UNIVERSITÀ di VERONA

github.com/dariogemo/UNIVR-machinelearning

Confronting Deep Learning methods for tabular data: application on faulty steel plates

Dario Gemo and Lorenzo Squassoni

This project was carried out during the Machine Learning for Data Science Course taught within the master's degree program Data Science, University of Verona.

Abstract

In this project, different machine learning models will be implemented to classify defects in steel plates. Following an exploratory data analysis, which highlights a significant class imbalance, different techniques for synthetic data generation will be applied. After evaluating traditional machine learning models for classification tasks, the primary challenge of this study is to achieve high accuracy by adapting conventional neural network architectures, which are typically less suited for tabular data. Ultimately, the findings will demonstrate that with appropriate adjustments, these models can also achieve a competitive accuracy.

1. INTRODUCTION

In this final project for the Machine Learning for Data Science course, we have explored and extended various models covered during the lessons, applying them to an industrial problem. Our primary challenge involved implementing neural networks for tabular data, which is a quite unusual approach, since neural networks are typically employed for other types of data. The dataset used in our study, Steel Plates Faults (available on the UCI Machine Learning repository: https://archive.ics.uci.edu/dataset/198/steel+plates+faults), aims at accurately classify surface defects in stainless steel plates, encompassing seven distinct defect types. The input vector consists of 27 indicators that approximately describe the geometric characteristics of the defects and their shapes. However, since no detailed information is provided regarding the nature of these indicators, the following variable definitions are based on our own assumptions.

- **'X_Minimum', 'X_Maximum', 'Y_Minimum', 'Y_Maximum'**: coordinates of the bounding box of the defect in the image.
- **'Pixels_Areas'**: represents the total number of pixels that make up the defect.
- **'X_Perimeter', 'Y_Perimeter'**: represent the perimeter of the defect in the X and Y directions.
- **'Sum_of_Luminosity', 'Minimum_of_Luminosity', 'Maximum_of_Luminosity'**: represent the brightness of the defect. The sum could be the total brightness, while the minimum and maximum could be the darkest and brightest points.
- **'Length_of_Conveyer'**: feature related to the production process, such as the length of the conveyor belt used in manufacturing.
- **'TypeOfSteel_A300', 'TypeOfSteel_A400'**: categorical features indicating the type of steel used in the plate.
- **'Steel_Plate_Thickness'**: numerical feature indicating the thickness of the steel plate
- **'Edges_Index', 'Empty_Index', 'Square_Index', 'Outside_X_Index', 'Edges_X_Index', 'Edges_Y_Index', 'Outside_Global_Index'**: indices related to the shape and location of the defect.
- **'LogOfAreas', 'Log_X_Index', 'Log_Y_Index'**: logarithmic transformations of 'Pixels_Areas', 'X_Index', and 'Y_Index'.
- **'Orientation_Index'**: feature indicating the orientation of the defect.
- **'Luminosity_Index'**: index related to the brightness of the defect.
- **'SigmoidOfAreas'**: sigmoid transformation of the 'Pixels_Areas' feature.
- **'Pastry', 'Z_Scratch', 'K_Scatch', 'Stains', 'Dirtiness', 'Bumps', 'Other_Faults'**: feature indicating the presence or absence of specific types of defect. **This is the feature we want to predict**.

In order to assess the performance of our models, we have taken as benchmark the results from more established architectures in the field. It is important to remark that in

this field the best performing methods for tabular data until now, are boosting methods like like XGBoost, CatBoost and LightGBM. However, since these type of models where not covered during our Machine Learning university course, we decided to not deploy them. Overall, we have implemented the following models:

- Naive Bayes classifier
- Support Vector classifier (SVC)
- Hypertuned SVC
- Hypertuned Feed Forward Neural Network
- FFNN with dimensionality reduction through an AutoEncoder
- SVC with dimensionality reduction through an AutoEncoder
- TabNet
- FTTransformer

As demonstrated in the next chapter, one of the primary challenges that needs to be addressed involves fixing a strong class imbalance, which will be mitigated through the adoption of the Smontec technique in combination with a CTGan for generating synthetic data. This approach ensures the creation of a well-representative dataset, which will be particularly beneficial when applying DL models. By increasing the volume of input data, these models will have a richer foundation to learn underlying patterns, ultimately enhancing their performance in defect classification. We summarize the main insights of this project as:

1. Solving the problem of class imbalance with Smontec + CTGan
2. Showing that that comparable to the ones of traditional machine learning algorithms can be achieved with neural networks on tabular data

2. DATA EXPLORATION AND PREPROCESSING

2.1 Data preprocessing: fixing class imbalance and generate synthetic data

2.1.1 Smontec

When analyzing the distribution of steel plates defects in our data, we immediately face an important issue: a significant class imbalance. The three most common defect types collectively represent 75% of the total instances, which could negatively impact the performance of our models in the classification task. In fact, having too few observations of the minority class is an obstacle for model to effectively learn the decision boundaries.

	count
pastry	158
z_scratch	190
k_scatch	390
stains	72
dirtiness	55
bumps	402
other_faults	673

Fig. 1. Count of the steel plates defect types.

