Report for the Machine Learning course

Carolina Polese
Robotic Engineering - University of Genoa
s4862903@studenti.unige.it

Naive Bayes classifier

***Abstract*—The Naive Bayes Classification is a family of algorithms for probabilistic classification based on Bayes' theorem. Relied on the probabilistic problem, naive Bayesian classifiers can be effectively trained in a supervised learning context to classify a set of observations. A well-known approach to smooth the Naive Bayes Classifier is the Laplace Smoothing, that consist in adds one observation to each input class. In this toy-study was considered a weather dataset to decide whether to go play outdoor.**

***Index Terms*—Naive Bayes Classifier, Laplace smoothing, Weather dataset**

I. Introduction

The Naive Bayes Classifier model is widely employed [15] in machine learning application from medical diagnosis [2] to students' education [11]. Based on the specific characteristics of each probabilistic model, naive Bayesian classifiers can be trained in a supervised learning context to classify a set of observations [4]. In this toy-study we used it to predict a easy probabilistic problem. The aim of the model was to predict if would be a good day to play tennis outside given four classes of data, i.e., *outlook*, *temperature*, *humidity*, and *windy*. To avoid null probabilities and enhanced the accuracy, during the model's training were been introduced a Laplace Smoothing, like other studies [12], [13].

II. Material and methods

A. Data processing

Before working with the data it was needed to be processed. The first thing was to shuffle raw's dataset, preventing paths in itself avoiding biases in the trained model. At this point, the data was splitted in four parts:

1) *training input data*, the 75 % of the input data (i.e., *Outlook*, *Temperature*, *Humidity*, and *Windy*);
2) *test input data*, the remaining 25 % of the input data;
3) *training output data*, the 75 % of the data in the column *Play*;
4) *test input data*, the remaining 25 % of the output data.

The input and output training data are used to fit the Naive Bayes Classifier. After testing the trained model (with test data), were evaluated with *error rate*.

B. Naive Bayes Classifier

To simplify the problem we assumed that each feature of each class is independent from the others. This method called *Idiot's Bayes*, although being almost always wrong is extremely convenient [14]. In order to train the Naive Bayes Classifier were computed the *priors probability* and *likelihood probability*.

1) Priors probability: The first part of train the model is to compute the *priors probability*

$$P(C_j) = \frac{N_{C_j}}{\sum_{k=1}^{m} N_{C_k}} \tag{1}$$

where N_{C_j} is the total number of instances that belong to class C_j.

2) Conditional probability: The *conditional probability* was unused to fit the model as

$$P(x_i \mid C_j) = \frac{N_{x_i,C_j}}{N_{C_j}} \tag{2}$$

where N_{x_i,C_j} is the number of times the feature x_i appear in the instance of class C_j. In order to avoid $P(x_i \mid C_j) = 0$ was implemented the *Laplace smoothing* and the Equation 2 turns into

$$P(x_i \mid C_j) = \frac{N_{x_i,C_j} + \alpha}{N_{C_j} + \alpha \nu_i} \tag{3}$$

3) Likelihood probability: The prediction in this model is given by the *likelihood probability*

$$P(C_j \mid X) = \frac{P(C_j) \prod_{i=1}^{n} P(x_i \mid C_j)}{P(X)} \tag{4}$$

where α is the *Laplace smoothing parameter* and ν_i is the number of possibles distinct values that the feature x_i can assume.
Although the probability $P(X)$ is often unknown, it is possible to choose which class C_i has more probability comparing numerator of the fraction in the Equation 4.

C. Model evaluation

The model accuracy was estimated using the error rate r_e defined as:

$$r_e = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{5}$$

which represents the average of the correct predictions.

III. Results and Conclusion

The model was trained using a toy-study dataset containing 14th rows. Consequently, the dataset was specifically designed for a toy-study, and the data presented were highly limited. Although the obtained results were coherent, they were insufficient to evaluate the model's performance, particularly in terms of the *error rate*.

