Comparative Analysis of Classification Algorithms on Imbalanced Datasets: CVA and Salary Prediction

Dumitru Bianca Stefania

May 2024

1. Datasets

1.1. Cerebrovascular accident (CVA) prediction

This dataset includes medical values and lifestyle information for 5,110 individuals. The goal is to predict whether a person might have a CVA based on various health indicators. The target attribute is binary (cerebrovascular_accident). Some key attributes include:

- mean_blood_sugar_level (numeric)
- cardiovascular_issues (categorical)
- body_mass_index (numeric)
- sex (categorical)
- tobacco_usage (categorical)
- high_blood_pressure (categorical)

1.2. Salary range prediction

This dataset contains personal, educational, and professional information about various employees. The objective is to perform binary classification to predict whether employees earn more than $50K per year or not. Some key attributes include:

- fnl (numeric): Socio-economic feature of the population.
- hpw (numeric): Number of hours worked per week.
- job (categorical): Job title of the individual.
- education (categorical): Type of education the individual has.
- gender (categorical): Gender of the individual.

2. Data exploration

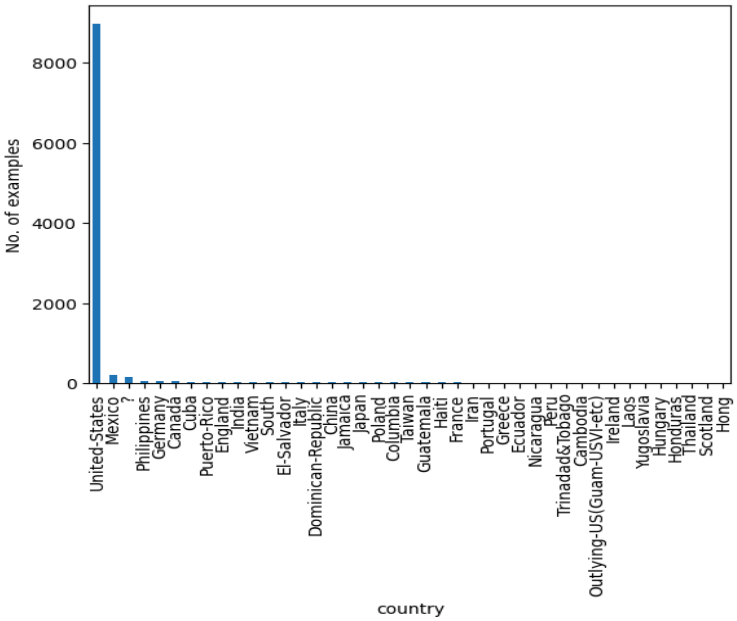
2.1. Analysis of the type of attributes and their value range

2.1.1. Continuous numeric attributes

For each numerical attribute in both datasets, I created a box plot and a table describing key statistics (mean, standard deviation, minimum value, etc.) for the respective attribute. After observing the BoxPlots for both datasets, I noticed that, for many of the attributes, there are quite a few outlier values, which will later be removed and imputed.

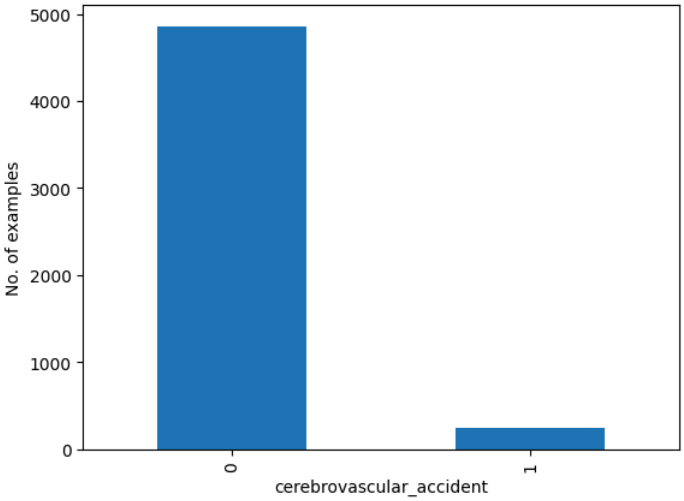
2.1.2. Discrete or ordinal attributes

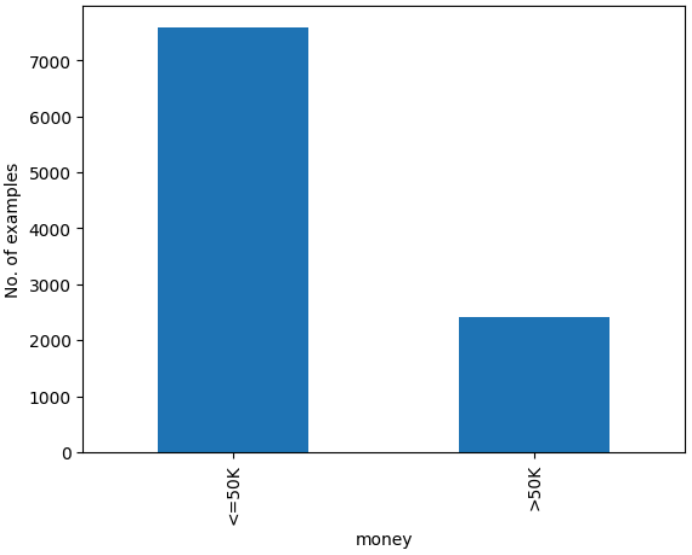
For each categorical attribute of both datasets, I created a plot showing the frequency of encountered values. What I noticed with both datasets, but especially with the Cerebrovascular accident (CVA) dataset, is that some attributes are very unbalanced, with one class being the majority. There are also very unbalanced attributes, such as 'race,' where 'White' predominates, or 'country,' where the most common value is 'United States.'



