Introduction to Machine Learning

Master HPC-IA, Mines Paris

Ugo Pelissier
Dimitrios Tsitsos

January 18, 2023

Contents

1 Preliminary analysis of the dataset and analysis plan.

1. How many proteins are in this dataset?

To begin with, we load our data and we print the length of the dataset.

```
protein_labels = pd.read_csv("labels.csv",index_col=0)
print("There are", len(protein_labels), "proteins in this dataset.")
```

There are 1210590 proteins in this dataset.

2. What is the proportion of positive labels in the dataset? What is the proportion of negative labels? What metric(s) would be appropriate to validate the machine learning pipeline? Justify each of the metrics used.

```
def labels_proportion(protein_labels):
    n_true = (1*protein_labels["label"]).sum()
    p_true = n_true/len(protein_labels)
    p_false = 1-p_true
    plt.bar(["True","False"], [p_true,p_false])
    plt.show()
    print("p_true =",round(p_true,3))
```

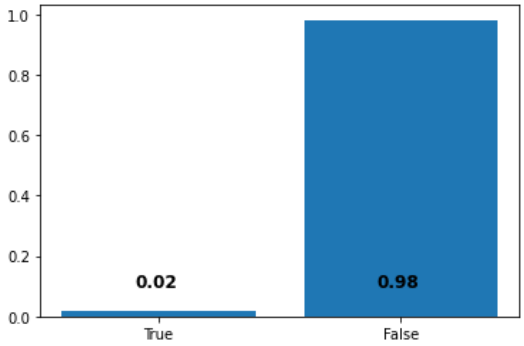

Figure 1: Proportion of true and false labels in the train data set

From the different metrics we have seen, we can first think of *accuracy* and *ROC_AUC*. However, these two metrics are better suited for balanced data set. In our case, we have to choose metrics that will take into account our uneven data set. From our research, both *imbalanced accuracy* and *F1-score* are good metrics when class metrics is uneven. To better understand the difference between these two metrics, we should write their formula:

$$\text{F1} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{Balanced Accuracy} = \frac{\text{specificity} + \text{recall}}{2}$$

From these formula, we can understand the following points:

- F1 score doesn't care about how many true negatives are being classified. When working on an imbalanced dataset that demands attention on the negatives, Balanced Accuracy does better than F1.
- In cases where positives are as important as negatives, balanced accuracy is a better metric for this than F1.
- F1 is a great scoring metric for imbalanced data when more attention is needed on the positives.

From this point, we believe that we need a similar attention to positive and negative labels so we prefer choosing balanced accuracy as our main metric, but we also plan on computing F1-score to have a comparison.

3. Now, looking at the complete list of labels, what is the number of elements in each category? What is the proportion of proteins labeled in two categories?

Looking at the *complete_labels.csv* file, we can notice that we have 134 columns, which represent subsets of the original secretion system families. We can then try to group our columns by families to know how many proteins belong to each big family.

```
def secretion_system_name(column_name):
    name = ""
    i = 0
    while(i<2):
        name+=column_name[i]
        i+=1
    return name
```

```
def secretion_system_list(complete_labels):
    column_name = complete_labels.columns
    secretion_system = np.array([])
    index = np.array([])
    i = 0
    for column in column_name:
        temp = secretion_system_name(column)
        if temp not in
            secretion_system.tolist():
            secretion_system =
                np.append(secretion_system,
                temp)
            index = np.append(index, i)
        i += 1
    index = np.append(index, i)
    index = index.astype(int)
    return secretion_system, index
```

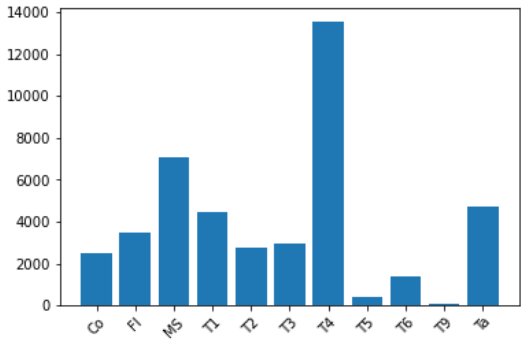

Figure 2: Number of elements in each secretion system family

We have 11 secretion system families, while 12 have been identified at least. Our data set is missing one or several secretion system families, which we need to keep in mind.