Linear Regression

***Abstract*—Linear regression is a statistical method used to analyze and predict the behavior of one variable based on one or more explanatory variables, by constructing a linear mathematical relationship that fits the collected data. This technique is widely used in various fields such as economics, biology, and engineering, due to its ability to provide clear and easy-to-interpret models. Despite its assumption of linearity between variables and its sensitivity to outliers, advanced approaches such as regularization and multivariate techniques enhance its stability and reliability. This paper explores the fundamental principles of linear regression, its various applications, and strategies to optimize its performance.**

***Index Terms*—Linear regression, multivariable, mean squared error**

IV. Introduction

Linear regression remains a fundamental tool in statistical learning for predicting quantitative outcomes. It aims to estimate a continuous target variable by determining the best-fitting linear relationship through the data points, minimizing the difference (or error) between predicted and observed values. This process is achieved by adjusting the slope and intercept of the line, typically minimizing error using a cost function such as the Mean Squared Error (MSE) [10]. In addition to its predictive capabilities, linear regression is widely used as a diagnostic tool for examining relationships between variables, particularly in domains like economics, biology, and social sciences. It also serves as a baseline for comparing more complex models in machine learning, despite its limitations [6].

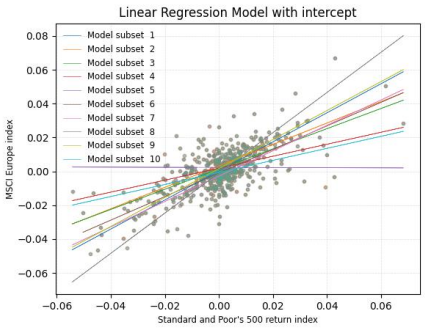


Fig. 1: Graphical representation of the model trained on ten subsets with interception, each corresponding to 5% of the Turkish dataset. In the background are visualized the corrispective test subsets (95% of the dataset).

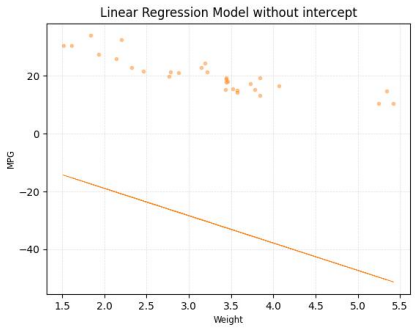


Fig. 2: Graphical representation of the model trained without interception, which corresponds to 5% of the MTK dataset. In the background is visible the test subset (95% of the dataset).

V. Material and methods

A. Data processing

Two distinct datasets were analyzed:

- *turkish-se-SP500vsMSCI*: the first column contains historical data on the returns of the S&P 500 index (USA), while the second contains the MSCI Europe index, representing the U.S. and European stock markets.
- *mtcarsdata-4features*: this dataset consists of four features from the mtcars dataset, which records data on various car models. The primary variables include *mpg* (Miles Per Gallon, as an indicator of fuel efficiency), *weight* (vehicle weight), *hp* (horsepower), and *disp* (engine displacement). This dataset is used to analyze the relationship between a car's mechanical characteristics and its fuel consumption.

Each dataset was randomly shuffled and then divided into training and testing subsets. For the Turkish dataset, ten subsets were used for cross-validation, while for the MTK dataset, a split of 5% for training and 95% for testing was employed. To highlight differences, data from the start and end of the Turkish dataset was selected rather than random samples, as data points in close proximity are often more similar.

B. Linear Regression Model

The linear regression model was evaluated using two distinct methods. For the Turkish dataset, the model's performance was assessed using ten random subsets, each corresponding to 10% of the dataset (Figure 1). For the MTK dataset, the model was applied using both univariate and multivariate regression, as represented in Figures 2, 3, and 4. The effect of including an intercept term was also examined.

1) Turkish dataset: : this dataset was shuffled and split into two subsets: one for training (5% of the data) and one for testing (95% of the data).

2) MTK dataset: : the analysis was conducted in two ways: first, considering only the first two columns (*weight* and *mpg*), and second, using all four columns (*disp*, *hp*, *weight*, and *mpg*) for multivariable linear regression. This allowed for evaluating how additional features affected the model.

Linear regression performance was also assessed in two scenarios: with and without the inclusion of an intercept term.