2.2. Class balance analysis

The classes in the CVA dataset are highly unbalanced, with most examples representing the class of people who have not had a CVA. The division between classes is very similar in the training and test sets. For this reason, classification may be more difficult, with a risk of erroneous predictions for the class with low support (people who have had a CVA). The SalaryPrediction dataset also shows an imbalance between the classes, but it is less significant compared to the first dataset. The class with low support is that of people with a salary > 50K. In the data processing step, I tried to address this imbalance through various resampling methods. Due to the imbalance, the models may perform worse in prediction accuracy, particularly regarding precision, recall, and F1 metrics.

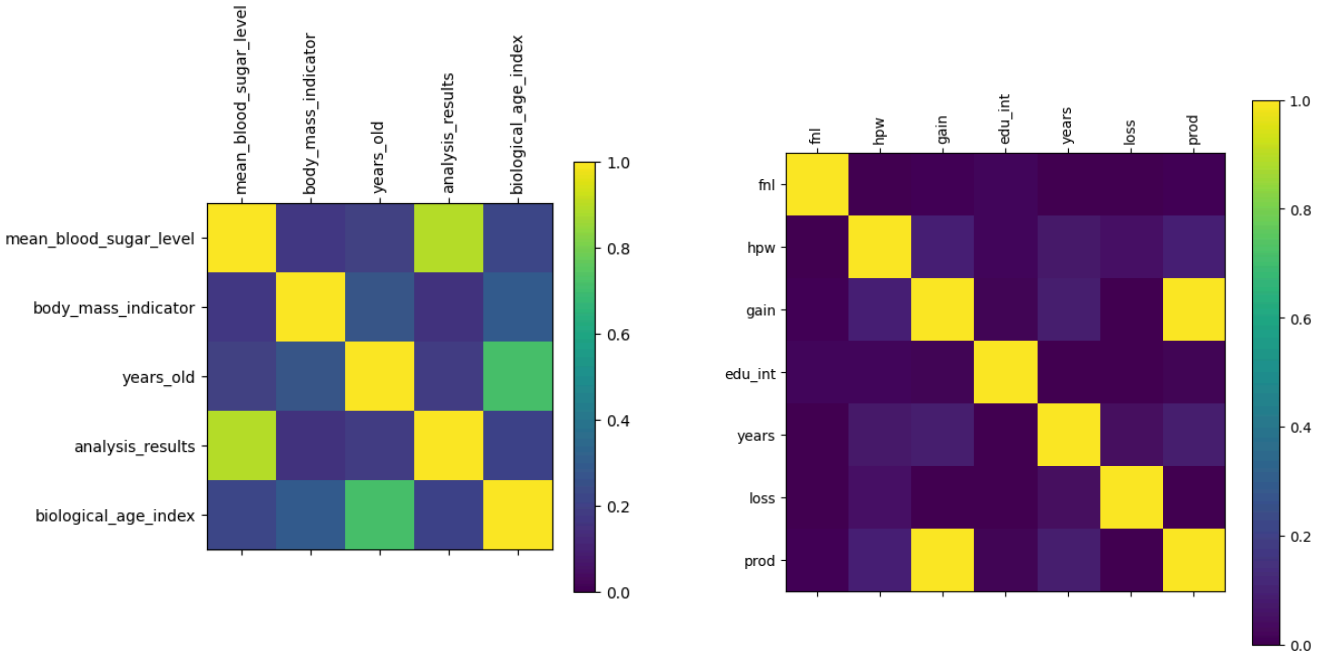




2.3. Correlation analysis between attributes

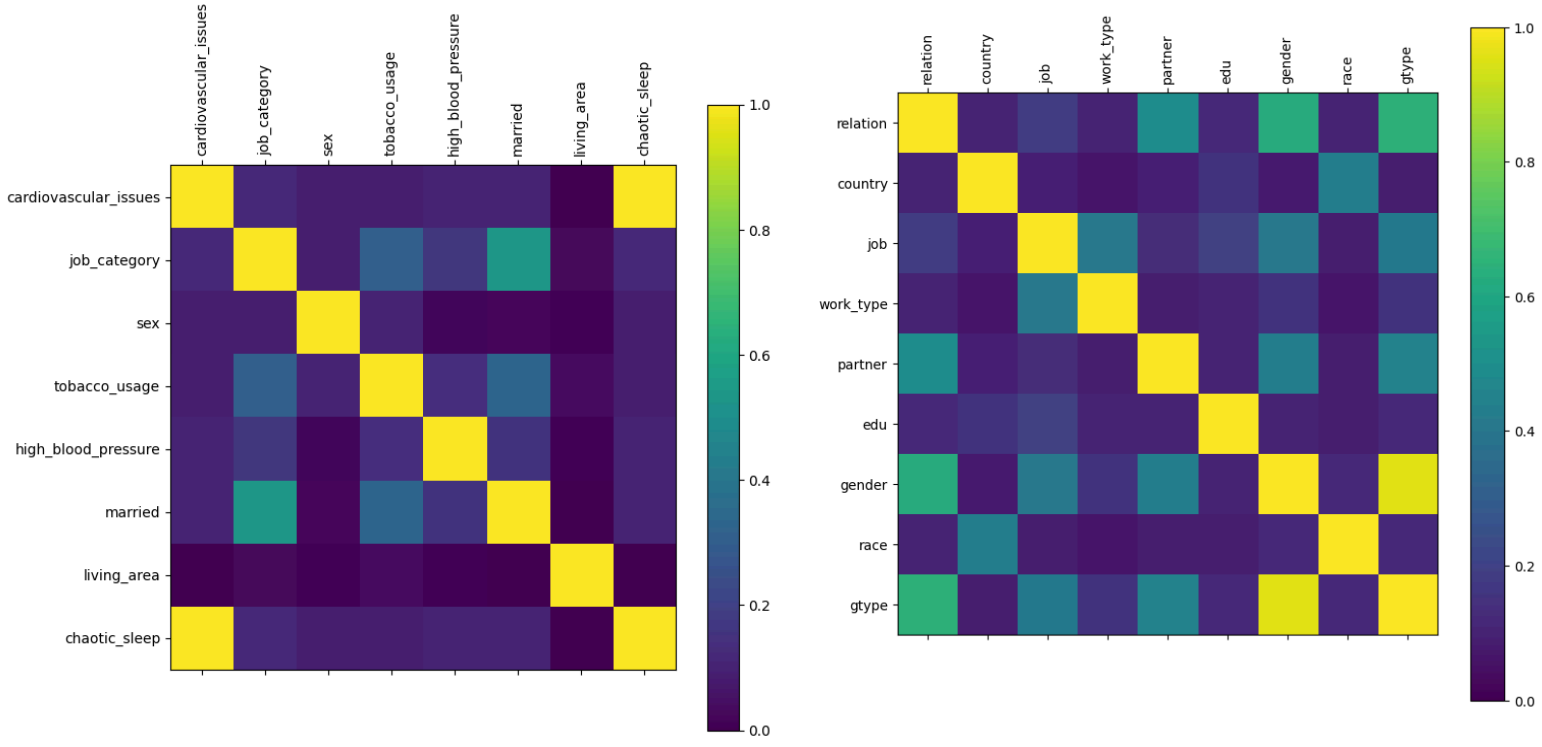
2.3.1. Correlation analysis between continuous numerical attributes

To measure the degree of correlation between each pair of numerical attributes in the dataset, I used the Pearson method, which returns a value between -1 and 1, with values close to -1 and 1 indicating a very strong correlation. After conducting research, I concluded that a strong correlation exists for values greater than 0.7. Based on this metric, I subsequently eliminated one attribute from each strongly correlated pair.



2.3.2. Correlation analysis between categorical attributes