Then, we can compute the proportion of proteins labeled in two categories.

```
def double_proportion(complete_labels):
    complete_labels = complete_labels.values
    n = 0
    for i in range(len(complete_labels)):
        if ((complete_labels[i,:].sum())>1):
            n += 1
    print("p_double =",round(n/len(complete_labels),3))
```

Proportion of proteins labeled in two categories is 0.007

4. Identify and describe three challenges when working on this machine learning problem.

Looking at our datasets we can identify the following challenges:

- Imbalanced Dataset.
  As we can see in the second question the proportion of negative labels is way bigger than the positive one. This can lead to difficulties both on finding the best possible metric to validate the problem, and on generally processing the data. This represents a challenge for our cross-validation strategy that we will have the chance to explain in the following question.

- Large Number of Data.
  The number of data that we need to work on is really big which leads difficulties while processing them. The time for training a Machine Learning pipeline on this data set will be long, so we need to prepare carefully each attempt not to loose too much time with useless tries.

- Limited Number of Secretion Systems.
  In the *complete_labels.csv* dataset we can see that we don't have all the 12 possible Secretion Systems that we expect to identify. This is also an important challenge since we'll have to focus our efforts on the generalization ability of our Machine Learning algorithm, so that it is able to detect unknown secretion system families.

5. Devise (on paper) a cross-validation strategy that would enable to check that the machine learning pipeline would extend to secretion systems not annotated in our data set. You can write a pseudo-algorithm describing the cross validation strategy, or draw a schema explain your though. Explain how this cross validation strategy is a good way to check for generalization in this setting.

Our cross-validation strategy could be split in two parts.

The first one would be to create representative folders in regard of the proportion of positive and negative values in the initial data set. To be more specific, not only do we want to avoid having one folder with only false labels, but we also want that each folder have a proportion of true and false labels close to what we computed initially. That strategy is called **stratified k-folds cross validation**.

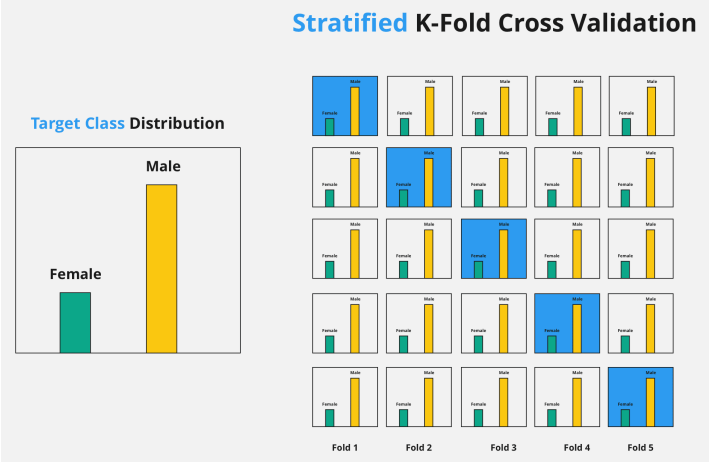

Figure 3: Stratified k-folds cross validation

In the same idea, we can stratify in order to have the same proportion of each secretion system family in each k-fold so that the pipeline does not over fit one family and keeps a good generalization ability.

2 Predicting protein function from its sequence.

2.1 Create a machine learning pipeline and evaluate it.

2.1.1 Cross-validation strategy

In order to check that the machine learning pipeline generalizes well to unknown secretion system types, protein families involved need to be grouped in the cross-validation setting. As such, we create a cross-validation such that each set contains specific groups of protein families.

2.1.1.1 Stratified Group Kfold implementation

The following figures (fig. 4, 5, 6) illustrate our work: each testing set contains a specific family of positive labels, and a number of negative labels to respect as much as possible the initial proportion of labels in each class. Initially, we thought this strategy would be done straight with the *StratifiedGroupKfold* option of **scikit-learn**, but it turned out it did not match our expectation. So we implemented on our own stratified group Kfold strategy, which is described above.

- First, we worked on the true labels in order to isolate each group for each testing set and storing the number of protein in each group.
- Then, we completed each testing set with a number of false labels to ensure that the proportion of true and false labels in the training and testing sets was approximately the proportion of the initial dataset.