Linear regression relies on the mathematical model:

$$y = \beta_0 + \beta_1 x \tag{6}$$

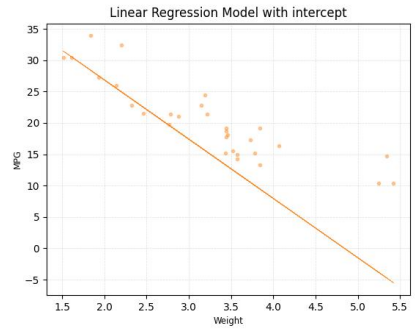


Fig. 3: Graphical representation of the model trained with interception, which corresponds to 5% of the MTK dataset. In the background is visible the test subset (95% of the dataset).

where y is the dependent variable, x is the independent variable, β_0 is the intercept, β_1 is the slope of the line [7]. However, unless the independent and dependent variables can be described by exactly two points, the model must account for error using the error term ε.

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{7}$$

The intercept β_0 represents the value of y when all predictors are zero. Including the intercept improves the model's ability to fit the data, especially when assuming $y = 0$ without independent variables is unrealistic. In linear regression, ε represents the error or residual term, capturing the variability in y not explained by the independent

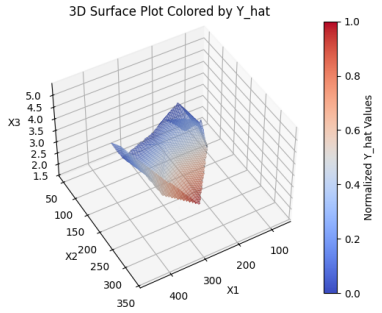


Fig. 4: Graphical representation in fourth dimensional representation, where the fourth one is the predicted output. The model, trained with interception, corresponds to 5% of the MTK dataset.

variables. The error term is calculated as:

$$\varepsilon = y_{\text{observed}} - y_{\text{predicted}} \tag{8}$$

This term is critical as it highlights the model's limitations, suggesting that even with careful modeling, unpredictable factors may influence the outcome.

Multivariable linear regression extends simple linear regression by modeling the relationship between a dependent variable and two or more independent variables:

$$y = X\mathbf{w} \tag{9}$$

where X is the matrix of independent variables and $\mathbf{w}$ is the slope's vector. If the intercept is included, a column of ones is added to the X matrix.
Additionally, when matrix inversion issues arise during gradient calculation, the pseudo-inverse is used to provide an approximate solution.

C. Model evaluation

Mean Squared Error (MSE) is a key metric for assessing the model's accuracy. It is calculated as the average of squared differences between observed and predicted values across all data points:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\varepsilon_i^2 \tag{10}$$

MSE serves as a loss function to minimize model error during training, providing an overall measure of the prediction error across the dataset.

VI. Results

Figures 2 and 3 demonstrate that training the model with only two variables (i.e. *weight* and *mpg*) from the MTK dataset results in noticeable differences depending on whether an intercept is included. When no intercept is used, the regression line does not intersect any data points. In contrast, including the intercept results in a line that better fits the data. Figure 4 shows that the fourth-dimensional regression model presents challenges in interpretation. This is likely due to the limited dataset used in the analysis.

For the Turkish dataset, the median Mean Squared Error (MSE) from the ten subsets was calculated as $2.44 \cdot 10^{-4}$, with the first quartile at $2.20 \cdot 10^{-4}$ and the third quartile at $2.65 \cdot 10^{-4}$, for both models with and without intercept.

VII. Conclusion

This analysis demonstrates the effectiveness of linear regression in modeling relationships between variables in two datasets: Turkish and MTK. The results for the Turkish dataset show that the model explains a significant portion of the variance, with an MSE of $2.44 \cdot 10^{-4}$. . For the MTK dataset, the model performed better when all four columns were used, with MSE values of 68.40 and 225.28, indicating that the additional features enhanced the model's predictive power. Despite the relatively small MSE values, the results also suggest that the models may not capture all underlying data patterns, emphasizing the limitations of linear regression.

kNN Classifier

***Abstract*—The k-Nearest Neighbors (kNN) classifier is a supervised machine learning algorithm known for its simplicity and versatility, particularly effective in classification tasks. For instance, it is widely used in the medical field which highlights the importance of ontological approaches to optimize feature selection and enhance the robustness of kNN in complex contexts.**

This report presents the implementation of kNN, emphasizing data preprocessing, relevant feature selection, and the optimization of the k parameter. The algorithm was applied to a reference dataset to evaluate its performance in terms of accuracy and precision. The results demonstrate how the choice of k significantly influences the trade-off between predictive accuracy and computational complexity.