To measure the degree of correlation between each pair of categorical attributes in the dataset, I used the Chi-square test, followed by the calculation of the Cramér's V value. A higher Cramér's V value indicates a very strong correlation. Based on my research, I concluded that a strong correlation exists for values greater than 0.7. According to this metric, I subsequently eliminated one attribute from each strongly correlated pair.



3. Data preprocessing

3.1. Replacing missing data for an attribute

I tried using two types of imputers (univariate - SimpleImputer and multivariate - IterativeImputer), first fitting them on the training set and then applying the transformation to both the training and test sets. After various tests, I concluded that using the univariate imputer was more suitable for both datasets. For numerical attributes, the average value was used, and for categorical attributes, the most frequent value was used. Additionally, for categorical attributes, I also treated "not_defined" and "?" as missing values and imputed them accordingly.

3.2. Replacing extreme values for an attribute

Considering the analysis from point 1.1.1, I noticed that there are many outliers. To identify them, I used the inter-quartile range (IQR). After identifying the outliers, I replaced these values with NaN to impute them later. I experimented with various values of Q1 and Q3, concluding that the best fitting values for both datasets are Q1 = 0.2 and Q3 = 0.8.

3.3. Removal of redundant attributes

Based on the correlation analysis from point 1.3. in the CVA data set, I found a strong correlation (>0.7 - Pearson for numerical attributes and >0.7 Cramer for categorical attributes) for the following pairs of attributes:

- mean_blood_sugar_level - analysis_results
- years_old - biological_age_index
- chaotic_sleep - cardiovascular_issues

I chose to remove analysis_results, biological_age_index and chaotic_sleep.