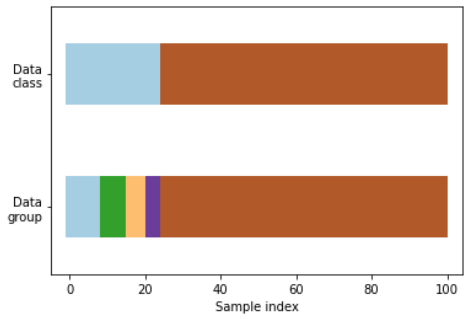

Figure 4: Illustration of class and group repartition

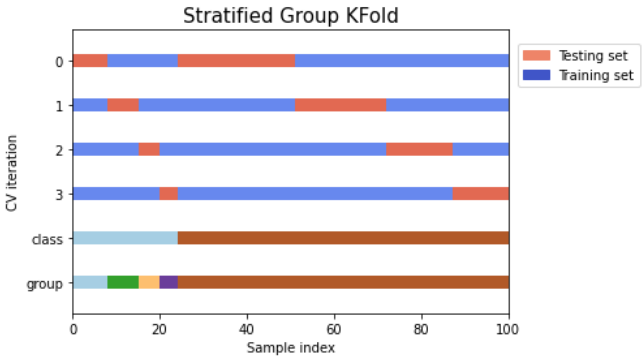

Figure 5: Illustration of cross-validation strategy

Note that for each protein belonging to more than one family, we decided to assign it to a single group for easier manipulation. To do this, we assigned each protein to the group that was least represented among the groups to which it belongs.

2.1.1.2 Random undersampling

A first test of a Machine Learning pipeline quickly showed us that this cross-validation strategy keeps imbalanced data sets, making it hard for the classifier to learn correctly to predict true labels.

We first implemented a basic random undersampling to have balanced data sets. For each cross-validation fold, we select a number of false labels, among the indexes already selected, to match the number of true labels in the considered group (fig. 6).
When the number of true labels in the testing set was too small (below 20), this iteration was let aside since the samples are not representative enough.

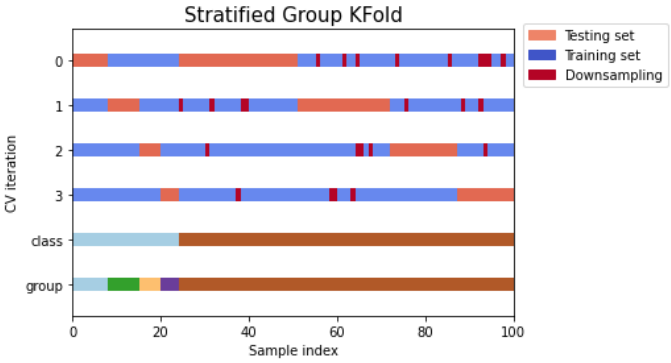

Figure 6: Basic random undersampling

2.1.2 Features processing

We have identified 4 challenges regarding features processing in this dataset:

- The feature matrix has 952 elements, meaning we'll have to go through a step of **dimensionality reduction**, using either PCA or other techniques such as SVD.
- The scales among features can be very different, most of them laying between 0 and 1, but other being much more important. **Scaling** the data appropriately will be another important step.
- Looking at the range of some features also indicate the presence of **outliers** in the data set. We'll have to implement a statistical strategy to get rid of them.
- Finally, the feature matrix is **sparse**, meaning we can either choose to discard the features with too many null values, or come up with a feature strategy well suited for sparse data set.

We have tackled them in the following order:

- First, we standardize features by removing the mean and scaling to unit variance by using the *StandardScaler* tool from **scikit-learn**.
- Then, we tried to remove the outliers by computing, for each feature and each protein, the associated **z-score**. It tells us how many standard deviations a given value is from the mean. We define an observation to be an outlier if it has a z-score less than -4 or greater than 4. However, this strategy was not giving better results in a first approximation so we decided not to remove outliers.
- Finally, we tackled the dimensionality reduction and the sparse aspect at the same time. We have the choice between PCA and SVD to reduce the dimensionality of our data. We'll explore both options in the next part.

2.1.3 Performance on the training data

First, we worked only with the first 30 000 proteins to be able to iterate quickly other the different pipelines that we were testing.