***Index Terms*—kNN Classifier, accuracy**

VIII. Introduction

In the rapidly evolving field of machine learning, the k-Nearest Neighbors (kNN) algorithm has emerged as a fundamental yet powerful tool for classification and regression tasks. Its intuitive approach—relying on proximity within the feature space to make predictions—makes kNN particularly attractive for applications where interpretability and simplicity are key. Despite its straightforward nature, the algorithm's performance is deeply influenced by factors such as data preprocessing, feature selection, and the choice of the k parameter, which determines the number of neighbors considered during classification.

One of the primary strengths of kNN lies in its versatility. It has been successfully applied in different fields, as image recognition, fraud detection, and medical diagnostics. A study *"Fuzzy Decision Ontology for Melanoma Diagnosis Using kNN Classifier"* [1] underscore the algorithm's role in improving diagnostic accuracy through tailored feature selection strategies. However, kNN is not without limitations. It is sensitive to noise, computationally intensive for large datasets, and may struggle with imbalanced data.

There were implemented and evaluated the kNN algorithm focusing on its practical aspects, performance optimization, the impact of preprocessing techniques, the role of feature selection in reducing dimensionality, and the trade-offs involved in determining the optimal number of neighbors k.

By providing a comprehensive analysis of kNN's strengths, limitations, and potential improvements, this report aims to offer valuable insights for leveraging the algorithm effectively in real-world applications.

IX. Material and methods

A. Data processing

This study is based on the analysis of a dataset describing various types of wines through their chemical characteristics and corresponding classifications. Specifically, the dataset is divided in two subsets: $wine.data$, which contains the chemical measurements associated with each wine sample (i.e., *Alcohol*, *Malic acid*, *Ash*, *Alcalinity of ash*, *Magnesium*, *Total phenols*, *Flavanoids*, *Nonflavanoid phenols*, *Proanthocyanins*, *Color intensity*, *Hue*, *OD280/OD315 of diluted wines*, *Proline*), and $wine.target$, which identifies the specific class to which each wine in the analysis belongs. Subsequently, both subsets were split and randomized to ensure accurate analysis. Specifically, the following subsets were generated:

- x_{train}, containing 80% of the data;
- x_{test}, containing 20% of the data;
- y_{train}, containing 80% of the targets;
- y_{test}, containing 20% of the targets.

In addition, prior to the analysis, it was necessary to normalize the obtained subsets. This step is crucial because the analysis involves calculating distances between different values. Given that the dataset contains variables with differing ranges of values, it is essential to scale the data to a uniform range to ensure that all features contribute equally to the computation of distances. These subsets were then used for training and testing the kNN Classifier implemented in our study.

B. kNN Classifier

The implementation of k-Nearest Neighbors (kNN) algorithm is based on computing of normalized linear distance between the points of the training set and the evaluated test's point. After computing the distance, the algorithm predicted the class to which points belong based on the number of neighbors k.

C. Model evaluation

The model was evaluated computing its accuracy in relation to the specific k parameter considered during the analysis and model training. As mentioned earlier, the k parameter represents the number of nearest neighbors taken into account for classifying a new data point, determining how many points in the training dataset are chosen to decide the class of a sample. The results obtained, specifically the model's accuracy for each value of k, are displayed in Figure 5. Another parameters used to evaluate the model are: the *error rate*, which represent the percentage of incorrect predictions; the *precision* that characterize the proportion of true positives (correctly classified instances) among all positive predictions made by the classifier; the *recall* which is the proportion of true positives among all actual positive instances; the *F1 Score* that represent the harmonic mean of precision and recall, providing a balanced measure of both. The last three parameters are obtained with a *confusion matrix*. A *confusion matrix* is a table used to evaluate the performance of a classification model by comparing its predictions against the actual outcomes. It provides a detailed breakdown of prediction accuracy across different classes, offering insight into the model's strengths and areas for improvement. The confusion matrix is structured as follows:

$$C = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

where the *true positives* (TP) are the cases where the model correctly predicted the positive class; the *true negatives* (TN) are the cases where the model correctly predicted the negative class; the *false positives* (FP) are the cases where the model incorrectly predicted the positive class (Type I error); and the *false negatives* (FN) is the cases where the model incorrectly predicted the negative class (Type II error).