In the SalaryPrediction dataset, I found a strong correlation for the following attribute pairs:

- gain - prod
- gtype - gender

I chose to remove gain and gtype.

3.4. Standardization of numerical attribute values

I experimented with data standardization using three scalers: StandardScaler, MinMaxScaler, and RobustScaler. For both datasets, I chose to use StandardScaler, as it provided better results. I first fitted the scaler on the training dataset and then applied the transformation to both the training and test sets.

3.5. The transformation of categorical attributes into a numerical form

To encode the target variable (necessary for the SalaryPrediction dataset), I used LabelEncoder. To encode the categorical predictor variables, I used the pandas.get_dummies method.

3.6. Using over-sampling or under-sampling to balance the dataset

Considering the strong imbalance presented by the datasets (especially CVA), and after testing the models on the datasets without resampling, I decided that a resampling method was necessary. Therefore, I researched and identified several methods for oversampling, undersampling, and combinations of both: SMOTE, SMOTEENN, SMOTETomek, ADASYN,

and RandomOverSampler. After various attempts, I concluded that SMOTETomek was the most suitable for my datasets, effectively combining oversampling and undersampling.

4. Classification

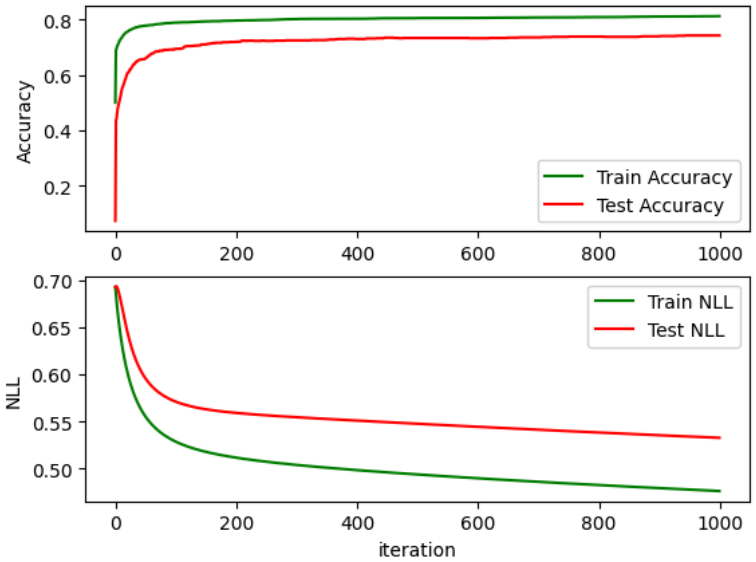
4.1. Logistic Regression

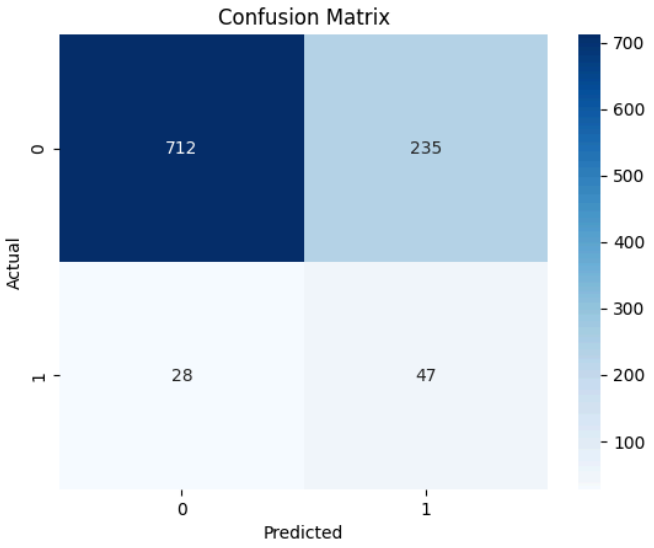
4.1.1. Manual implementation

I manually implemented logistic regression, adapting it for my case by adding a L2 (Ridge Regression) regularization to avoid overfitting. During the training, I calculated at each iteration the cost (regularized Negative Log Likelihood) and the accuracy of the predictions on the train and test set and plotted them in order to be able to observe if the model is overfitting. The hyperparameters used (and decided through various attempts) are:

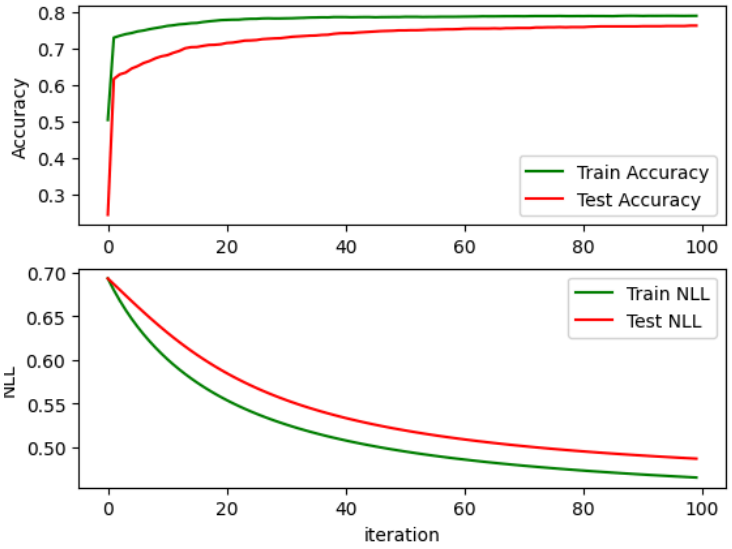
Hyperparameters used	learning rate	alpha (regularization coefficient)	epochs_no
The CVA data set	0.1	3	1000
The SalaryPrediction dataset	0.1	0.1	100