In order to fix this problem, we have applied an over-sampling technique called Smote, in its version able to handle both numerical and categorical features **Smontenc**[1]. Once applied Smotenc, the number of instances for each class has been leveled to the most represented class (other faults with 673 observations) by creating synthetic data. These synthetic data are created by placing them randomly on the interpolation of the k closest instances of each class.

2.1.2 Conditional Tabular Generative Adversarial Networks -CTGAN-

After balancing the classes, we have decided that 676 instances for each class might still be insufficient to train the models. Therefore, we opted to generate additional synthetic data using **CTGAN** [4] to enhance the dataset. This method is an adaptation of general adversarial networks (GANs), usually applied for image generation, for tabular data generation. The main task of synthetic data generation consist in training a data synthetizer G to learn from the original table T and than using G to generate a new synthetic table T_{syn}. Each column, being either continuous C_i or a discrete D_i, is treated as a random variable. All these columns together, $\{C_1, \ldots, C_{Nc}\}$ and $\{D_1, \ldots, D_{Nd}\}$ have an unknown joint distribution and each row $\mathbf{r}_j = \{c_{1,j}, \ldots, c_{N_c,j}, d_{1,j}, \ldots, d_{N_d,j}\}, \quad j \in \{1, \ldots, n\}$ is interpreted as one observation from this joint distribution. The data synthesizer G is trained on T_{train} and then the new data table T_{syn} is built by independently sampling rows using G. The CTGAN model will have good performance if the columns of T_{syn} have the same joint distribution of T_{train}. This model has two main steps:

1. **Mode-specific normalization**: before training the neural network that generates synthetic data, all values belonging to different column types must be normalized. While discrete values can be easily express with one-hot encoding, a different approach must be followed for continuous variables. For each column C_i we use variational Gaussian mixture in order to estimate the number of modes m_i. Than, for each value $c_{i,j}$ in the column C_i we compute its probability of coming from each existing mode m_i. Supposing that the value $c_{i,j}$ belongs to the third and last mode, it will be represented as the concatenation of the one-hot-encoding (telling to which mode it belong)

$$\beta_{i,j} = [0, 0, 1]$$

and a scalar

$$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$$

which is the original value normalized with the mean and standard deviation of the corresponding mode.

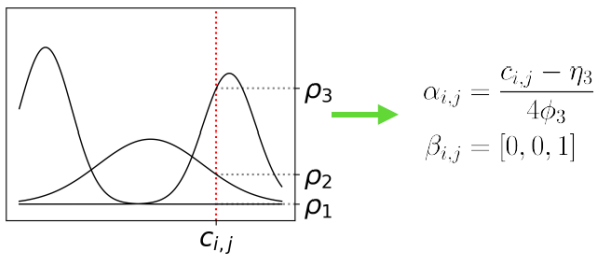

Fig. 2. Mode specific normalization, L. Xu *et.al* (2019).

Therefore, each observation (each row of the table) is expressed as the concatenation of all normalized continuous variables and the one-hot representation $d_{i,j}$ of discrete variables.

$$\mathbf{r}_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \cdots \oplus \alpha_{N_c,j} \oplus \beta_{N_c,j} \oplus \mathbf{d}_{1,j} \oplus \cdots \oplus \mathbf{d}_{N_d,j}$$

2. **Conditional generation of synthetic data**: in order to generate reliable synthetic data, the model takes a specific value k^* of a randomly selected discrete column D_i (i.e. bumps for the column steel defect), and it learns the distribution of other features given this condition. The final aim is to have as similar as possible the conditional joint distribution of original data and the conditional joint distribution of new synthetic data $\hat{r}$.

$$\mathbb{P}_G(\text{row} \mid D_{i_*} = k^*) = \mathbb{P}(\text{row} \mid D_{i_*} = k^*),$$

During training, the generator can produce various outputs for discrete variables, which are encoded as one-hot vectors. To ensure that it generates realistic values for these discrete columns, it incorporates in its loss the cross-entropy between the generated and real values. This guides the generator to align its outputs with the true data distribution over time. Finally, the output is assessed by the **critic** which estimates the distance between the learned conditional distribution $P_G(\text{row} \mid D_{i_*} = k^*)$ and the conditional distribution on real data $P(\text{row} \mid D_{i_*} = k^*)$.

At the conclusion of the data generation process, we obtained 2000 observations for each defect type class. Through comprehensive visual analysis, we confirmed that the synthetic data accurately retains the key characteristics of the original dataset.

	count
pastry	2000
z_scratch	2000
k_scatch	2000
stains	2000
dirtiness	2000
bumps	2000
other_faults	2000

Fig. 3. Count of the steel defect type after CTGAN.