One important point is to explain which metrics we chose to compare the models. As said in the first part of this project, we used **balanced accuracy** since it is a good metric for imbalanced data set. In addition, we decided also to compute the **F2 score**, which is an extension of the F1 score that put a greater emphasis on the recall than the precision.

$$F_\beta = (1+\beta^2).\frac{\text{Precision.Recall}}{(\beta^2.\text{Precision}) + \text{Recall}}, \quad F_2 = F_{\beta=2}$$

Before any additional data pre-processing, we wanted to fix which classifier we were going to use and optimize. So we selected a variety of classifiers and tested them against our training data set.

Since we are working in a cross-validation approach, we looked at the best combination of balanced accuracy and F2 score to select the best classifier to work with. We tested among the following classifiers:

- LogisticRegression
- Nearest Neighbors
- SVM
- RBF SVM
- Gaussian Process
- Decision Tree
- Random Forest
- Neural Net
- AdaBoost
- Naive Bayes
- QDA

According to our results, we chose the **SVC** algorithm in the **Support Vector Machines** classifiers proposed by **scikit-learn**. This classifier supports an option about the class weights. When put to *balanced*, this option enables the classifier to take into account the fact that the data set is imbalanced, thus giving a greater importance o the less represented class in the training operation. Additionally, this classifier supports some other options that we tested but without giving huge improvement in the performance.

The main part of the optimization for the training concerned 3 aspects:

- The ratio of true and false labels for the random undersampling operation $\rightarrow [2, 5, 10, 20, 50]$
- The process of dimensionality reduction, namely PCA or SVD $\rightarrow [\text{PCA}, \text{SVD}]$
- The number of components for the step, either PCA or SVD $\rightarrow [1, 2, 5, 10, 20, 50, 100, 200, 500]$

We created a pipeline to test at the same time all the possible combinations of parameters written above. Representing a total of 90 computations, we can assemble the results in the following table (Table 1).

ROC AUC	**Balanced accuracy**	**F2 score**
0,786	0,786	0,428
0,814	0,814	0,489
0,802	0,802	0,407
0,859	0,859	0,483
0,877	0,877	0,511
0,886	0,886	0,525
0,889	0,889	0,540
0,897	0,897	0,616
0,891	0,891	0,533
0,885	0,885	0,524
0,894	0,894	0,540
0,893	0,893	0,537
0,899	0,899	0,548
0,889	0,889	0,541
0,897	0,897	0,546
0,878	0,878	0,530
0,906	0,906	0,563
0,698	0,698	0,322
0,787	0,787	0,429
0,819	0,819	0,480
0,816	0,816	0,432
0,866	0,866	0,501
0,897	0,897	0,565
0,919	0,919	0,602
0,893	0,893	0,636
0,888	0,888	0,674
0,908	0,908	0,588
0,932	0,932	0,636
0,920	0,920	0,605
0,900	0,900	0,610
0,895	0,895	0,585
0,905	0,905	0,583
0,908	0,908	0,578
0,910	0,910	0,582
0,911	0,911	0,595

ROC AUC	**Balanced accuracy**	**F2 score**
0,699	0,699	0,323
0,799	0,799	0,423
0,831	0,831	0,498
0,821	0,821	0,438
0,893	0,893	0,557
0,879	0,879	0,610
0,903	0,903	0,668
0,886	0,886	0,688
0,795	0,795	0,699
0,885	0,885	0,661
0,882	0,882	0,653
0,883	0,883	0,656
0,889	0,889	0,655
0,884	0,884	0,658
0,855	0,855	0,652
0,878	0,878	0,655
0,888	0,888	0,654
0,893	0,893	0,673
0,707	0,707	0,325
0,803	0,803	0,421
0,820	0,820	0,498
0,823	0,823	0,470
0,845	0,845	0,592
0,843	0,843	0,667
0,805	0,805	0,697
0,816	0,816	0,760
0,670	0,670	0,669
0,837	0,837	0,717
0,818	0,818	0,707
0,825	0,825	0,712
0,868	0,868	0,735
0,843	0,843	0,722
0,813	0,813	0,708
0,840	0,840	0,733
0,845	0,845	0,732

Table 1: Table of the scores of the different classifiers for each combination of parameters

Inside this table, we selected two classifiers: the one giving the best balanced accuracy and the one giving the best F2 score. The corresponding cells are highlighted in green. The first one will be named *b_acc_clf*, since it maximizes the balanced accuracy, and the other one *f2_clf* for an analogous reason.

