MNIST Classification

Machine Learning Regression Algorithm Selection

Eduardo Solano Jaime

0213663

Especialidad en Ciencia de Datos

Universidad Panamericana Campus Guadalajara

Abstract

This report presents a comprehensive study on the selection of an optimal machine learning classification algorithm for the MNIST dataset. The goal is to develop a robust model that accurately classifies handwritten digits. The process begins with data preparation and processing, followed by feature engineering to extract meaningful attributes from the images. Various machine learning models are then built and rigorously tested on these features. The performance of each model is evaluated using appropriate metrics. Finally, cross-validation techniques are employed to ensure the model's generalizability and to determine the most effective classification method. The findings of this report contribute to the ongoing efforts in the field of image classification and provide valuable insights for future research. The full code is available in my github repository.

Methodology

1. **Goal Setting**: Define the objective of your project. In this case, it is to classify handwritten digits from the MNIST dataset.
2. **Data Preparation**: Load the MNIST dataset and split it into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance.
3. **Data Processing**: This step can help improve the performance of many machine learning algorithms.
4. **Feature Engineering**: In the context of the MNIST dataset, each pixel in the image can be considered a feature.
5. **Model Building**: Train different machine learning models on the processed data.
6. **Model Testing**: Evaluate the performance of each model on the testing set.
7. **Cross-Validation**: Perform k-fold cross-validation and other methods to assess how well your model generalizes to unseen data.
8. **Model Selection**: Based on the cross-validation results, select the model that performs the best on the validation sets.
9. **Final Evaluation**: Finally, evaluate the performance of your selected model on the testing set.

Dataset Exploration

The MNIST database of handwritten digits with 784 features. It is a subset of a larger set available from the National Institute of Standards and Technology. The digits have been size-normalized and centered in a fixed-size image. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

The dataset is extracted from OpenML's mnist_784, from which, after extraction, a `pandas.DataFrame` with the shape of *70000x784* is obtained containing the pixel values ranging from 0 to 255 and an aditional `pandas.Series` with the corresponding label.