An example of the validness of the synthetic data, we can take the class of faults "k_scratch". From figure 4 we have a visual representation that the correlation between the features, so the relationships and the patterns that the variables have between each others, are retained by the synthetic data generated through CTGAN. Moreover, from figure 5 we see that the new data also maintains basic statistical characteristic of the real data.

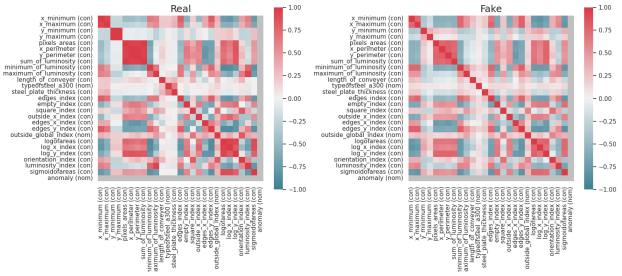

Fig. 4. Correlation heatmap for original and synthetic data.

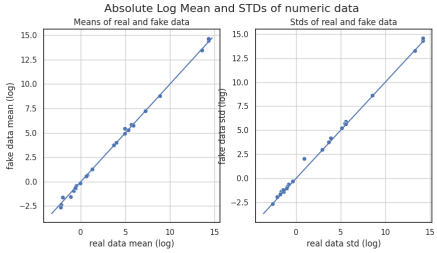

Fig. 5. Logarithmic mean and standard deviation scatterplot of original vs synthetic data.

2.2 Feature correlation and data standardization

Given the large number of variables, our exploratory data analysis begins by examining the correlation between features. Many variables exhibit high correlations, with several exceeding the commonly used threshold of 0.33. Notably, some features display a perfect linear relationship, such as the coordinates of the bounding box defining the defect (x_minimum - x_maximum; y_minimum - y_maximum). These findings suggest the presence of redundant information, which may necessitate dimensionality reduction techniques to enhance models efficiency.

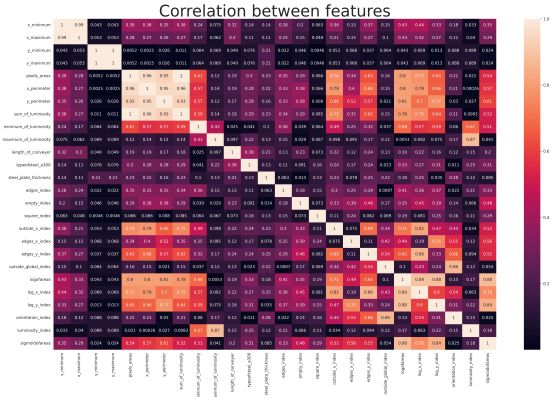

Fig. 6. Correlation matrix.

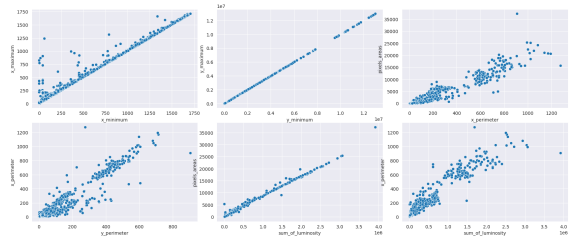

Fig. 7. Correlation scatter-plots of most correlated features.

From figure 7 we clearly notice a strong linearity between some features. Unfortunately, except x_minimum/x_maximum and y_minimum/y_maximum, all the other features are on very different range of data. This means that we cannot use simple mean for aggregating some of these features. Because of this, and also because usually machine learning models benefit from scaled data, we have normalized the continuous features so that they have mean 0 and standard deviation 1.

Following up we have aggregated, using simple mean, those features that presents a very high Pearson correlation (higher than 95%). After features aggregation we have obtained the correlation heatmap in figure 8, where no outstanding high correlation between features can be noticed.

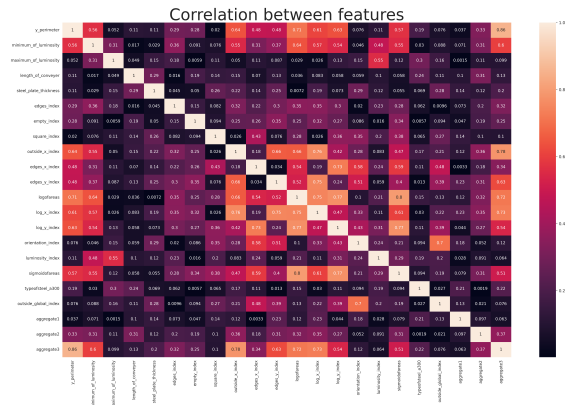

Fig. 8. Correlation matrix after features aggregation.