Looking back to our parameters tables, where are the two sets of parameters giving these two classifiers:

- $b\_acc\_clf = [10, \text{SVD}, 200]$
- $f2\_clf = [50, \text{PCA}, 200]$

We'll use the validation data set to make a final choice of classifier, as it is described at the next section.

2.1.4 Performance on the validation data

In order to choose between these two classifiers, we test them on the validation data set. We have to get a closer look to the confusion matrices to understand the difference in the two approaches (fig. 7, 8). On the one hand, *b_acc_clf* works better on the true labels, but will also wrongly classify as true a lot of false labels. On the other hand, we could say that *f2_clf* is more cautious, meaning that when

it classifies a protein as true, there is a higher chance that it is really the case than with the other classifier.

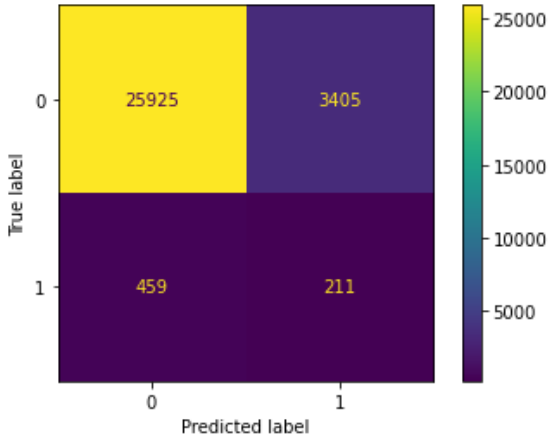

Figure 7: *b_acc_clf*

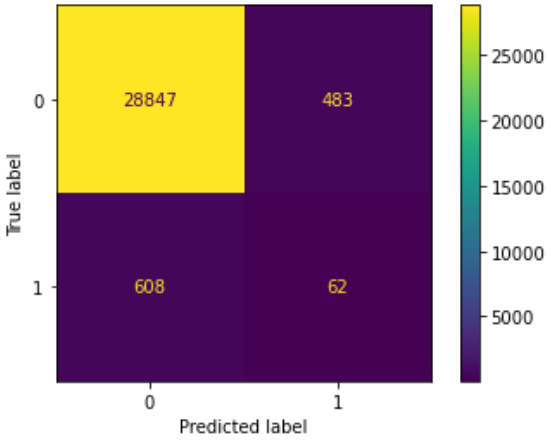

Figure 8: *f2_clf*

2.1.5 Choice of a model

Overall, we decided to keep the *f2_clf* for the reason explained above and we can test it on the test data set. This classifier performs a PCA as a first step of pre-processing.

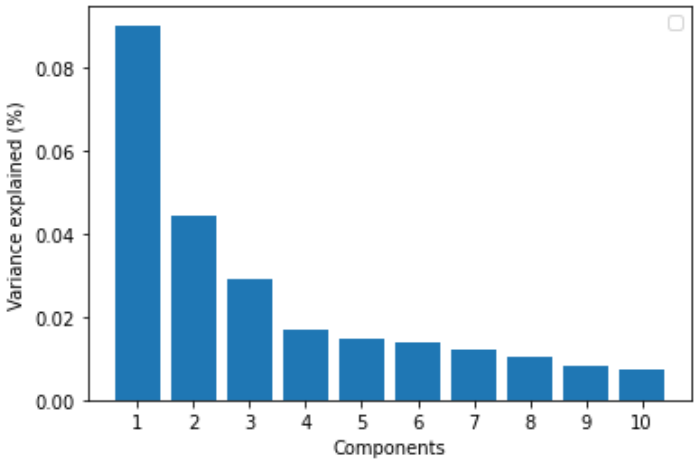

Figure 9: Percentage of variance explained by the first 10 components of our PCA decomposition

2.2 Exploring strategies: Learning on large datasets

2.2.1 Detailed explanation of random undersampling

Imbalanced datasets are those where there is a severe skew in the class distribution, such as *1:100* or *1:1000* examples in the minority class to the majority class. In our case, we have *1:62*.

This bias in the training dataset can influence many machine learning algorithms, leading some to ignore the minority class entirely. This could be an issue as it is typically the minority class on which predictions are most important.

One approach to addressing the problem of class imbalance is to randomly resample the training dataset. The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling.