TABLE I: Class 3 Statistics over all k values

Metric	Mean	Median	AAD	I Quantile	III Quantile
Precision	0.949495	1	0.0550964	0.888889	1
Recall	0.983766	1	0.0177096	0.964286	1
F1 Score	0.966295	0.962925	0.00612813	0.962925	0.962925

TABLE II: Total Statistics over all k values for of all Classes

Metric	Mean	Median	AAD	I Quantile	III Quantile
Precision	0.967651	0.96558	0.0364505	0.957537	0.976729
Recall	0.972353	0.983766	0.029891	0.962662	0.987751
F1 Score	0.969831	0.966295	0.0106918	0.959848	0.978046

X. Results and Conclusion

The parameter k determines how many neighbors are considered to make the classification. The value of k has a significant impact on the model's performance. If k is too small, the algorithm can be sensitive to noise in the data and may lead to overfitting, where the model fits too closely to the training data. On the other hand,

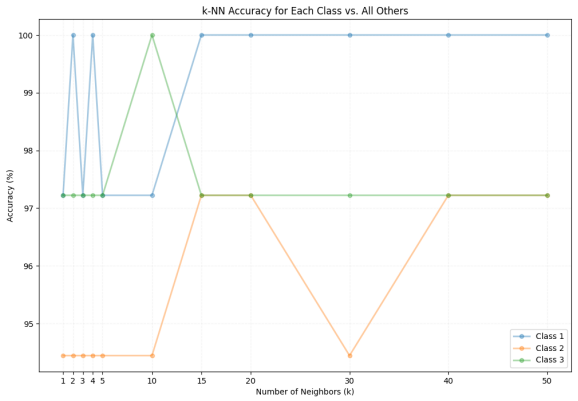


Fig. 5: Relationship between the number of neighbors k and the classification accuracy across all classes for the kNN classifier.

if k is too large, it can reduce the model's ability to capture local variations in the data, resulting in under-fitting, where the model fails to adequately represent the complexities of the dataset. As shown in Figure 5, choosing the optimal value of k is crucial to balance accuracy and computational complexity. The *error rate* for each class and each k-Nearest Neighbors was computed, and the mean of these rates was calculated. The following results were obtained:

- *Class 1*: 1.01%
- *Class 2*: 4.45%
- *Class 3*: 2.53%

The last important parameters used to evaluate the k-Nearest Neighbors (kNN) algorithm are presented in the Tables I, and II. From the results shown in the Figure 5 and Table II, it is evident that the model's predicted precision is strong. In general, the results from the tables demonstrate good performance of the kNN model in terms of precision, recall, and F1 Score. The means are high for all metrics, and the first and third quantiles confirm that the extreme values (both low and high) are fairly stable. Overall, the evaluation of the model is positive, with a good balance between the different metrics for each class and for the entire dataset.

Neural Network

***Abstract*—rtificial neural networks are a fundamental approach in machine learning due to their efficiency in modeling complex relationships and learning meaningful representations from data. Autoencoders, a specific class of neural networks, are widely used for dimensionality reduction, data compression, and unsupervised learning tasks. This report explores the core principles of neural networks and autoencoders, analyzing their architecture, training process, and practical applications.**

***Index Terms*—Artificial neural networks, auto-encoder**

XI. Introduction

Artificial neural networks represent an innovative approach in machine learning, capable of tackling complex problems across various domains, from image processing to sequential data prediction. Autoencoders are among the earliest neural network architectures designed to learn latent representations of data in an unsupervised manner. They function by compressing data into a reduced-dimensional space and then reconstructing the original input, aiming to capture relevant features and redundancies. The significance of autoencoders lies in their versatility: they have been successfully applied to tasks such as dimensionality reduction [5], anomaly detection [3], and

deep network pretraining. Furthermore, recent developments, such as variational autoencoders [9], have expanded their applications to include synthetic data generation and probabilistic learning. This report aims to provide an analysis of the fundamental concepts related to neural networks and autoencoders, illustrating practical examples and experiments that demonstrate their effectiveness in real-world applications.