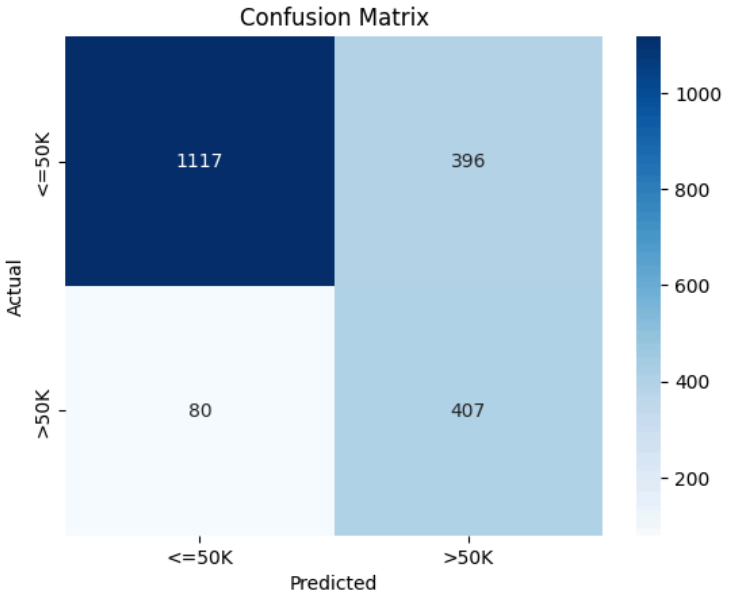
Plotting the evolution of the accuracy and the cost function in the case of the hyperparameters specified above for the CVA dataset, respectively the confusion matrix resulting from the prediction of the model on the test set:





Plotting the evolution of the accuracy and the cost function in the case of the hyperparameters specified above for the SalaryPrediction dataset, respectively the confusion matrix resulting from the prediction of the model on the test set:





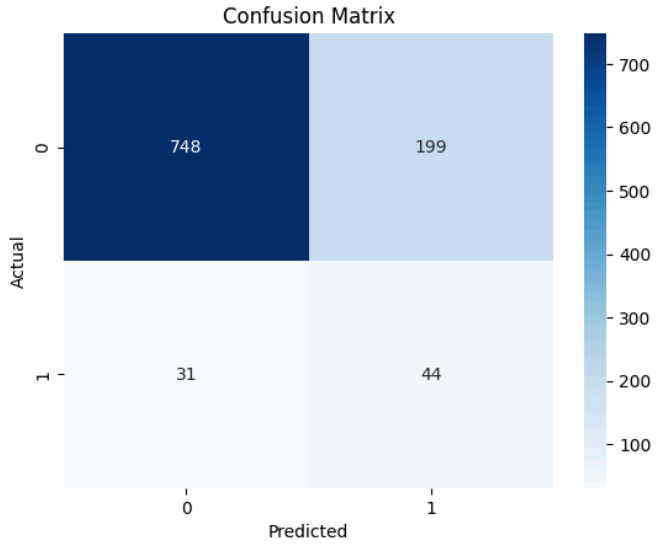
4.1.2. Implementation using the scikit-learn library

To determine the hyperparameters using the model from scikit-learn, I used GridSearchCV to find the best parameters to maximize the F1 score (since the dataset was unbalanced and resampled, this metric was usually the most informative). I tried to optimize the hyperparameters C, penalty, and tol.

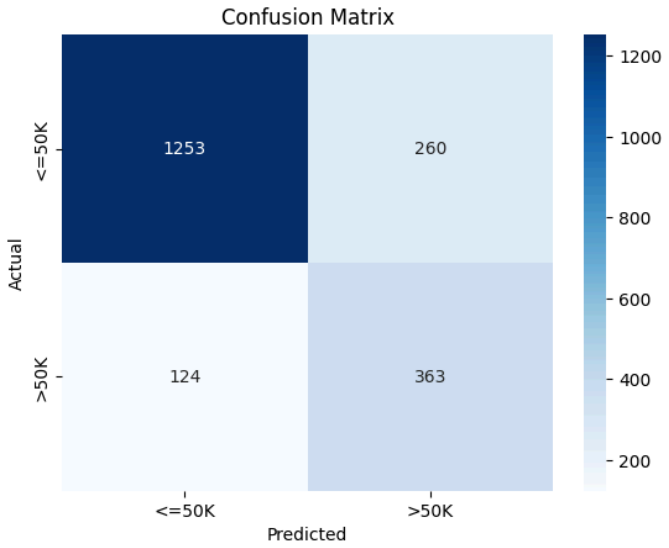
Hyperparameters used	max_iter	solver	C	penalty	toll
The CVA data set	1000	saga	0.01	l2	0.001
The SalaryPrediction dataset	1000	saga	100	l2	0.1

By manually changing the parameters, I noticed that the biggest influence is the solver (for example, the lbfgs solver gives very different, much worse results, considering the same parameters) and the C value of the regularization coefficient within the L2 penalty (a smaller value causes a stronger regularization).