Oversampling by duplicating existing protein sequences can quickly lead to overfitting on the training dataset. For this reason, we chose to explore random undersampling.

As explained in the previous part, we use random undersampling on every false labels training set in the cross validation folds (fig. 6). As such, we can decide how many false labels we want to extract. The extraction process can be parameterized by the ratio $r = \frac{\text{false labels}}{\text{true labels}}$ we wish to get.

2.2.2 Random undersampling performance

We studied the influence of the parameter r on the performance of the training data. We can note that $1 < r < 62$ since it cannot be bigger than the actual proportion of true and false labels in our dataset. As such, we trained the model for various values of $r \in [10, 20, 30, 40, 50, 60]$ and observed the resulting balanced accuracy and F2 score (fig. 10).

The more unbalanced the dataset is, the closer balanced accuracy and F2 score are. However, balanced accuracy tends to decrease with a more unbalanced dataset, whereas the F2 score has an opposite behavior.

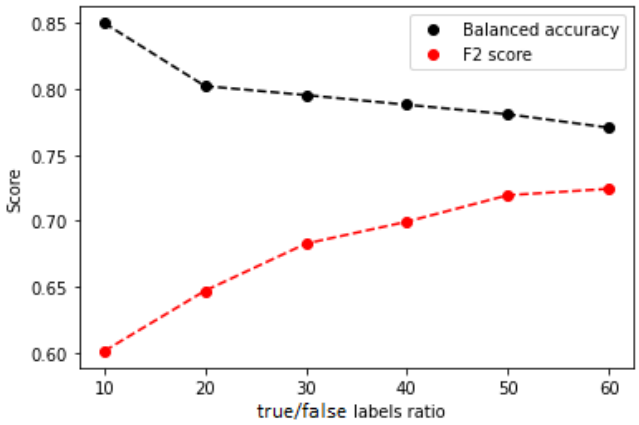

Figure 10: Scores as a function of the undersampling ratio

These observations are very interesting because they imply that we have to make a trade-off between undersampling and scoring. Indeed, a small undersampling ratio enables us to train our model faster, but since we are more interested in maximizing F2 score, we should consider a rather big undersampling ratio.

In fact, the results of our parameterized study of models (Table 1) made us choose two models, one the with a small undersampling ratio ($r = 10$ for *b_acc_clf*) and another one with a bigger ratio ($r = 50$ for *f2_clf*). A quick look at the confusion matrices (fig. 11, 12) show the same difference of behavior that we observed at the end of the validation part (fig. 7, 8).

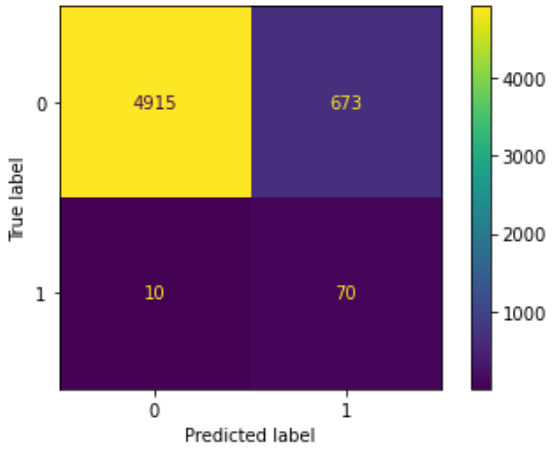

Figure 11: Confusion matrix for ratio 10

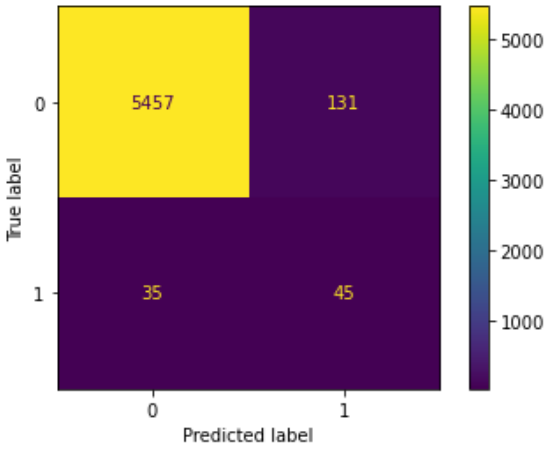

Figure 12: Confusion matrix for ratio 60