2.3 Visualizing the classes

To gain more information about the dataset, and specifically about the classes we are trying to predict, we have tried different dimensionality reduction techniques in order to visualize clusters of data.

- **PCA**: reduce dataset's dimensionality into k uncorrelated new variables called principal components. It assumes that the **relationships between variables are linear** (principal components are a linear combination of the original features).
- **t-SNE**: it is a **non-linear** dimensionality reduction technique. Given the desired k dimensions, t-SNE ensures that the probability of two points being close together in the low-dimensional space closely matches their probability of proximity in the high-dimensional space.
- **UMAP**: it is a **non-linear** dimensionality reduction technique similar to t-SNE. It first build a weighted graph, with edge weights representing the likelihood that two points are connected, then such structure is replicated into the lower k-dimensionality space.

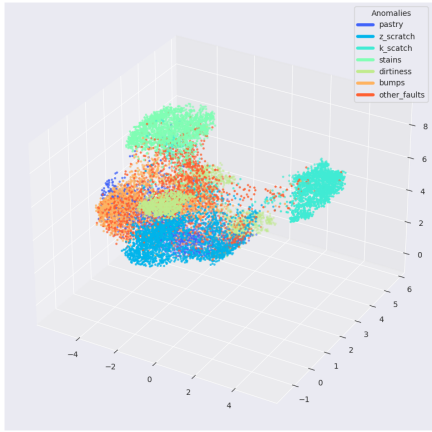

Fig. 9. UMAP dimensionality reduction with 3 components.

From our analysis we have observed that UMAP and t-SNE achieve better results than PCA in the task of separating steel defects in 3-dimensionality space. This implies strong non-linearity in the correlation between the features. From figure 9 we clearly see that some classes are easily differentiable from the others; z_scratch and k_scratch create very clear clusters of points, differently from pastry, bumps and other_faults where their respective clusters overlap each other. This could mean that our classification models could face problems when it comes to these latter classes.

3. TRADITIONAL MODELS

In order to have some sort of benchmark with which we can compare the results of more recent Deep Learning models, we first use some more established methods in the industry to predict the steel plates faults and calculate some metrics on their results.

To better generalize the results that we ended up with, we used 5-fold cross validation and then calculated the mean accuracy, precision, recall and f1-score between folds. This should ensure robust results for models' effectiveness, meaning that they are not some sort of random outcome of the specific train/test split.

3.1 Naive Bayes Classifier

Given the fact that it is a widespread easy-to-implement solution and it is an easily understandable deterministic model, we decided to use Naive Bayes Classifier as our first benchmark model. Even though it one of the models with the lowest training time and it requires very little computational power, we clearly see from table 1 that it produces poor predictions.

Accuracy	Precision	Recall	F1-score
79.67%	80.59%	79.67%	79.56%

Table 1. Scores for Naive Bayes Classifier.

In chapter 2 we have noticed thanks to t-SNE and UMAP visualizations that some classes (specifically classes pastry bumps and other faults) aren't easily differentiable. This can be seen in figure 15 representing the confusion matrix for fold 5 of the NB Classifier, where the model makes some mistakes for classes 5 (bumps) and 6 (other faults); this will be later confirmed by the confusion matrices and classification reports of other models as well.

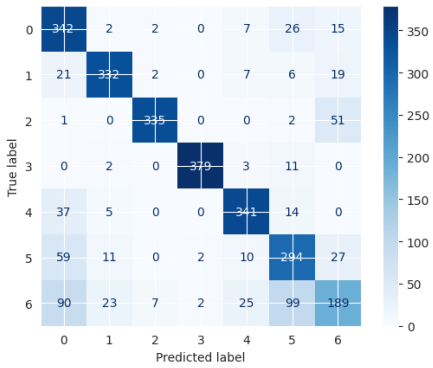

Fig. 10. Confusion matrix for fold 5 of Naive Bayes Classifier.

3.2 Support Vector Classifier

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that best separates data points of different classes in a high-dimensional space. The goal is to maximize the margin between the classes, with the support vectors being the data points closest to the hyperplane. We have implemented first the classifier with standard parameters from Scikit-learn, testing how it performed without tuned parameters. The main advantage of this approach is that training takes very little time, while giving somewhat accurate predictions that are summarized in table 2.

Accuracy	Precision	Recall	F1-score
91.96%	91.96%	91.95%	91.89%

Table 2. Scores for standard SVM.

However, to test if this results could be improved, we have tried tuning the classifier parameters with a grid search. Particularly, we have tried to tune the value of the regularization parameter **C** and the the kernel coefficient γ for the radial basis function kernel. The results of this hyper-parameter tuning are shown in table 3, where we can see that the predictions have greatly improved. Unfortunately, this gain comes at a price of a greater training time, arriving at 384 seconds of average training time per fold.