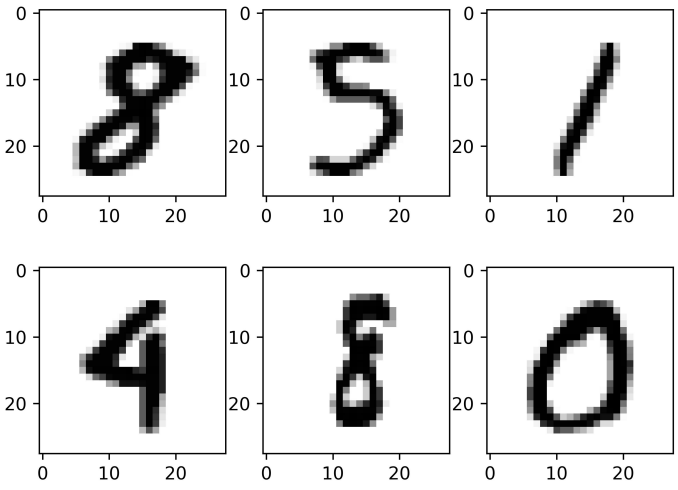
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Columns: 784 entries, pixel1 to pixel784 dtypes: int64(784) memory usage: 418.7 MB
```

Noteworthy to mention is the fact that the dataset is perfectly clean in terms of null and nan values, both in the features and in the labels.

Features characteristics

It comprises 28x28 pixel grayscale images of handwritten digits (0 to 9). Each image is a matrix where each pixel's value represents the grayscale intensity. Therefore, each pixel correspond to a feature in the dataset plus its corresponging label.

- Image Pixels: Each image consists of 28x28, totaling 784 pixels, each containing a value denoting the intensity of the grayscale (ranging from 0 to 255).
- Label: The correct classfification of the sample (0-9).



Random Digits

Upon visualizing the data, it becomes apparent that most of the pixels in the handwritten digits hold minimal relevance to the assigned labels.

While the MNIST data points are *embedded* in 784-dimensional space, they live in a very small subspace. *Christopher Olah*

This pushed the report into a recently introduced (and not discussed in class) feature descomosition. Based on the assumption that the MNIST dataset *occupies* a much smaller dataspace, and appealing to the Manifold hypothesis, the incoming model selection uses PCA following by tSNE as feature enineering process.

Data Processing

For further processing, despite the recomendation of using standarization for PCA (later on), normalization is the numeric transformer of choice. Normalization, unlike standardization, ensures that all features lie within a similar range, allowing for a consistent comparison of their relative importance. In scenarios where the distance between features matters more than their absolute values, normalization proves beneficial, just as in this case.

So now, pixel information ranges from 0 to 1.

Data Transformation

Contrary to the conventional transformations given in class, and after the proper research, it was decided to use the Principal Component Analysis linear dimension reduction algorithm to greatly reduce the dataset's number of features and then a t-Distributed Stochastic Neighbor Embedding to better represent the datapoints in a 3-dimensional space.

The utilization of PCA and t-SNE is driven by their effectiveness in simplifying complex datasets. PCA aids in identifying and consolidating the most essential aspects of the data, simplifying its complexity. It essentially condenses information by highlighting the most critical patterns. On the other hand, t-SNE specializes in revealing relationships between closely positioned data points. It creates a visual representation that emphasizes the similarities between nearby points in the dataset.

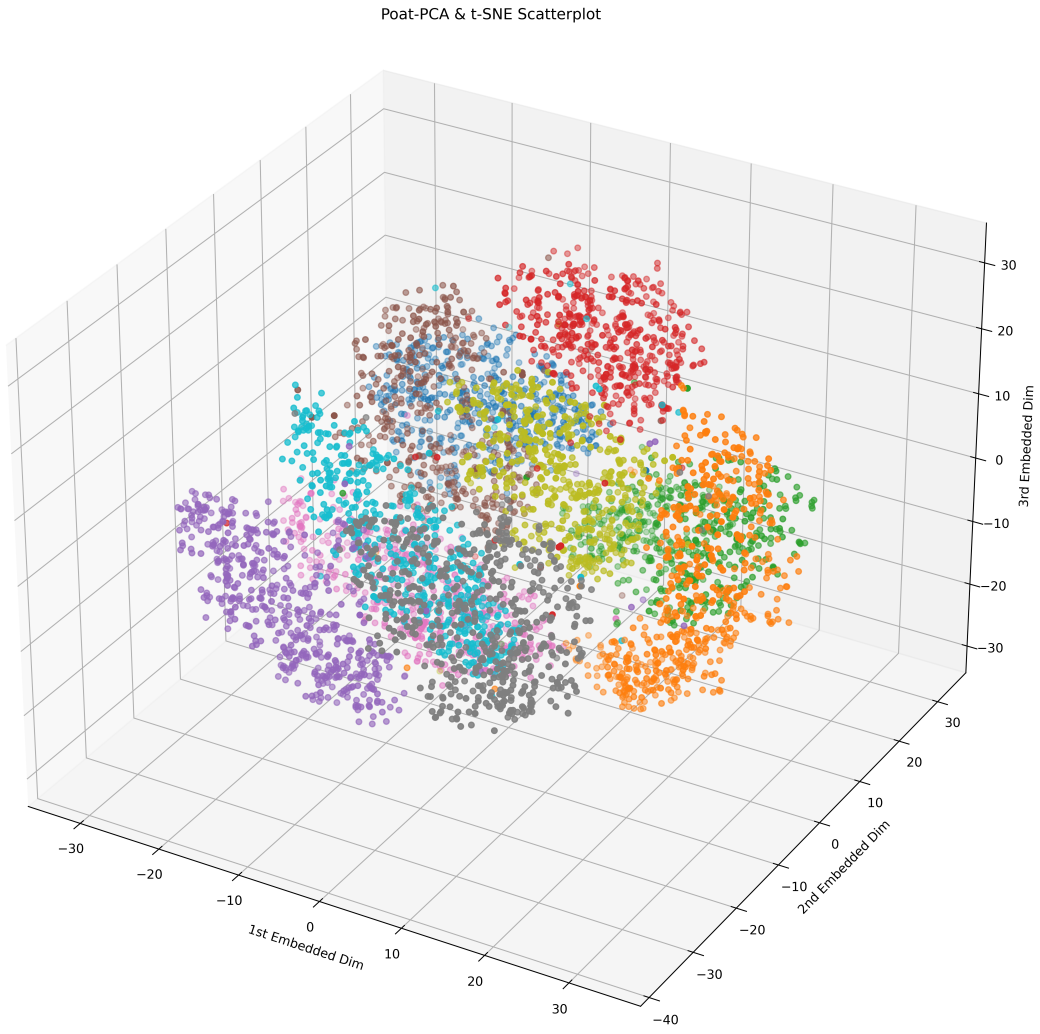
By combining these techniques, a clearer understanding of the dataset is achieved. PCA streamlines and identifies the most important elements, while t-SNE visually represents how these elements relate, particularly focusing on similar items. Together, they facilitate a more comprehensive comprehension of the data, simplifying the complex information and making it more accessible for analysis and interpretation.