XII. Material and methods

A. Data processing

For this analysis, the MNIST dataset (Modified National Institute of Standards and Technology) was used, one of the most well-known datasets for image classification tasks. This dataset contains images of handwritten digits (ranging from (0 to 9), where each image is grayscale, has dimensions of 28×28 pixels, and each pixel has a value between 0 and 255, representing the intensity of the color (0 means black, 255 means white).

After loading the data, only the digits 1 and 8 were selected. This reduced the dataset to two classes of interest. The filtered data were then split into two distinct sets:

- (x_{train}, y_{train}): 80% of the filtered data containing the training images. The images are represented as 28×28 matrices describing the pixel values, and the corresponding training labels (1 and 8), represent the digits associated with the images.
- (x_{test}, y_{test}): 20% of the filtered data containing the test images. These are also represented as 28×28 matrices describing the pixel values, with the corresponding labels for the test images (1 and 8), which represent the digits associated with the images.

The following steps were performed to make the data compatible with the model architecture:

- *Data Normalization*: to enhance performance and ensure compatibility with the model architecture, the data were normalized by scaling the pixel values from their original range (0-255) to a range between 0 and 1. This transformation is crucial for accelerating convergence during training and minimizing the risk of numerical instability.
- *Data Reshaping*: since the MNIST images, when loaded, have the shape of a three-dimensional array ($numsamples$, 28, 28), where: $numsamples$ refers to the number of images in the dataset and 28×28 represents the dimensions of each image, many deep learning models require the data to be represented in a four-dimensional format: ($numsamples$, *height*, *width*, *channels*). In the case of MNIST images, the channels value is 1 because the images are grayscale. Thus, the data were reshaped from the form ($numsamples$, 28, 28) to ($numsamples$, 28, 28, 1) using the *reshape* command. This step explicitly adds the necessary depth channel, making the data compatible with the autoencoder architecture.

B. Auto-encoder

An autoencoder is a neural network model designed to learn a compact and meaningful representation of input data. The model consists of two main parts: the encoder, which receives the input data and compresses it into a lower-dimensional representation, and the decoder, which receives the compressed representation and reconstructs it into the original output data.

The autoencoder model used in this work consists of the following components:

- *Encoder*: the neural network that receives the input data and compresses it into a lower-dimensional representation. The encoder consists of a series of neural network layers, each applying a nonlinear transformation to the input data.

Fig. 6: Reconstruction of the digits data 1 and 8. In the first row there are the original data digit and in the second row the reconstructed data

- *Decoder*: the neural network that receives the compressed representation and reconstructs it into the original output data. The decoder consists of a series of neural network layers, each applying a nonlinear transformation to the data.
- *Autoencoder*: the complete model that combines both the encoder and the decoder. The autoencoder is trained to minimize the reconstruction error, which is the difference between the original input data and the reconstructed output data.

The autoencoder operates as follows: the input data is fed to the encoder, which compresses it into a lower-dimensional representation. The decoder then receives this compressed representation and reconstructs it into the original output data. The reconstruction error is calculated as the difference between the original input data and the reconstructed output data. The model is then trained to minimize the reconstruction error using the Adam optimizer algorithm, which stands for "Adaptive Moment Estimation," to minimize the loss function during the training of neural networks.

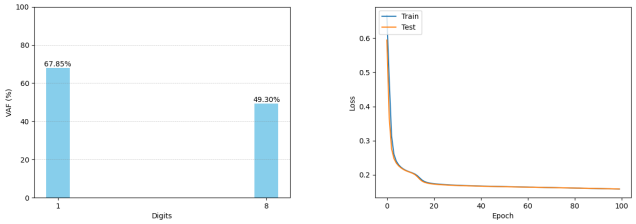
C. Model evaluation

There are various methods to evaluate the correct implementation of the autoencoder model. The most intuitive approach is visual inspection, which involves checking whether the digital reconstruction accurately reflects the original input data (see Fig. 6)

An alternative approach involves using the Variance Accounted For (VAF) metric, which is specific for evaluating the performance of autoencoders. The VAF measures the quality of data reconstruction by evaluating how well the compressed data represents the original input. The VAF is calculated as:

$$VAF = 1 - \left(\frac{var(\mathrm{x}_{test} - \mathrm{x}_{rec})}{var(\mathrm{x}_{test})} \right) \cdot 100$$

where: x_{test} represents the original values; x_{rec} represents the values reconstructed by the autoencoder. The calculated values for each reconstructed data point are reported in Fig. 7.