Accuracy	Precision	Recall	F1-score
92.54%	92.54%	92.54%	92.5%

Table 3. Scores for tuned SVM.

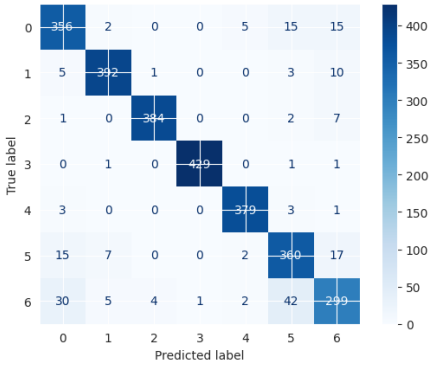

Fig. 11. Confusion matrix for fold 5 of tuned SVM.

4.DL MODELS

For the Deep Learning part of the project we have decided to implement more basic Deep Learning architectures like a Feed Forward Neural Network and an AutoEncoder linked with a FFNN, and more modern approaches to tabular data. In particular, we have implemented the TabNet model developed in 2019, and an implementation of the transformers architecture with a feature tokenizer developed in 2021.

As done in previous chapters, in order to compare different models we check the average accuracy, precision, recall and f1-score of the folds in a 5-fold cross validation.

4.1 Feed Forward Neural Network

First of all, a simple Feed Forward Neural Network has been tested as a sort of "benchmark" for later models, to see if they can actually outperform the results of the most basic DL architecture. To achieve competitive results, we have tuned the parameters of the network with the optuna library, exploiting the fact that it can early stops trials if they show unpromising results, effectively reducing the computation needed. In this regard, also early stopping has been included in the architecture, reducing even more computation time. After finding the best architecture, the model is trained on the train set for each fold and makes predictions on the test set for each fold, actually creating 5 different models for each fold; this should give us the most accurate result for this parameter tuning approach.

After some trials, we have decided that the parameters object for tuning are the number of layers in the network, the number of neurons in the layers, the dropout rate and the learning rate for the Adam optimizer. A better understanding of the hypertuning can be achieved by looking at table 4.

N. early layers	N. neurons early layers	Dropout early layers	N. later layers	N. neurons later layers	Learn. rate
$\{1, 2, 3\}$	64-128, step 8	$[0.1, 0.3]$	$\{1, 2, 3\}$	8-64, step 8	$[1e^{-5}, 1e^{-2}]$

Table 4. Parameters tuned for FF.

The last layer will be a simple dense layer with softmax activation function and 7 neurons, that is used for the classification.

With this approach we were able to achieve very good results for this model, shown in table 5.

Accuracy	Precision	Recall	F1-score
92.41%	92.47%	92.43%	92.33%

Table 5. Scores for tuned FFNN.

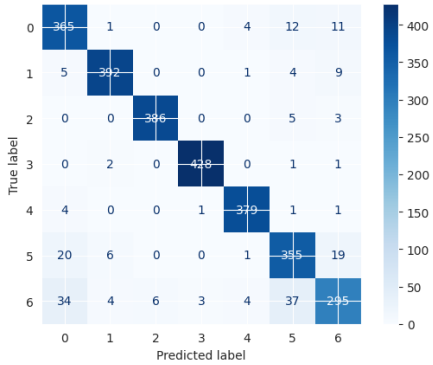

Fig. 12. Confusion matrix for fold 5 of tuned FFNN.

4.2 AutoEncoder dimensionality reduction

With the following model we have tried to implement the architecture for an AutoEncoder used for dimensionality reduction and a simple FFNN used for classification. First of all, the AutoEncoder is formed by an encoder with the latent features layer composed by 15 neurons, and by a decoder which tries to perfectly reconstruct the initial input. In this way we can effectively perform a reduction of the dimensions of the initial dataset to 15. On top of this, a FFNN is linked to the latent features layer which takes the compressed dataset as input and tries to correctly classify the faults classes. For this goal, the FFNN takes as last layer a dense layer with 7 neurons and a softmax activation function.

The idea behind this type of model is to train the AutoEncoder to compress the dataset to fewer dimensions, while preserving as much as possible the classificability of the classes performed by the FFNN.

In this setting, the loss function for this model is composed by the loss of the AutoEncoder plus the loss of the FFNN. The final loss function is defined as:

$$L_{\text{final}} = L_{\text{AE}} + L_{\text{FFNN}} \tag{1}$$

where:

$$L_{\text{AE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2} \tag{2}$$

is the Root Mean Squared Error (RMSE) loss for the AutoEncoder, and

$$L_{\text{FFNN}} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{3}$$

is the Categorical Cross-Entropy (CCE) loss for the FFNN.

Accuracy	Precision	Recall	F1-score
85.42%	87%	85.41%	84.26%

Table 6. Scores for AutoEncoder FFNN.