The confusion matrix resulting from the prediction of the model on the CVA test set:



The confusion matrix resulting from the prediction of the model on the SalaryPrediction test set:



4.2. Multi-Layered Perceptron

4.2.1. Manual implementation

I manually implemented the MLP model with L2 regularization (Ridge Regression) to avoid overfitting. I used the CrossEntropy cost function and calculated the gradient within the Linear layer.

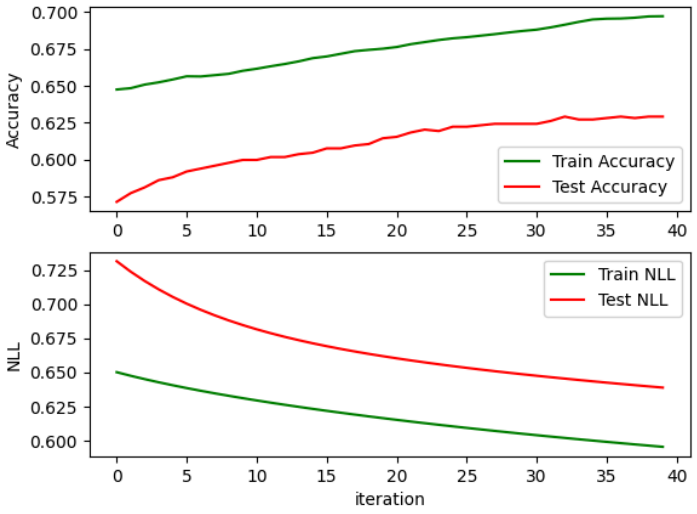
The FeedForwardNetwork consists of a Linear input layer (which receives the N features of the dataset), a hidden ReLU layer (with hidden_units neurons), and a Linear output layer that returns 2 values (the probabilities of the prediction being 0 or 1).

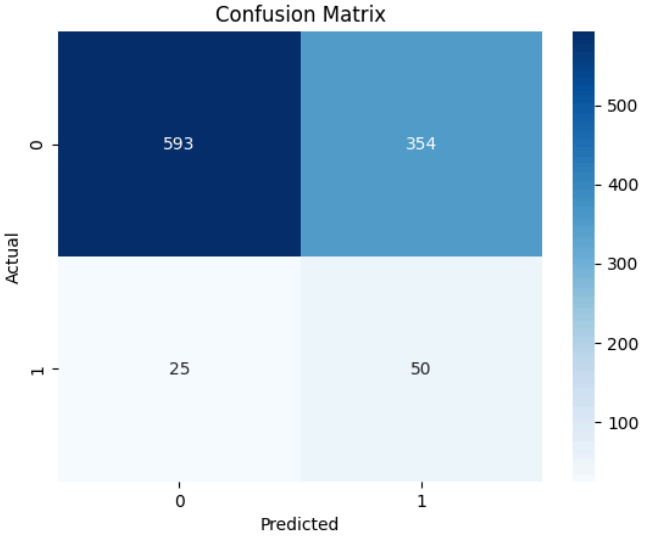
During training, I calculated the cost (regularized CrossEntropy) and the accuracy of the predictions on both the training and test sets at each iteration, and plotted them on the same graph to observe if the model was overfitting. The hyperparameters used (determined through various attempts) are:

Hyperparameters used	epochs _no	alpha (regularization coefficient)	batch_ size	hidden_units (number of neurons of the ReLU layer)	learning_rate
The CVA data set	40	0.01	200	20	0.0005
The SalaryPrediction dataset	40	0.01	200	20	0.0005

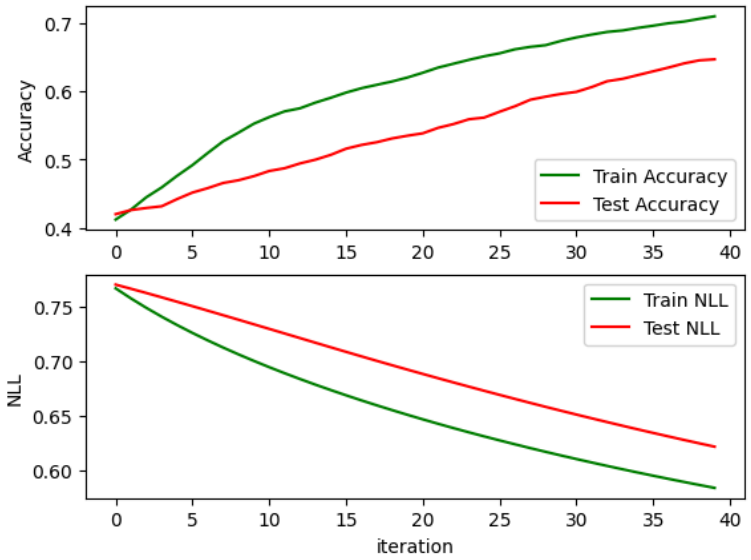
I noticed that the parameters alpha and batch_size have the greatest influence on the result, their modification greatly changing the results.

