University of Applied Sciences and Arts Northwestern Switzerland
School of Business

Assignment2: Classification Task

Group 1:
Erëza Abdullahu
Piero Jean Pier Hierro Canchari
Daniele Porumboiu
Piermichele Rosati

Supervisors:
Prof. Dr. Holger Wache
Dr. Gwendolin Wilke

Master of Science in Business Information Systems
Data Science
Autumn Semester 2023

Abstract

Neural networks have proven to be powerful tools in recent years, demonstrating their efficacy in solving real-life problems. In this task, we employ neural networks to predict the pay-back behavior of individuals, engaging in classification tasks. Our journey begins with a meticulous analysis of the data distribution and exploration of the number of classes to be predicted.

Subsequently, we embark on data preprocessing, a crucial step involving handling missing column values, one-hot encoding of categorical variables, and standardizing the data. These measures ensure the data is appropriately formatted and ready for training.

To optimize our predictive model, we experiment with two different architectures: Convolutional Neural Network (CNN) and Artificial Neural Network (ANN). The ANN were found to be the ones that performed the best so we went ahead on this type of architecture.

With the chosen ANN architecture, research has begun on the best hiperparameters such as batch size and dropout rate.

The final step involves the calculation of various metrics to comprehensively evaluate the performance of our model. Metrics such as accuracy, precision, recall, and F1-Score are computed, providing nuanced insights into the model's effectiveness in predicting pay-back behavior.

The project, was made using python instead of R, it is nothing but a different tool to achieve the same goal. This choice is due to the fact that we are more familiar and comfortable with this programming language.

Contents

Chapter 1

Credit Card Dataset

For this task, we utilize a version of the "A Credit Card Dataset for Machine Learning", which includes authentic (anonymized) data detailing information from customer applications along with their repayment patterns for credit card debts. The objective is to predict the likelihood of favorable repayment behavior by customers. It's important to note that the emphasis in this assignment is on forecasting customer behavior, rather than determining whether to accept or reject credit card applications. The original dataset is available on Kaggle at the following link.

1.1 Data Exploration

Unlike the dataset in task1, in this case it is not necessary to perform a business understanding of the features since they are very simple features to understand. At first the dataset was analyzing using the basic commands of the pandas library, these commands are useful to have a generic view of the dataset (mean and standard deviation of the features). To go into more detail in the features, the distributions of the various features were displayed, starting with the categorical ones.