The model is then trained, and the FFNN is used to predict the faults classes. The metrics for the predictions of this architecture are displayed in table 6.

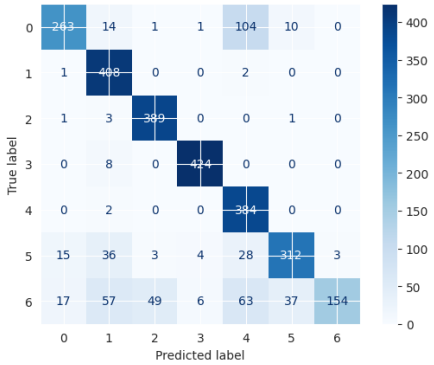

Fig. 13. Confusion matrix for fold 5 of AutoEncoder FFNN.

From figure 13 we can see that, while the usual missclassification of class 6 is still present, the model also often fails to recognize class 0 from class 4 which may indicate that the dataset resulting from the AutoEncoder could be further improved. However, this does not necessarily mean that the compressed dataset cannot be used to train good classification models. It might be the case that by applying different algorithms to this data we achieve competitive results.

4.2.1 SVM on fewer dimensions

In section 4.2 we have found a way to compress the dataset to fewer dimensions, obtaining a compressed dataset able to retain all necessary information to correctly classify steel plates defects. However, the FFNN underperformed compared to other models.

With these premises, we have tried to use the compressed dataset resulting from the AutoEncoder approach for training an SVM classifier: the reason of this decision is that we think that the resulting model could theoretically achieve similar results to the hyperparameter tuned SVM, while decreasing significantly training time given the fewer dimensions. As we did in chapter 3.2, a grid search on the same parameters is performed to find the optimal SVM.

Accuracy	Precision	Recall	F1-score
92.21%	92.16%	92.22%	92.17%

Table 7. Scores for AutoEncoder SVM.

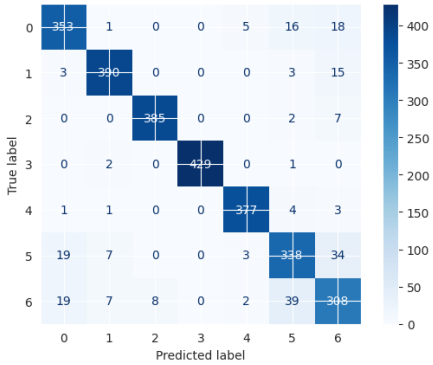

Fig. 14. Confusion matrix for fold 5 of AutoEncoder SVM.

The results we ended up with are very promising: the model performed only 0.3% worse of the hypertuned SVM while decreasing considerably the tuning time. On top of this, we can see that the model does not missclassify classes 0 and 4 (which was the case in chapter 4.2) hinting towards the fact that the AutoEncoder dimensionality reduction dataset can effectively be used to train well-performing classification models.

4.3 TabNet

The TabNet DL architecture has been developed specifically for tabular data with the aim of providing a valid alternative to tree-based models. Analyzing the paper of Arik and T. Pfister[3] we have highlighted two main relevant aspects of the model:

- It is an **instance wise feature selection**, meaning that for each observation the best features are selected in order to produce its class prediction. Moreover, the feature selection is dynamic so different features may be selected at different stages.
- It provides a good explicability of the results.

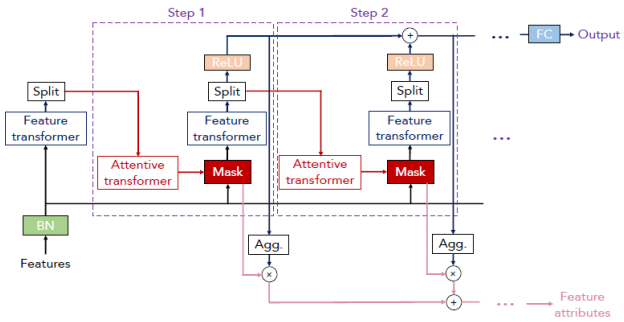

Fig. 15. TabNet architecture, S. Arik and T. Pfister (2019).

The TabNet architecture is based on these sequential steps:

1. Apply **batch normalization** to features.
2. Pass data to a **feature transformer**: in the paper it is made of 4 consecutive blocks having a fully connected layer + batch normalizer + gated linear unit each. Some of these blocks are shared across all decision steps meaning that the weights are shared as well.
3. With the **attentive transformer** the model learns which features are relevant or not for the prediction and than produce a mask. In the attentive transformer a **prior scale term** $P[i]$is applied in order to take into account how much a particular feature has been used in previous steps. Prior scale term incorporates γ **relaxation parameter** and, if $\gamma \approx 1$, then the model tends to select different features for every step, while if $\gamma > 1$ the model reuses the same features for the following steps.
4. The **mask** of salient features is created.
5. 1 Apply **ReLU** and produce the prediction for step 1.
5. 2 Go back to point 2 and repeat everything for second iteration.