```
pca= PCA(n_components=50, random_state=5338)
X_pca= pca.fit_transform(X_fit)
tsne = TSNE(n_components=3, n_jobs=-1, random_state=5338)
X_tsne = tsne.fit_transform(X_pca)
X_tsne['label'] = y
```

After anextensive computatin time and effort, the dataset's dimensions were reduced from 784 to only 3 dimensions, without compromising valuable information.

	dim1	dim2	dim3	label
0	-10.5559	2.88536	11.2567	5
1	-13.6346	26.1194	-3.3482	0
2	-29.1363	-18.7214	-3.54328	4
3	19.417	-7.35925	-19.0398	1
4	-4.46703	-15.393	-1.30892	9

For an easier visualization, the data can be plot according to its new embedded dimenions. A comprenhensive animatyion of the distributed observations was created to display the excellent results provided by the data transformation. Said animation can be seen in this link. Or in a (not-so-visually pleasing, not animated) image representation:



Embedded scatter plot

Using PCA and t-SNE for feature selection and transformation might lead to a downside: the potential loss of the original meaning or interpretability of the features. As these methods focus on reducing dimensions and emphasizing relationships in the data, the direct connection to the original features might be obscured.

Model Building

Some dirty-models are built in order to identify out top champions for classifying the MNIST Dataset. And for that, the data must me split for training. from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X_tsne, y, test_size=1/3,
random_state=5338)
```

One-vs-Rest

One such strategy is the **One-vs-Rest** (OvR) or **One-vs-All** (OvA) approach. In this method, instead of directly classifying into multiple classes, logistic regression is applied iteratively, creating a separate model for each class while treating it as a binary classification problem (one class versus the rest). Then, during prediction, the model that gives the highest probability is chosen as the predicted class.

A score of **0.9833** is achieved against the *test* set with `OneVsRestClassifier(LogisticRegression())`, meaning **98.33%** of the test samples were correctly classified.

Logistic Regression

Another approach is the **multinomial logistic regression**, where the model directly predicts the probability of each class relative to a baseline (often the last class) using a softmax function. The softmax function normalizes the outputs for each class, providing probabilities summing up to 1.

A score of **0.9965** is achieved against the *test* set with `LogisticRegression(multi_class='multinomial', solver='newton-cg')`, meaning **99.65%** of the test samples were correctly classified.

Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training. For multiclass classification, it extends decision trees to handle multiple classes. It combines predictions from multiple trees to determine the final class.

A score of **1.0** is achieved against the *test* set with `RandomForestClassifier()`, meaning **100%** of the test samples were correctly classified.

Support Vector Machines

SVMs can be used for both binary and multiclass classification. In the case of multiclass, it utilizes strategies like One-vs-Rest (OvR) or One-vs-One (OvO) to extend binary SVMs to handle multiple classes.

A score of **0.9883** is achieved against the *test* set with `SVC()`, meaning **98.83%** of the test samples were correctly classified.

K-Nearest Neighbors

KNN is a simple algorithm that classifies data points based on the majority class among their nearest neighbors. It can be extended to handle multiple classes by considering the classes of the k-nearest neighbors.

A score of **0.9887** is achieved against the *test* set with `KNeighborsClassifier()`, meaning **98.87%** of the test samples were correctly classified.

Neural Networks

Neural networks can be adapted for multiclass classification. Output layers with multiple nodes corresponding to each class are used along with appropriate activation functions like softmax or relu. In this case, the use of **Multi-layer Perceptron classifier** is the optimal choice.

A score of **0.9998** is achieved against the *test* set with `MLPClassifier()`, meaning **99.98%** of the test samples were correctly classified.