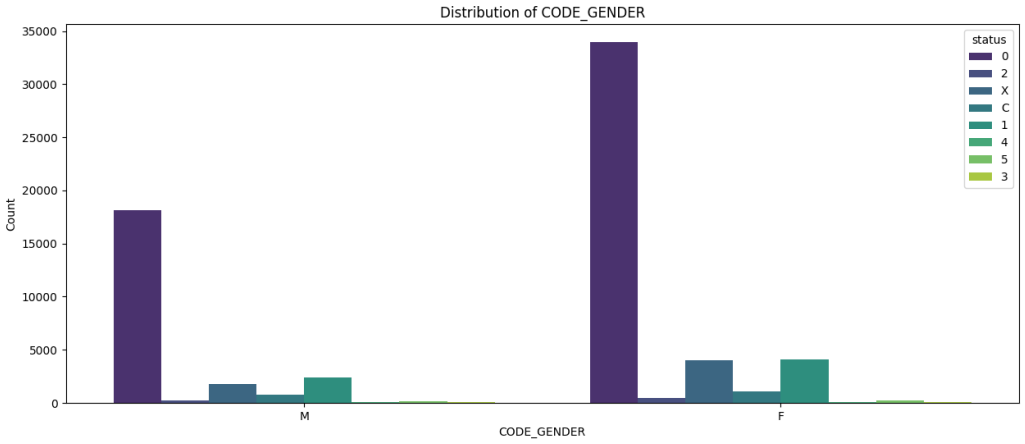

FIGURE 1.1: Distribution of CODE GENDER

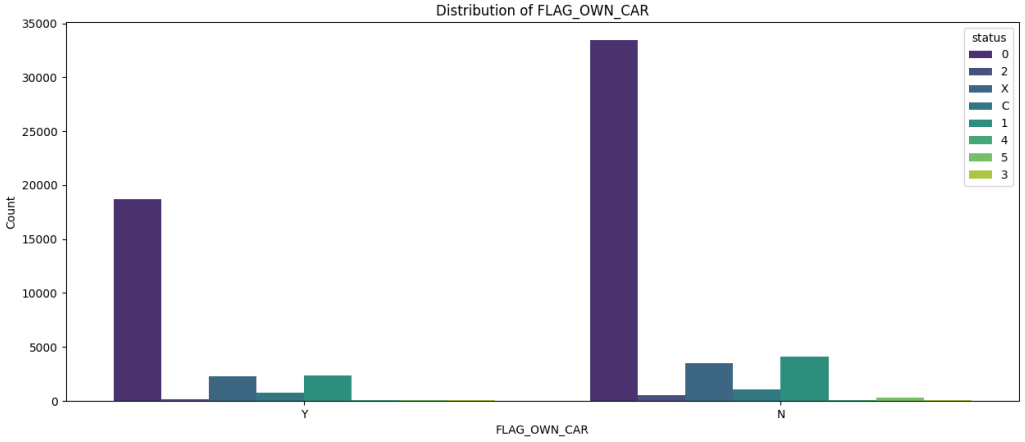

FIGURE 1.2: Distribution of FLAG OWN CAR

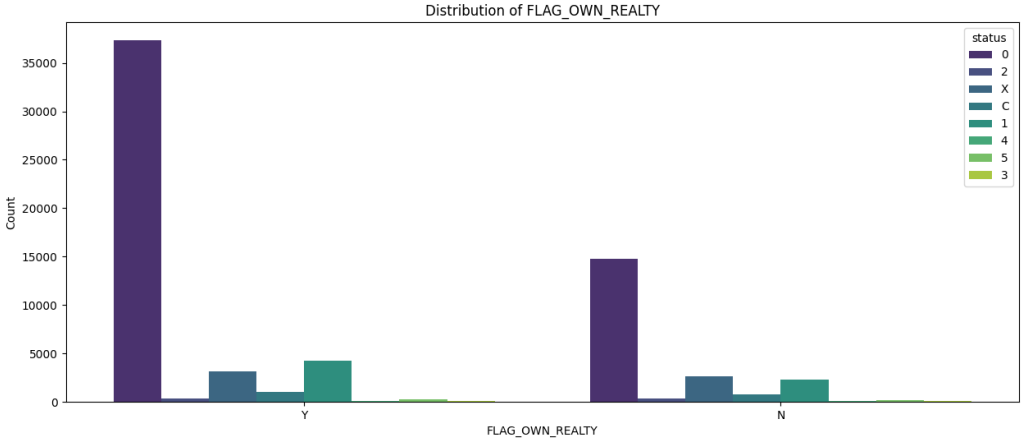

FIGURE 1.3: Distribution of FLAG OWN REALTY

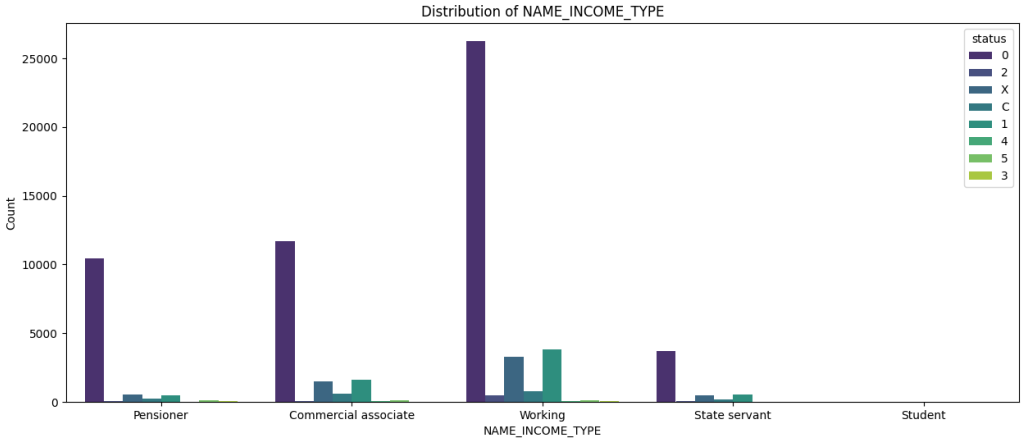

FIGURE 1.4: Distribution of NAME INCOME TYPE

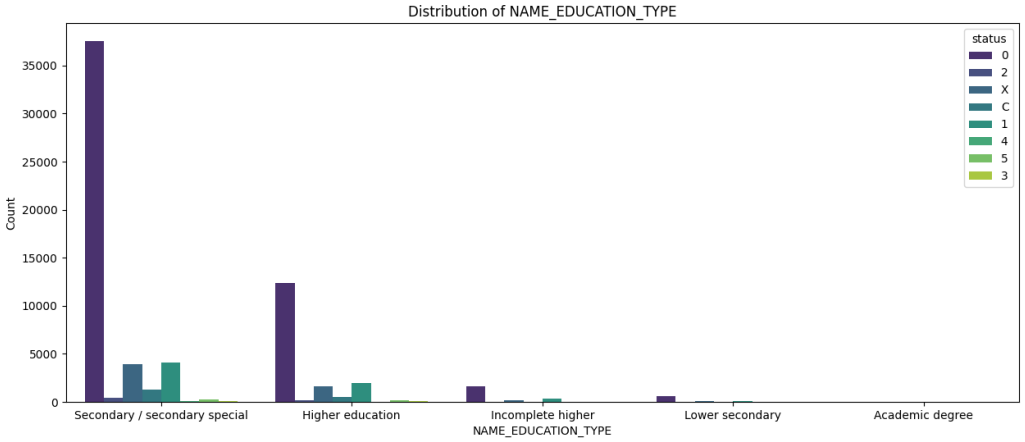

FIGURE 1.5: Distribution of NAME EDUCATION TYPE

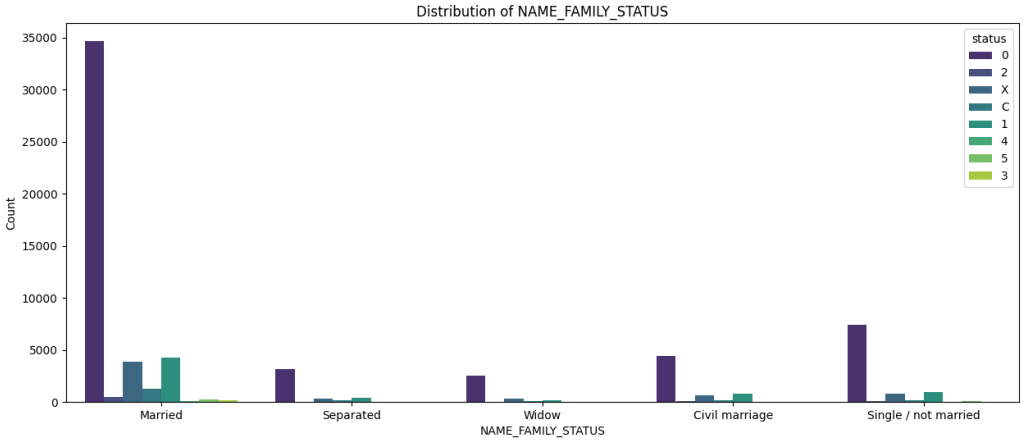

FIGURE 1.6: Distribution of NAME FAMILY STATUS

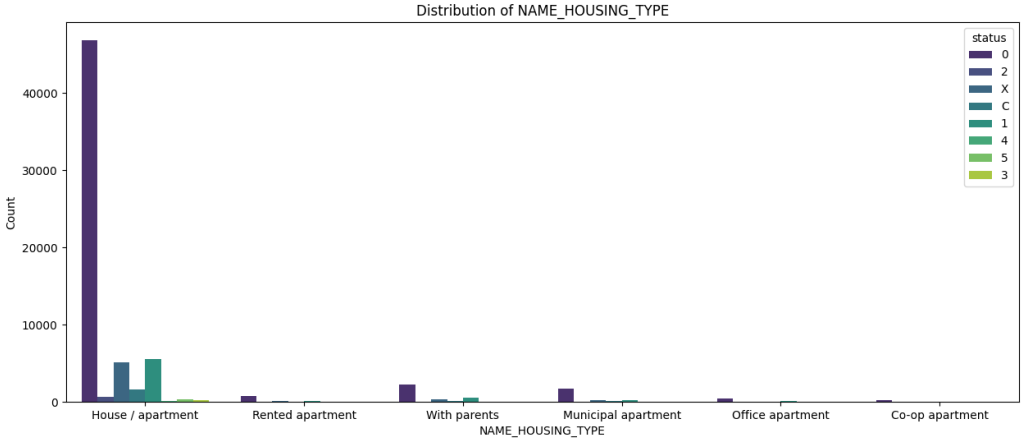

FIGURE 1.7: Distribution of NAME HOUSING TYPE

Upon initial examination, the most notable observation is that class 0 appears more frequently than the other classes, indicating a substantial imbalance in the dataset. This suggests that the distribution of classes is skewed, with one class (class 0) being predominant while the occurrences of other classes are comparatively lower. Imbalanced datasets can pose challenges for machine learning models, as they might struggle to effectively learn patterns from the minority classes. Specialized techniques and considerations, such as resampling methods or the use of appropriate evaluation metrics, may be necessary to address the imbalanced nature of the data and enhance the model's performance.