The most important features for each observation in generating class predictions can be visualized using a heatmap. This is crucial as it enhances the model's explainability. Figure 16 illustrates this, where brighter colors represent higher feature importance in predicting the faults in our steel plates. The x-axis corresponds to the features used in the prediction process, while the y-axis represents the each steel plates observation.

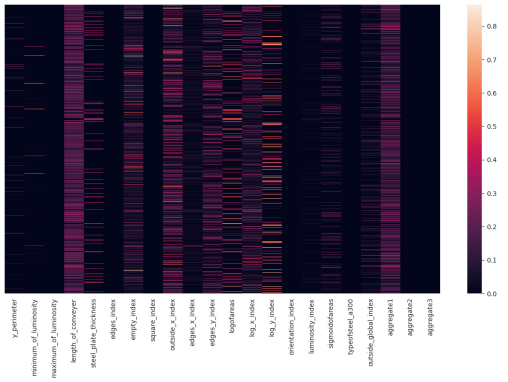

Fig. 16. TabNet learnable mask

We have trained the TabNet model on the full dataset with synthetic data created through SmoteNC and CTGan, and the results are shown in table 8. The model is trained in relatively low time, while still failing to perfectly classify classes 0, 5 and 6.

Accuracy	Precision	Recall	F1-score
91.61%	91.58%	91.61%	91.55%

Table 8. Scores for TabNet.

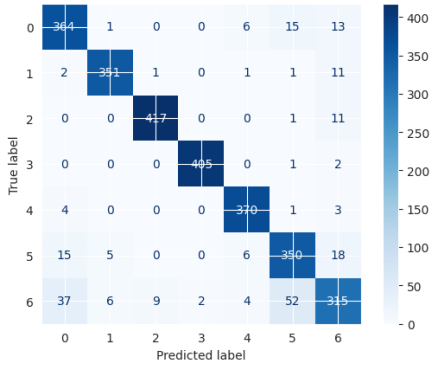

Fig. 17. Confusion matrix for fold 5 of TabNet.

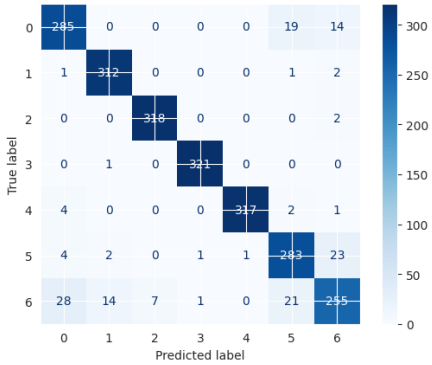

Fig. 19. Confusion matrix for fold 5 of FTTransformer.

4.4 FTTransformer

FT-Transformer[2] is an adaptation of the Transformer architecture for the tabular domain. This adaptation consists mainly of two elements:

- **Feature tokenizer module**: the implementation of such module allows to convert all the features of each observation, both categorical and numerical, into embeddings.
- **Self-attention mechanism**: differently from NLP, in which the self-attention mechanism focuses on the **sequence** of tokens, in FTTransformer model the data is not a sequence of words but a set of features from a tabular dataset. In FTTransformer each feature is treated as a token and so **self-attention is applied across features**. This allows the transformer to learn the interactions between the different features and how they influence each other in order to improve the final prediction.

Basically, the FFTransformer first transforms all the features into embeddings via the feature tokenizer module, then it applies a stock of transformer layers learning the relations among the features, and finally the output of the transformer layers is used for class prediction by using softmax function. To have a visual explanation of the FTTransformer architecture, figure 18 can be helpful.

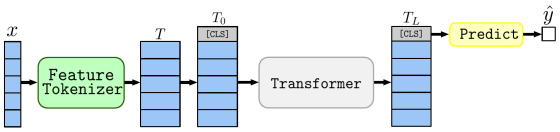

Fig. 18. FTTransformer architecture.

This is a very direct adaptation of the transformer model, which in recent years has been incredibly successful in sequence-based tasks and has now been adapted for tabular data, leveraging self-attention mechanisms to capture feature interactions more effectively than traditional architectures.

Accuracy	Precision	Recall	F1-score
93.16%	93.08%	93.16%	93.07%

Table 9. Scores for FTTransformer.

As we can see from table 9 we achieved very good results when it comes to the accuracy, precision, recall and f1-score of the model prediction, directly comparable to some of the best performing algorithms deployed in this project.

5. RESULTS

After implementing both traditional ML and more recent DL architectures, we can assess their performance based on accuracy and average training time. First of all it is important to notice that, except for the Naive Bayes and the FFNN trained on the dataset compressed through an AutoEncoder, all the models achieved similar performances as we can see from figure 20 and table 10. However, some considerations need to be done.