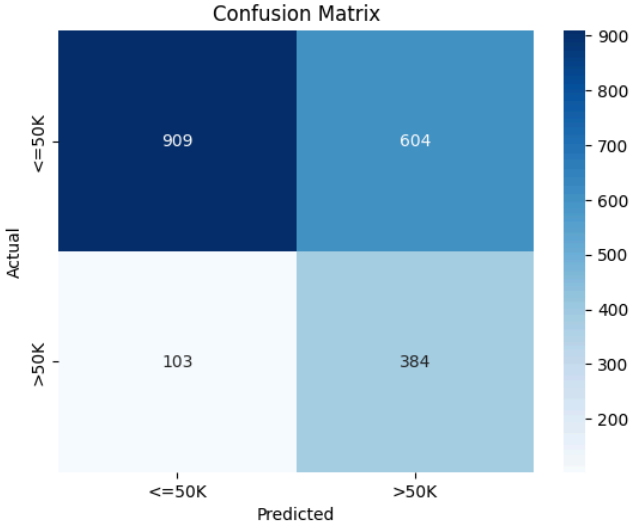
Plotting the evolution of the accuracy and the cost function in the case of the hyperparameters specified above for the CVA dataset, respectively the confusion matrix resulting from the prediction of the model on the test set:





Plotting the evolution of the accuracy and the cost function in the case of the hyperparameters specified above for the SalaryPrediction dataset, respectively the confusion matrix resulting from the prediction of the model on the test set:





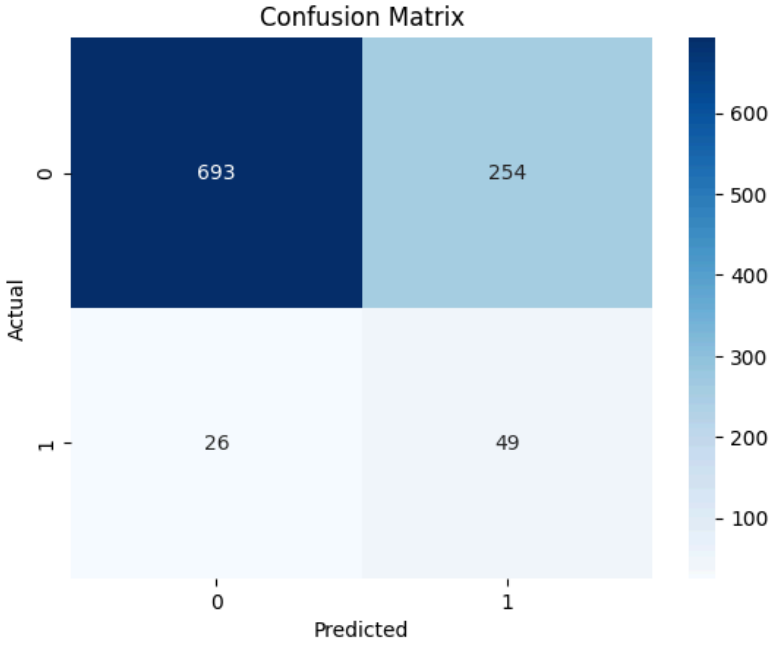
4.2.2. Implementation using the scikit-learn library

To determine the hyperparameters for the scikit-learn MLPClassifier model, I used GridSearchCV to find the best parameters to maximize the F1 score. I optimized the alpha, activation, and solver hyperparameters. Additionally, I used the early_stopping=True parameter to help prevent overfitting.

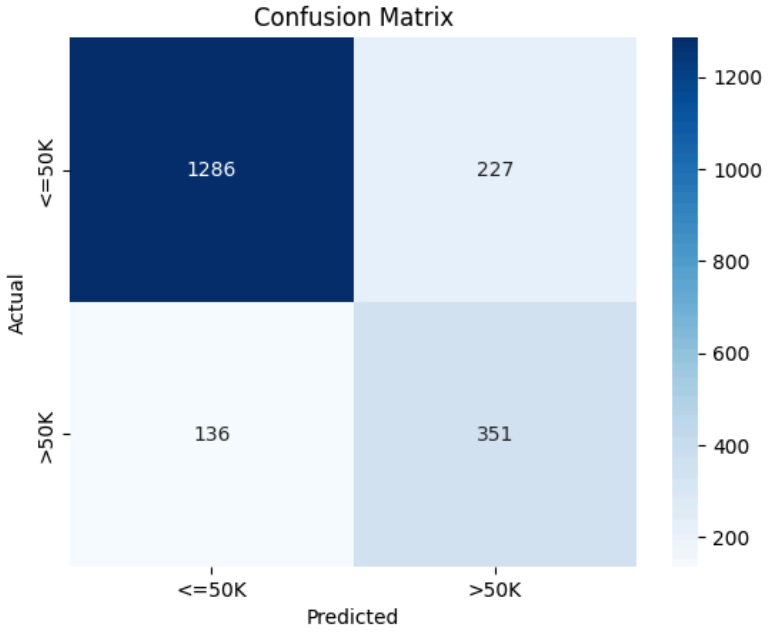
Hyperparameters used	alpha	solver	activation
The CVA data set	3	sgd	resume
The SalaryPrediction dataset	0.1	adam	resume