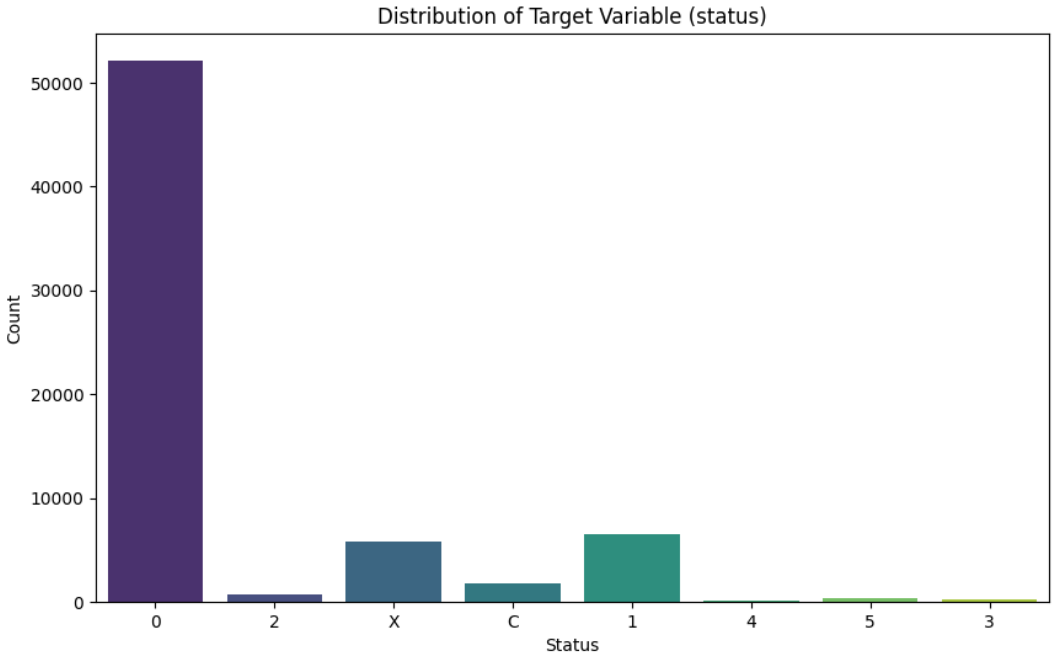

FIGURE 1.8: Distribution of status

Within this visual depiction, we gain insights into the distribution of class frequencies, reinforcing the acknowledgment of our dataset's pronounced imbalance, as highlighted earlier. It is important to note that classes 2, 4, 5, and 3 are heavily outnumbered by other classes. This notable class imbalance underscores the importance of employing strategies to address these disparities during the modeling process, ensuring fair and effective learning across all classes.

Chapter 2

Data Preprocessing

Preprocessing is vital in machine learning because it enhances data quality by handling missing or inconsistent data. Normalization and scaling ensure fair treatment of features, preventing one from dominating the learning process. Converting categorical data into a numerical format is crucial for many algorithms. Feature engineering allows the creation of new features or transformations for better pattern recognition. Dimensionality reduction speeds up training and prevents overfitting. Preprocessing also addresses class imbalance, ensuring fair representation of classes. Overall, preprocessing contributes to model robustness by preparing data for optimal learning and generalization. After prepocessing, the data consists of 55 features.

2.1 Handling Missing Values

The dataset contains missing values exclusively in the `OCCUPATION_TYPE` column. Instead of removing rows with these empty fields, we chose an alternative strategy. Specifically, we replaced the empty values within the `OCCUPATION_TYPE` column with the designated value `'jobless'`. This decision aims to retain the data instances associated with other features in the dataset while providing a meaningful placeholder for the absent occupation information. It allows us to preserve the integrity of the dataset and proceed with the analysis without discarding potentially valuable information due to missing values.

2.2 Target variable

The last attribute `status` contains the "pay-back behavior", indicating when the customer paid back their debts:

- 0: 1-29 days past due;
- 1: 30-59 days past due;
- 2: 60-89 days overdue;
- 3: 90-119 days overdue;
- 4: 120-149 days overdue;
- 5: Overdue or bad debts, write-offs for more than 150 days;
- C: Paid off that month;
- X: No loan for the month.