(a) The figure represented the VAF computed for 1 and 8

(b) The figure represented the information lost during the process

Fig. 7

Another useful method for evaluating the model is by calculating the information loss during reconstruction, as shown in Fig. 7.

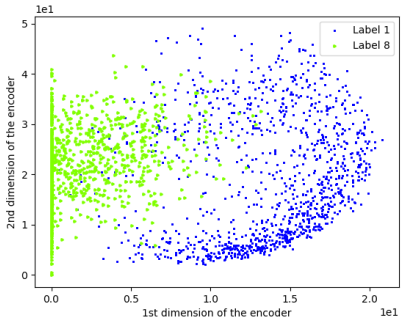


Fig. 8: Visualization of the encoded latent space, showing the dimensions of the encoded representation. Points are color-coded based on digital values: Label 1 (blue) and Label 8 (green).

XIII. Results and Conclusion

The proper functioning of the autoencoder is demonstrated in Fig. 6, which highlights the model's ability to reconstruct the original data. This result showcases the architecture's effectiveness in capturing the essential features of the input data while minimizing reconstruction error.

Additionally, the progressive reduction in information loss as the number of epochs increases indicates the steady improvement in model performance (see Fig. 7b). This behavior reflects the optimization of the autoencoder's weights and biases over time, allowing for better alignment between the encoded latent representation and the original data structure.

The analysis of the Variance Accounted For (VAF), shown in Fig. 7a, provides further insights into the autoencoder's performance for different digits. For the digital value 1, the VAF reaches 67.85%, indicating the model's good ability to capture and reconstruct this value with minimal information loss. In contrast, for the digital value 8, the VAF is 49.30% , which suggests a greater challenge in preserving information during reconstruction. These results emphasize the superior performance of the autoencoder with certain digits compared to others, likely due to the complexity or distribution of the data associated with each digit. Furthermore, the scatter plot in Fig. 8 shows the latent space encoded by the autoencoder, visualizing the first two dimensions of the encoded representation. The points are color-coded based on the digit labels 1 (blue) and 8 (green). This visualization highlights the distinct separation of the two classes in the encoded space, demonstrating the autoencoder's ability to learn meaningful representations.

The cluster corresponding to 1 appears well-defined and compact, suggesting that the autoencoder effectively captures the structure of this digit. In contrast, the points associated with 8 exhibit greater dispersion, reflecting more variability in the encoded representation. This aligns with the previously observed lower VAF for 8, indicating that the reconstruction process struggles more with this digit.

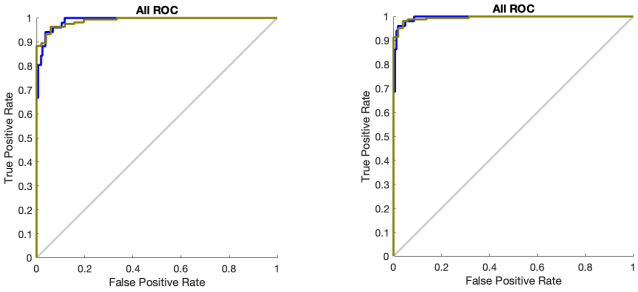
These differences in the encoded space emphasize the variability in the autoencoder's performance based on the complexity and distribution of the input data. Future improvements could focus on enhancing the encoder's ability to capture finer details for more challenging digits, potentially through increased latent dimensionality or advanced regularization techniques.

XIV. Deep Learning Toolbox

The Deep Learning Toolbox in MATLAB provides advanced tools for designing, training, and analyzing neural networks, making it easier to solve classification problems and other machine learning tasks. Through its integrated functions, it is possible to visualize and evaluate model performance using confusion matrices and ROC curves. Confusion matrices offer an intuitive method to compare the predicted classes of a model with the actual ones, allowing for the analysis of classification errors and correct predictions. Receiver Operating Characteristic (ROC) curves, on the other hand, examine the trade-off between sensitivity and specificity, providing a graphical representation of model performance across various scenarios. Using these tools, the Deep Learning Toolbox simplifies the analysis and improvement of implemented models, enabling the optimization of design choices. In this study, three different neural networks were implemented:

- Neural network 1 is characterized by 2 layers: the first consisting of 10 units, and the second consisting of 2 units.
- Neural network 2 is characterized by 2 layers: the first consisting of 20 units, and the second consisting of 2 units.
- Neural network 3 is characterized by 4 layers: the first consisting of 20 units, the second consisting of 10 units, the third consisting of 5 units, and the fourth consisting of 2 units.