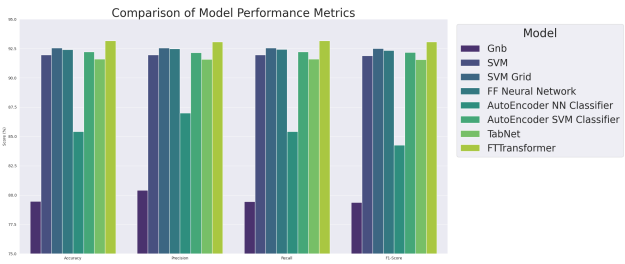

Fig. 20. Metrics for the predictions of different models.

Model	Accuracy	Precision	Recall	F1-score
Gnb	79.67%	80.59%	79.67%	79.56%
SVM	91.96%	91.96%	91.95%	91.89%
Tuned SVM	92.54%	92.54%	92.54%	92.5%
Tuned FFNN	92.41%	92.47%	92.43%	92.33%
AE FFNN	85.42%	87%	85.41%	84.26%
AE SVM	92.21%	92.16%	92.22%	92.17%
TabNet	91.61%	91.58%	91.61%	91.55%
FTTransformer	93.16%	93.08%	93.16%	93.07%

Table 10. Final sores.

Among all the models, the Support Vector Machine (SVM) without hyperparameter tuning offers the best balance between computational efficiency and classification performance, achieving an accuracy of 91.96% while requiring

only one second for training. However, when considering pure performance the FT-Transformer emerges as the best model, achieving an accuracy of 93.16%; on top of this, we can say that the transformer model also achieved very promising results when it comes to average training time per fold.

On the other hand, some models like the simple tuned FFNN and SVM both achieved very good and comparable results but at the cost of a very high training time (direct cause of the hyperparameter optimization). Still, they were not able to beat the performance of the FTTransformer, an architecture which took a fraction of their time to train.

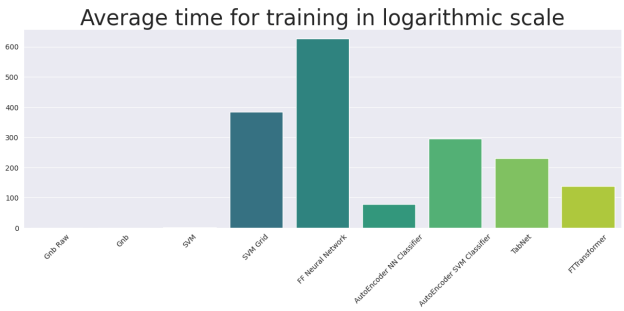

Fig. 21. Average time per fold of different models

Finally, it is important to highlight the significance of TabNet. Although it is neither the most accurate model among both DL and traditional approaches, it introduces a crucial advantage: global interpretability. This property allows the model to quantify feature importance across all observations, providing deeper insights into the decision-making process.

6. CONCLUSIONS

In this project, we explored various ML and DL models to classify defects in steel plates. We started with an exploratory data analysis, where we identified a strong class imbalance, which was addressed using SMOTENC and CTGAN to generate synthetic data. Then, we implemented traditional ML models such as Naive Bayes, Support Vector Machine (SVM), and SVM with tuning, to establish a benchmark. Then, we focused on DL models including a Feed Forward Neural Network (FFNN), an AutoEncoder NN classifier and an AutoEncoder SVM classifier. Finally, we have two advanced architectures specifically designed for tabular data: TabNet and FT-Transformer. The goal was to evaluate the effectiveness of Deep Learning for tabular data, an area where these models are traditionally not considered.

Our comparative analysis has highlighted two key aspects. First, in terms of accuracy, DL models can match or even outperform well-established machine learning models traditionally used for tabular data. However, the main limitations of DL models lie in their higher implementation complexity and longer training times. Considering these factors, we conclude that there is no universally best model. The choice of the model should depend on specific requirements and circumstances. For instance, if training time is not a constraint and even small accuracy improvements are important (as in industrial applications) the FT-Transformer would be the preferred choice. Conversely, if real-time classification is required, the SVM would be a more suitable option due to its efficiency.

For future research, a more in-depth comparison between Deep Learning models specifically designed for tabular data and tree-based models, which currently represent the state-of-the-art for this domain, should be conducted. Tree-based models were not included in this study, as they were not covered in class, and the focus was instead placed on Deep Learning methods. Additionally, a key challenge as stated by Gorishny et. al [2], remains the lack of a widely recognized benchmark dataset for tabular data similar to ImageNet in computer vision, which would enable standardized evaluation and comparison of Deep Learning advancements in this field.

References

1. Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
2. Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18948, 2021.
3. Tomas Pfister Sercan O . Arık. Tabnet: Attentive interpretable tabular learning. *The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*, 2019.
4. L. Xu et al. Modeling tabular data using conditional gan. *33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada*, 2019.