The target variable, which is a key part of our data used for making predictions, currently has a mix of both numbers and letters. To make it easier for our prediction system to understand and work with this information, we need to make sure it's all in the same format. Right now, we have a combination of numbers (like 0 or 1) and letters (like C or X). To simplify things, we're going to convert these letters into numbers, creating a consistent and uniform way of representing this important aspect of our data. This transformation helps our prediction system do its job more effectively. The mapping into numbers is done with the following code:

```
1 status_replacement = {'X': 7, 'C': 6}
2 df['status] = df['status'].replace(status_replacement).astype(int)
```

Listing 2.1: Replacing and Converting 'status' Values

2.3 One-hot Encoding

One-hot encoding is important in the context of machine learning when dealing with categorical variables. Categorical variables are features that can take on values from a limited and fixed set of categories. However, many machine learning algorithms work with numerical data, and directly using categorical variables with labels may introduce unintended ordinal relationships.

One-hot encoding is a technique where each unique category in a categorical variable is represented as a binary (0 or 1) value in a new column. Each category becomes a separate binary feature, and only one of these features is "hot" (set to 1) for a particular data point, indicating its category. This transformation ensures that the categorical information is properly represented in a format suitable for machine learning algorithms. The categorical columns we one-hot encoded are the following: `'CODE_GENDER'`, `'FLAG_OWN_CAR'`, `'FLAG_OWN_REALTY'`, `'NAME_INCOME_TYPE'`, `'NAME_EDUCATION_TYPE'`, `'NAME_FAMILY_STATUS'`, `'NAME_HOUSING_TYPE'`, `'OCCUPATION_TYPE'`.

2.4 Standardization

As for the numerical columns, we performed standardization. Standardization is a preprocessing technique that transforms numerical data to have a mean of 0 and a standard deviation of 1. This process is particularly useful when working with machine learning algorithms that are sensitive to the scale of features. The formula for standardization is given by:

$$z = \frac{x - \mu}{\sigma}$$

where z is the standardized value, x is the original value, μ is the mean of the column, and σ is the standard deviation of the column. The numerical columns that were standardized are: `'AMT_INCOME_TOTAL'`, `'CNT_FAM_MEMBERS'`, `'CNT_CHILDREN'`, `'DAYS_BIRTH'`, `'DAYS_EMPLOYED'`. As learned in class, it is important to standardize the testing dataset using the values the mean and the standard deviation obtained from the training dataset.

2.5 Oversampling

The last step of preprocessing concerns oversampling. As mentioned earlier, the dataset is unbalanced. To address this issue, we initially attempted to undersample the majority class. However, the results were less than satisfactory, and the model's performance was adversely affected.

In response, we explored another technique—oversampling the classes with fewer occurrences, specifically classes 2, 3, 4, and 5. The oversampling technique employed was Synthetic Minority Over-sampling Technique (SMOTE). Unlike undersampling, SMOTE enhances the dataset by generating synthetic instances of the minority classes. This allows the model to learn from a more balanced representation of the data.

SMOTE works by creating synthetic examples along the line segments connecting existing minority class instances. This technique mitigates the risk of the model being biased toward the majority

class, as it introduces additional instances of the minority classes without merely duplicating existing ones. The resulting dataset, now more balanced, contributes to the improved performance of the final model. This approach, in contrast to undersampling, proved effective in addressing the challenges posed by the imbalanced dataset.

2.6 Featuring Extraction

In the pursuit of this task, we derived additional features to augment the existing dataset:

$$\text{INCOME\_PER\_FAMILY} = \frac{\text{AMT\_INCOME\_TOTAL}}{\text{CNT\_FAM\_MEMBERS}}$$

This feature is calculated by dividing the total income (AMT_INCOME_TOTAL) by the count of family members (CNT_FAM_MEMBERS). Essentially, it provides an estimate of the individual contribution or access to income within the family, accounting for its size.

$$\text{TOTAL\_BALANCE\_CHILDREN} = \text{INCOME\_PER\_FAMILY} \times (1 + \text{CNT\_CHILDREN})$$

Expanding on the previous feature, this calculates the total balance for each family. It involves multiplying INCOME_PER_FAMILY by a factor that considers the number of children (CNT_CHILDREN). The addition of (1 + CNT_CHILDREN) signifies an adjustment for families with children, capturing the overall financial capacity while factoring in both the income per member and the presence of children.

However, despite their intuitive appeal, during the model training, these features did not significantly contribute to the model's performance so they were not included in the final preprocessing.