To analyze the performance of each neural network, the graphs generated by the confusion matrices and ROC curves were compared. The confusion matrices, shown in Figs. 10, highlight an improvement in model accuracy proportional to the number of layers used to define the architecture. Specifically, neural network 1 achieves an accuracy of 94.9%, while neural network 3 reaches 96.7%. This improvement suggests that more complex architectures, when well-designed, lead to better predictive performance. The ROC graphs (see Figs. 9) confirm these findings: in neural network 3, the combined curve is closer to the upper-left corner, representing the ideal value, compared to that of neural network 2. Overall, the models defined in this study demonstrate good predictive capabilities.



(a) ROC for Neural Network 2 (b) ROC for Neural Network 3

Fig. 9

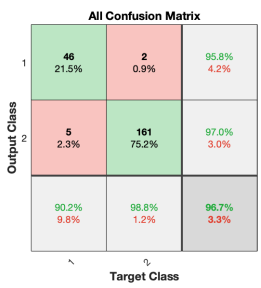
References

[1] W. Abbes, D. Sellami, S. Marc-Zwecker, and C. Zanni-Merk, "Fuzzy decision ontology for melanoma diagnosis using KNN classifier," *Multimedia Tools and Applications*, vol. 80, no. 17, pp. 25 517–25 538, Jul. 2021.

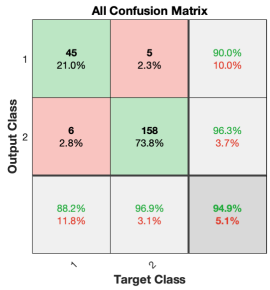
[2] K. Al-Aidaroos, A. A. Bakar, and Z. Othman, "Medical data classification with naive bayes approach," *Information Technology Journal*, vol. 11, no. 9, pp. 1166–1174, 2012.

[3] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *ArXiv*, vol. abs/1901.03407, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:57825713

[4] A. Derbel and Y. Boujelbene, "Automatic classification and analysis of multiple-criteria decision making," in *Proceedings of the 8th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT'18), Vol. 1*. Springer, 2020, pp. 83–93.



(a) Confusion matrices for Neural Network 3



(b) Confusion matrices for Neural Network 1

Fig. 10

[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, ser. Springer Texts in Statistics. Springer, 2013.

[7] ——, "Linear Regression," in *An Introduction to Statistical Learning: With Applications in R*, G. James, D. Witten, T. Hastie, and R. Tibshirani, Eds. New York, NY: Springer US, 2021, pp. 59–128.

[8] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, "Linear regression," in *An introduction to statistical learning: With applications in python*. Springer, 2023, pp. 69–134.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:216078090

[10] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, 5th ed. Wiley, 2012.

[11] I. T. Nafea, "Machine learning in educational technology," *Machine learning-advanced techniques and emerging applications*, pp. 175–183, 2018.

[12] S. Narayan and E. Sathiyamoorthy, "Early Prediction of Heart Diseases using Naive Bayes Classification Algorithm and Laplace Smoothing Technique," *International Journal of Grid and High Performance Computing*, vol. 14, no. 1, pp. 1–14, 2023.

[13] F. F. Sabiq, A. Rahmatulloh, I. Darmawan, R. Rizal, R. Gunawan, and E. Haerani, "Performance Comparison of Multinomial and Bernoulli Naïve Bayes Algorithms with Laplace Smoothing Optimization in Fake News Classification," in *2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD)*, Aug. 2024, pp. 19–24.

[14] M. Schonlau, *Applied Statistical Learning: With Case Studies in Stata*, ser. Statistics and Computing. Cham: Springer International Publishing, 2023.

[15] I. Wickramasinghe and H. Kalutarage, "Naive Bayes: Applications, variations and vulnerabilities: A review of literature with code snippets for implementation," *Soft Computing*, vol. 25, no. 3, pp. 2277–2293, Feb. 2021.