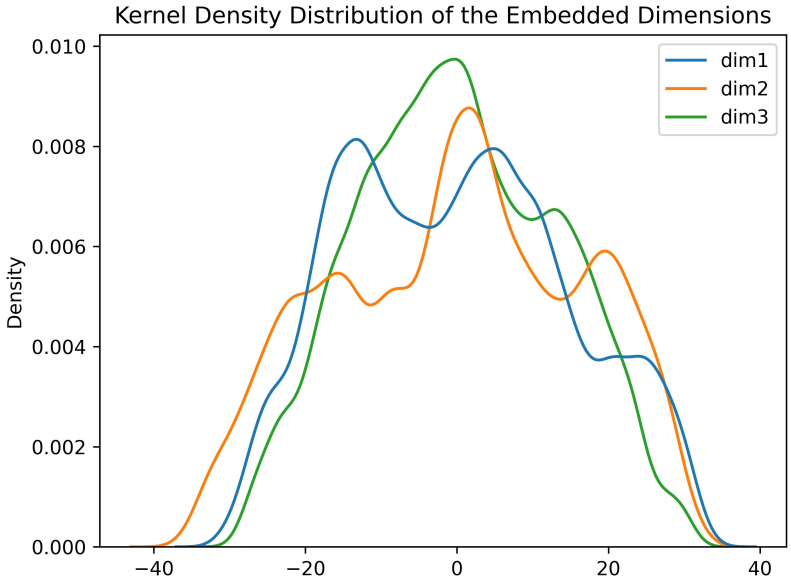
Gradient Boosting Algorithms

Gradient boosting algorithms like **XGBoost**, LightGBM, or CatBoost can also handle multiclass classification problems. They iteratively build a set of weak learners and combine them to make accurate predictions for multiple classes.

A score of **1.0** is achieved against the *test* set with `xgb.XGBClassifier(objective='multi:softmax', num_class=10)`, meaning **100%** of the test samples were correctly classified.

Naive Bayes

Naive Bayes is a simple yet powerful classification algorithm that is naturally suited for multiclass problems. It's based on Bayes' theorem and assumes independence among features, hence the term "naive." For this method, the **Gaussian approach** was selected due to the fairly normal distribution of the newly created dimensions and the small amount of features compared to the required by the Multinomial Naive Bayes approach.



Embedded dimensions KDE

A score of **1.0** is achieved against the *test* set with `GaussianNB()`, meaning **100%** of the test samples were correctly classified.

Thanks to effective data preprocessing and precise dimensionality reductions, all candidate models perform exceptionally well on the training data. Particularly noteworthy are the Gaussian Naive Bayes, the Multi-layer Perceptron classifier, and the XGBoost models. Each demonstrates high accuracy and proficiency in handling the data after preprocessing and dimensionality reduction. Consequently, any of these models could be considered as a strong candidate for the optimal classification model choice.

Fine-Tune the models

In this case, **Gaussian Naive Bayes** stands out for the relation between effectiveness and runtime while training. For the fine tune of the selected models, the `GridSearchCV` method is used.

The optimal model for `GaussianNB()` contains the default parameter of `var_smoothing=1e-09`

Cross Validation

Test score

By assessing the model's performance on new, unseen data, a score of **1.0** is obtained.

Repeteaded Train & Test Split

Iterating the train-test splitting process multiple times to obtain more stable and robust performance evaluations

```
Accuracy:100.00% std: 0.000
```

K-Folding

It involves splitting the dataset into 'k' equal-sized subsets or folds. The model is trained and evaluated 'k' times, each time using a different fold as the validation set and the remaining folds as the training set.

```
The meanof the 10 folds is 100.000% std of 0.000
```

Repeated K-Folding

It is an extension of k-fold cross-validation that introduces additional iterations to enhance the robustness of model evaluation. It involves running the k-fold cross-validation process multiple times with different random splits of the data into 'k' folds.

```
Accuracy:100.00% std: 0.000
```