Chapter 3

Hyperparameter Optimization

The parameters we aimed to optimize were batch size and dropout rate. We employed the technique of Grid Search Cross-Validation (GridSearchCV) with 10 folds (as suggested in the assignment) to systematically explore different combinations of these parameters. As wen can see in Table 3.1, the batch size values we experimented with were 512, 1024, and 2048, while the dropout rate values were 0.1, 0.2, and 0.3.

Batch Size	**Dropout Rate**
512	0.1
1024	0.2
2048	0.3

Table 3.1: Parameter Combinations Explored

We intentionally limited the number of parameters to optimize due to time constraints. The training time increases significantly with a larger parameter space. Additionally, we experimented with larger batch sizes (such as 1024 and 2048) for two reasons: first, to improve training efficiency, and second, to attempt to include instances from all classes in each batch. This strategy aims to enhance the model's learning by exposing it to representatives of all classes during each training batch.

GridSearchCV is a technique that exhaustively searches through a predefined hyperparameter space to find the combination that yields the best model performance. In our case, it explored various combinations of batch size and dropout rate using a cross-validation approach with 10 folds.

The term "folds" refers to the division of the dataset into a specified number of subsets (in this case, 10 folds). During cross-validation, the model is trained and evaluated 10 times, each time using a different fold as the testing set and the remaining folds as the training set. This process ensures that the model is tested on different portions of the data, helping to provide a more robust

estimate of its performance across various scenarios. Additionally, after thorough exploration, the best hyperparameters we found were a batch size of 512 and a dropout rate of 0.2. These values yielded optimal model performance and were chosen based on their ability to enhance training efficiency and capture diverse patterns across all classes.

As mentioned earlier, time and computational power did not allow us to perform a better search for hyperparameters; nevertheless, the following list encapsulates the hyperparameters we would have liked to have tried:

- **Batch Size:** Integer values from 4 to 2048 in logarithmic scale;
- **Learning Rate:** values from 1×10^{-7} to 1×10^{-1} in logarithmic scale;
- **Optimizer:** "Adam", "AMSGrad", and "SGD";
- **Different Architectures:** exploring various combinations of the number of layers and neurons;
- **Activation Functions:** experimenting with different activation functions.

Chapter 4

Training

After finding the best hyperparameters we were looking for, we used them in the neural network. The neural network architecture is as follows:

```
model.add(Dense(units=256, activation='relu', input_dim=55))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=32, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=16, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(units=8, activation='softmax'))
```

The optimizer used is a variant of Adam known as AMSGrad. AMSGrad stands for "Accumulate, Move, and Scale Gradien". It is an enhancement to the Adam optimization algorithm designed to address potential issues with the adaptive learning rate method.

BatchNormalization is a technique that normalizes the inputs of a layer, aiming to improve the training stability and speed. It helps the network learn more efficiently by reducing internal covariate shift.

Dropout is a regularization technique where randomly selected neurons are ignored during training. This helps prevent overfitting by introducing redundancy and making the network more robust.

Rectified Linear Unit (ReLU) is an activation function that introduces non-linearity to the network. It replaces all negative values with zero, allowing the network to learn complex patterns.

The training process involved creating a validation dataset before starting the training. The network was trained for 20,000 epochs, and at the end of the training, the weights corresponding to the best performance on the validation set were restored.

Fig. 4.1 illustrates the error progression during the training of both the training and validation datasets. As depicted, there is still room for improvement in reducing the error by increasing the number of epochs. However, due to constraints in time and computational resources, it was not feasible to extend the training further.