By manually changing the parameters, I noticed that the alpha value of the L2 regularization coefficient has the greatest influence (a higher value determines a stronger regularization).

The confusion matrix resulting from the prediction of the model on the CVA test set:



The confusion matrix resulting from the prediction of the model on the SalaryPrediction test set:



5. Evaluation of algorithms

As mentioned above, the datasets are quite unbalanced. Therefore, it is essential to analyze the quality of the predictions, focusing particularly on the metrics of precision, recall, and F1, in addition to accuracy.

Accuracy measures the proportion of the overall correctness of a classification model, that is, how many of its predictions are correct.
Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision indicates the proportion of relevant examples (correct positives) among all examples that the model predicted as positive. Intuitively, precision is the ability of the model not to categorize as positive an example that is actually negative.
Precision = TP / (TP + FP)

Recall measures the ability of the model to identify all relevant (positive) examples in the data set.
Recall = TP / (TP + FN)

The F1 score is the harmonic mean of precision and recall, offering a balance between them.
F1 = (2 * Precision * Recall) / (Precision + Recall) = (2 * TP) / (2 * TP + FP + FN)

Where:

- TP (True Positive) are correctly identified positive cases.
- TN (True Negative) are correctly identified negative cases.
- FP (False Positive) are negative cases incorrectly identified as positive.
- FN (False Negative) are positive cases incorrectly identified as negative.

For the CVA dataset, positive is considered class 1 (of people who have had a CVA), and for the SalaryPrediction dataset, positive is considered class >50K.

After running the 4 algorithms for the CVA data set, I obtained the metrics:

Model	Accuracy	Precision	Recall	F1-score
Manual Logistic Regression	0.742661	0.166667	0.626667	0.263305
Scikit Logistic Regression	0.774951	0.181070	0.586667	0.276730
Manual MLP	0.629159	0.123762	0.666667	0.208768
Scikit MLP	0.726027	0.161716	0.653333	0.259259

It can be seen from the table that all algorithms have similar performance, resulting in values of approximately 0.7 for accuracy, 0.15 for precision, 0.6 for recall, and 0.25 for F1. The lowest value is precision, which indicates that the model classifies as positive examples that are actually negative. This likely occurs due to the data imbalance, which I attempted to address through resampling. The model remains strongly influenced by the initial predominance of the negative class, which the resampling method did not fully resolve. Without resampling, the precision is 0, meaning no examples are categorized as positive.

Among the four algorithms, the best predictions are made by LogisticRegression from scikit-learn, with the hyperparameters described above. The recall value is better for the

manual MLP, but this seems to come at the expense of other metrics. The two manual regression algorithms have extremely similar values, while the MLP algorithms show slightly different values, which could potentially be improved by adjusting the hyperparameters. Depending on which metric is more important for the dataset, we could choose an algorithm that excels in that specific metric, even if it means compromising on others.

After running the 4 algorithms for the SalaryPrediction data set, I obtained the metrics:

Model	Accuracy	Precision	Recall	F1-score
Manual Logistic Regression	0.762000	0.506849	0.835729	0.631008
Scikit Logistic Regression	0.808000	0.582665	0.745380	0.654054
Manual MLP	0.646500	0.388664	0.788501	0.520678
Scikit MLP	0.818500	0.607266	0.720739	0.659155

We can see by analyzing the table that, compared to the one for CVA, the values for this dataset are much better across all parameters. This improvement is due to a less prominent imbalance, which was also mitigated by the resampling methods, resulting in better outcomes starting from a less imbalanced dataset.

In this case, the algorithms have somewhat similar performances, but the MLP algorithm shows poorer results, possibly due to the inability to add early stopping, modify the solver, or due to unfavorable hyperparameter settings (such as alpha, batch_size, etc.). However, the values for manual MLP are also close to those of the other algorithms, with only lower values for accuracy and precision. Excluding MLP, the approximate values are ~0.78 for accuracy, ~0.55 for precision, ~0.75 for recall, and ~0.6 for F1 score.

The best metrics were obtained from the MLPClassifier algorithm from scikit-learn, which were extremely close to those of LogisticRegression from scikit-learn. For recall, the best value came from the manual Logistic Regression algorithm, though the other metrics suffered.