

Description automatically generated with medium confidence

VALUE OF ALPHA:

**alpha\_values = np.logspace(-2, 0, 5)**

For every alpha we use create a model and we cross validate:

**cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)**

**cv\_scores = cross\_val\_score(model, X\_train\_balanced, y\_train\_balanced, cv=cv, scoring='balanced\_accuracy')**

It is a cross-validation technique for imbalanced dataset

**1st method: cv shuffle**

A black screen with green text

Description automatically generated

A black background with white text

Description automatically generated

A graph with a line

Description automatically generated

A black screen with white text

Description automatically generated

A graph with a line

Description automatically generated

**2nd method: with cv shuffle and x\_train\_balanced**

A black background with white text

Description automatically generated

A graph with a line

Description automatically generated

A black screen with white numbers

Description automatically generated  
  
**3rd method: X\_train\_balanced**

**4th method: nothing**

**Method 2 : Random Forest**

1 step

We need to use all the features in order to rank their importance after, so we built a tree with all the features.

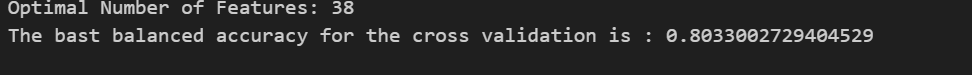
2step

We rank the features by importance – we find their indice and we sort them in a descending order

3 step

Now we select the number of features that maximixes the accuracy of our data. We create a loop that adds at each step a new feature from the most important to the least and iterate over the entire tree – we remove 29 000 features from the loop because we don’t want more than 1000 features.

**1ST VERSION WITH NOTHING**



A graph showing a number of features

Description automatically generated

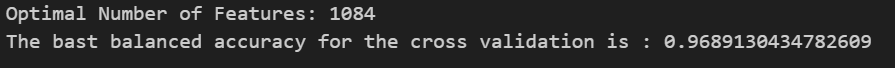
A black screen with white text

Description automatically generated

A graph with a line

Description automatically generated

**2nd WITH X\_train\_balanced :**



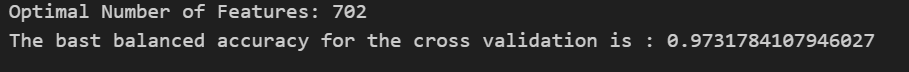
A graph with blue lines

Description automatically generated

A black screen with white text

Description automatically generated

**3 rd with X\_train\_balanced + cv shuffle:**



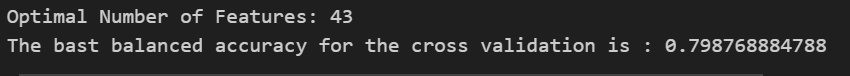
A graph with blue lines

Description automatically generated

A black screen with white text

Description automatically generated

**4th with Cv shuffle:**



A graph with blue lines and dots

Description automatically generated

A black screen with white text

Description automatically generated

A graph with a line

Description automatically generated

**Method with elastinet penalty**

**T2.3: Elastic Net Logistic Regression for Feature Selection**

The **elastic net** is a hybrid of LASSO and ridge regression that combines the L1 (sparsity-inducing) and L2 (grouping effect) penalties. For feature selection in high-dimensional data, this method is particularly useful as it can handle correlated features better than LASSO alone.

A math equation with black text

Description automatically generated

α: Mixing parameter (0≤α≤1):

* α=1: LASSO regression (L1 penalty).
* α=0: Ridge regression (L2 penalty).

A math equation with black text

Description automatically generated

**1st method (nothing):**

**2nd method (cv shuffle):**

**3rd method (cv shuffle + X\_train\_balanced):**

**4th method (X\_train\_balanced):**

**Method nearest Shrunken Centroid:**

It is an extension of LDA. The method involves shrinking the centroids (the average feature values for each class) toward the overall mean of the features, with the shrinkage parameter controlling how much the centroids are pulled toward the mean.

For LDA, the classification rule is based on computing the distance of a data point from the centroids of each class, using the covariance matrix. The class with the smallest distance to the point is chosen.

Features are independent – means theh covariance matrix is diagonal - we don’t have enough data to prove their dependency

By selecting only the most relevant features, NSC provides a sparse model.

*The idea behind NSC is to shrink the centroids toward the overall mean of the data to improve classification performance and reduce overfitting, especially when there are a large number of irrelevant features. This procedure shrinks the coefficients for each class to zero for the most irrelevant features.*

*The shrinkage procedure consists of two parts:*

* ***Step 1: Soft Thresholding****: Soft thresholding is a regularization technique where small coefficients are shrunk to zero, and larger coefficients are reduced but not to zero. It is smoother and more stable compared to hard thresholding, which would simply set the coefficients below a certain threshold to zero.*
  + *For each feature in each class, the coefficient (i.e., the difference between the class centroid and the overall mean) is shrunk toward zero.*
  + *The degree of shrinkage is controlled by a* ***shrinkage parameter (delta)****, which needs to be chosen.*
  + *Soft thresholding helps to discard irrelevant features, providing a sparse model that only includes the most informative features.*
* ***Step 2: Compute Shrunk Centroids****: After applying soft thresholding, the new centroids (shrunk centroids) are computed for each class. This reduces the influence of features that are not discriminative, improving the model's ability to generalize.*
* ***Step 3: Classification Rule with Shrunk Centroids****: Once the centroids are shrunk, the classification rule remains the same as in LDA: assign the class with the nearest centroid to the test data point. The difference now is that the centroids have been shrunk, potentially making them closer to each other and reducing overfitting.*

The **delta** parameter controls the amount of shrinkage applied to the centroids. A higher value of delta means stronger shrinkage (more features are shrunk to zero), while a lower value means less shrinkage.

Centroid of a class c:

A mathematical equation with numbers

Description automatically generated with medium confidence

The shrinkage is applied to the centroid of a class c :

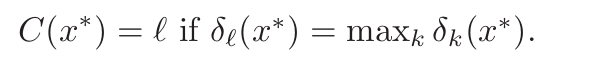
A mathematical equation with black text

Description automatically generated

Diagonal covariance LDA rule :

A mathematical equation with black text

Description automatically generated

Classification rule :  


We call the procedure nearest shrunken centroids (NSC). The shrinkage procedure is defined as follows. -- to shrink the coefficient to 0.

1st step A math equations with a white background

Description automatically generated with medium confidence

We prefer the soft thresholding it is a smoother operation and typically works better.

2nd step =

**def soft\_threshold(dkj, delta): return np.sign(dkj) \* np.maximum(0, np.abs(dkj) - delta)**

A black text on a white background

Description automatically generated

3rd step =

A black text on a white background

Description automatically generated

Delta need to be determined

Xg-boost

SVM – GBM