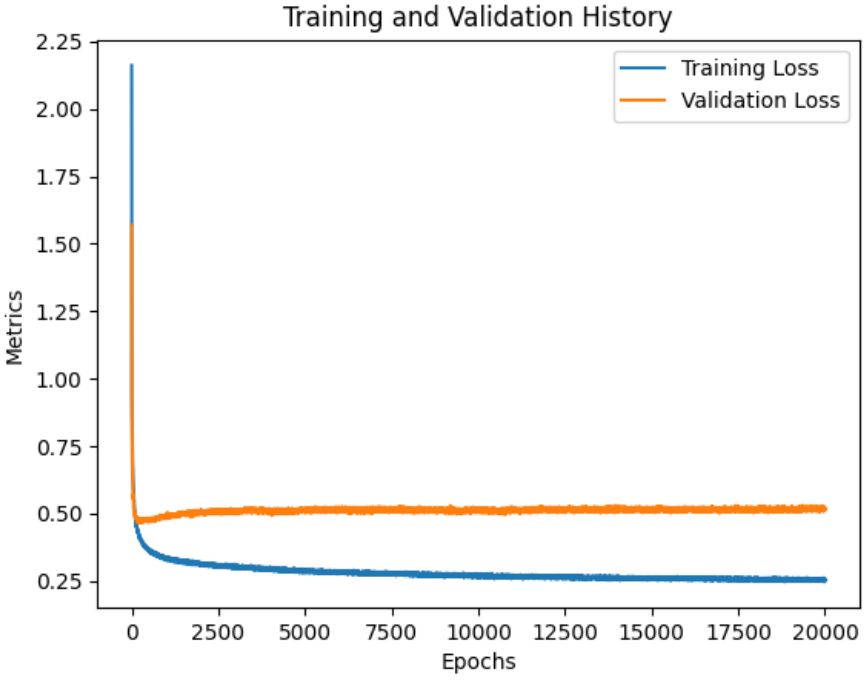

FIGURE 4.1: Error Progression During Training

The x-axis represents the number of epochs, while the y-axis denotes the error on both the training and validation datasets. The maximum accuracy achieved during the training phase is 91.68% on the train dataset and 86.98% on the validation dataset. The plot demonstrates that the model could potentially benefit from additional training epochs, leading to further refinement in its performance. However, due to practical limitations, the training was stopped at a certain point.

Increasing the number of epochs can be advantageous for complex models, allowing them to capture more intricate patterns in the data. In situations with more computational resources and time availability, exploring a higher number of epochs could be considered to achieve even better model convergence.

4.1 Alternative Training Methods

In addition, we also tried a CNN architecture. To do this, we mapped the data available to us into images so that we could leverage the capabilities of a CNN. Despite the effort, the results were not satisfactory due to a large overfitting of the network, the maximum accuracy achieved by this model on the training phase is 89.71% on the train dataset and 79.87% on the validation dataset, the idea was consequently discarded.

In our pursuit of enhancing the training phase, we explored an additional approach known as *class weights*. Class weights play a crucial role in addressing imbalances within the dataset, especially when dealing with uneven class distributions. In a classification task, the concept of class weights acknowledges that not all classes contribute equally to the overall objective. Imbalanced datasets, where certain classes have fewer instances than others, can lead the model to prioritize the majority class during training, potentially neglecting the minority classes. Class weights provide a mechanism to mitigate this imbalance by assigning different weights to each class. Essentially, higher weights are assigned to under-represented classes, signaling to the model that these classes are more critical and should be given more attention during the learning process. Although everything is very nice in theory, in practice, it did not work for us and this method was also discarded

Chapter 5

Results

In this chapter, we present and analyze the outcomes derived from our trained model evaluated on the test dataset. The evaluation serves as a critical examination, providing valuable insights into the model's performance, generalization capabilities, and its aptitude to make accurate predictions on unseen data.

Throughout the following sections, we explore diverse performance metrics, shedding light on the model's accuracy, precision, recall, and F1-Score. Together, these metrics serve to evaluate the model from several perspectives.

Test Accuracy and Loss

The model achieved a test accuracy of 84.83%, indicating that it correctly classified 84.83% of the test samples. The test loss, a measure of the model's prediction error, is 0.6086, suggesting relatively good performance.

Metric	**Value**
Accuracy	0.848332
Precision	0.836593
Recall	0.848332
F1-Score	0.837497

Table 5.1: Metrics Table

Analysis of Metrics

Analyzing the metrics table provides insights into the model's performance:

- **Accuracy:** The overall correctness of the model's predictions is approximately 84.83%;
- **Precision:** Out of the instances predicted as positive, around 83.66% are true positives;
- **Recall:** The model captures around 84.83% of all actual positive instances;
- **F1-Score:** A balance between precision and recall is reflected in the F1-Score of 83.75%.

These metrics collectively provide insights into the model's effectiveness in classification tasks.

5.1 Confusion Matrix

The confusion matrix provided in Fig. 5.1 summarizes the performance of a classification model. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class

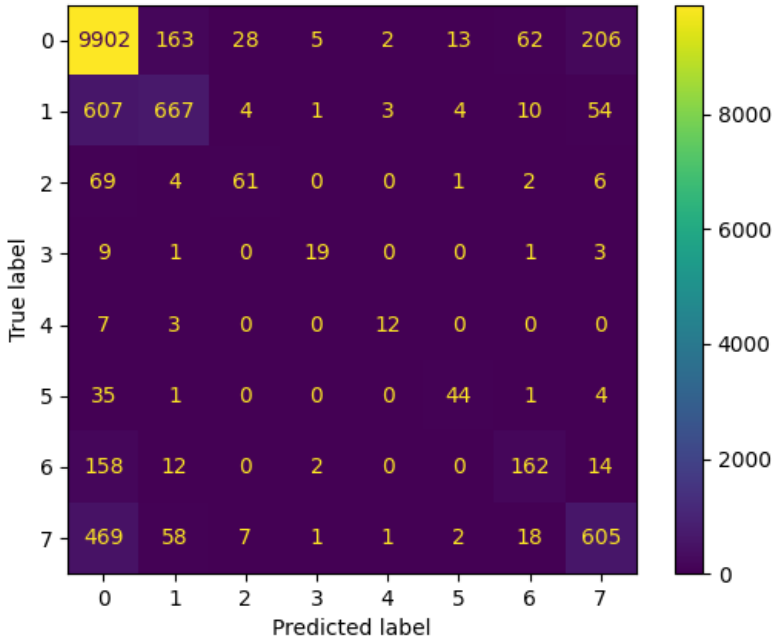

FIGURE 5.1: Confusion Matrix

- **True Positives (TPs):** Values along the diagonal (9902, 667, 61, 19, 12, 44, 162, 605);
- **True Positives (FPs):** Sums of each column excluding the diagonal (163 + 4 + 1 + 0 + 3 + 1 + 12 + 58);
- **False Negatives (FNs):** Sums of each row excluding the diagonal (28 + 4 + 0 + 0 + 0 + 0 + 2 + 7);
- **True Negatives (TNs):** Values outside the diagonal.

Other models

In addition to the presented results for the final model, we tested other models on the validation dataset. Table 5.2 summarizes the accuracies achieved by each model:

Model	**Accuracy**
Final Model	84.83%
CNN Model	79.87%
Dummy Model	77%

Table 5.2: Model Accuracies on Validation Dataset

Explanation:

The final model achieved an accuracy of 84.83%, indicating its ability to correctly classify instances on the validation dataset. The CNN model, while still performing well, exhibited a slightly lower accuracy of 79.87%.

The Dummy Model, on the other hand, is a baseline model that always predicts the majority class. In this case, since the dataset is unbalanced, the majority class is likely the class labeled as 0. Consequently, the Dummy Model achieves a relatively high accuracy of 77%, emphasizing the importance of considering class distribution when interpreting accuracy.

Chapter 6

Conclusions

Neural networks stand out as one of the most potent machine learning models, capable of adapting to diverse contexts, ranging from medical applications to financial analyses. The versatility of neural networks lies in their ability to capture complex patterns and relationships within data.

The primary metrics employed to evaluate the effectiveness of our model include accuracy, precision, recall, F1-Score, and the confusion matrix. These metrics provide a comprehensive understanding of the model's performance from various perspectives.

While the model exhibits a good performance on the training dataset, it is crucial to note that the performance on the test dataset is somewhat diminished. This is expected as the test dataset comprises data the model has never encountered during training.

Although satisfactory, the accuracy obtained, can still be improved by making the training process last for several epochs. Although this has not been possible, We can conclude by saying that we are satisfied with the results obtained, the task to be carried was not easy and the unbalanced dataset made things more difficult but nevertheless we knew how to overcome the difficulties.

Chapter 7

Lessons Learnt

The things learned during the duration of the course are many, some of which were not possible to apply in this task. The task is undoubtedly interesting and although it did not seem so at first glance, it turned out to be more difficult than we expected. Nevertheless, we learned how neural networks require a great deal of computing power and time.

The biggest obstacle was undoubtedly the optimization of hyperparameters as with only a few combinations, the time to perform the search grew disproportionately. We tried different approaches to accomplish this task, such as cnn which in other projects we have done proved to be very useful. The idea was to find a way to map the data we had available in images and then use a cnn-based architecture, although it seemed like a good idea, the idea was quickly discarded because of the bad results.

One thing we would have liked to try would have been an rnn architecture but we did not know how to set up the problem. Finally we decided to use a classical architecture, which turned out to be the best choice. In conclusion we can say that this task allowed us to get our hands on many aspects we learned in class, we are satisfied both with the results we got and with the task we were given as applying the things we learned in class allows us to better understand how it all works.

Abbreviations

AMSGrad Accumulate, Move, and Scale Gradien. 13

ANN Artificial Neural Network. i

CNN Convolutional Neural Network. i, 14, 17

FN False Negative. 17

FP True Positive. 17

GridSearchCV Grid Search Cross-Validation. 10

ReLU Rectified Linear Unit. 13

SMOTE Synthetic Minority Over-sampling Technique. 8

TN True Negative. 17

TP True Positive. 17

List of Figures

List of Tables