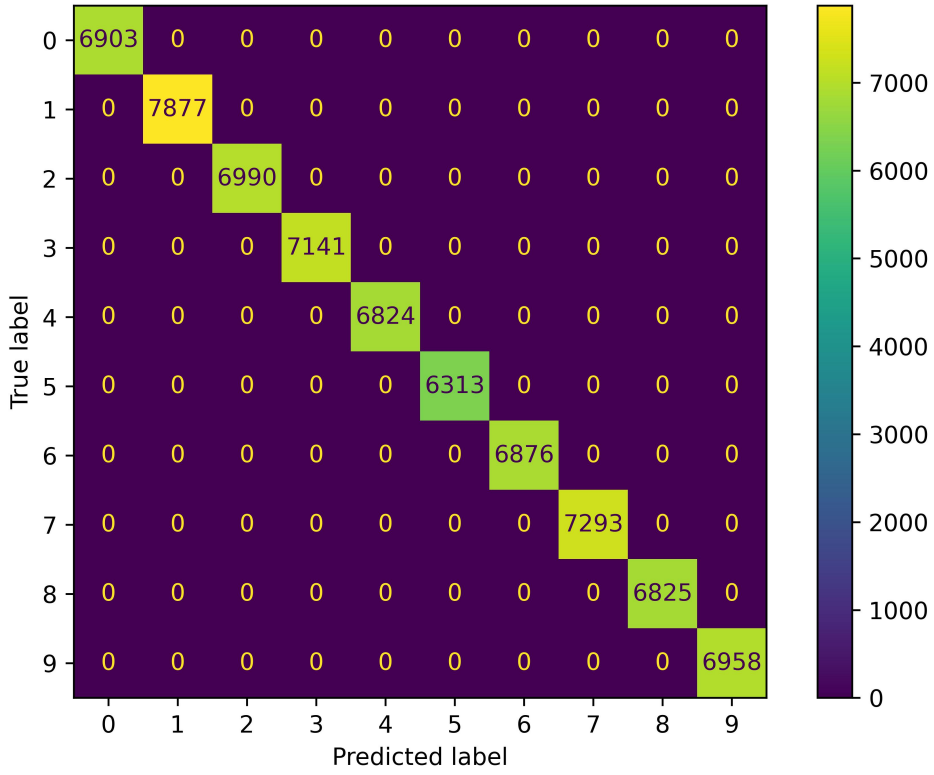
Cross validation conclussion

After conducting various cross-validation tests, the Gaussian Naive Bayes mantained its position as the selected algorithm. The next crucial step involves evaluating the chosen algorithm's performance.

Multiclass Metrics

Confussion Matrix

It is a powerful tool for evaluating the performance of a classification model, especially when dealing with more than two classes. It provides a visual representation of the model's performance by comparing the predicted labels against the true labels.



Confussion Matrix

A rare case of a perfect confussion matrix is shown above, displaying how there are no misclassified samples.

Precission, Recall, and F1-Score

By using `sklearn`'s `classification_report` we can observe the next metrics:

1. **Precision**: Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. It answers the question: "Out of all the instances the model labeled as positive, how many are actually positive?"

2. **Recall**: Recall is the ratio of correctly predicted positive observations to all the actual positives. It answers the question: "Out of all the actual positive instances, how many did the model correctly identify?"

3. **F1-Score**: The F1 Score is the weighted average of Precision and Recall. It tries to find the balance between precision and recall. It is useful in the case where we have an uneven class distribution as precision and recall may give misleading results.

In a multilabel classification context, these metrics are calculated for each label and then averaged - either by taking the mean average ('macro') or by weighting each label's score by the number of instances it has ('weighted').

	precision	recall	f1-score	support
0	1	1	1	6903
1	1	1	1	7877
2	1	1	1	6990
3	1	1	1	7141
4	1	1	1	6824
5	1	1	1	6313
6	1	1	1	6876
7	1	1	1	7293
8	1	1	1	6825
9	1	1	1	6958
accuracy	1	1	1	1
macro avg	1	1	1	70000
weighted avg	1	1	1	70000

ROC AUC Curve

The **Receiver Operating Characteristic Area Under Curve (ROC AUC)** score is a performance measurement for classification. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much a model is capable of distinguishing between classes. For this model, a score of **1.0** was obtained, which is the highest.

It would be an overkill to display all the ROC curves in this report since they all fall into the best fitting of the curve, it is coded in the github repositor regardless.

Cohen's Kappa Score

Cohen's Kappa Score is a statistical measure used to evaluate the level of agreement between two raters or judges who each classify items into mutually exclusive categories. In this case, the model obtained the maximum score which is **1.0**

Log Loss

Also known as cross-entropy loss measures the performance of a model by quantifying the difference between predicted probabilities and actual values by indicating how close a prediction probability comes to the actual/corresponding true value. The less loss the better, and the model obtained **1.99e-15** which essentialy is a **0**.

Conclussions

In conclusion, the results of the classification model were exceptional, demonstrating the power and importance of data preprocessing and dimensionality reduction techniques such as **PSA** and **t-SNE**. These techniques were instrumental in clustering the data effectively, which in turn significantly improved the performance of the classification model.

The model's performance was not only impressive in terms of multiclass metrics but also in terms of cross-validation. This indicates that the model is robust and likely to perform well on unseen data, a key indicator of a successful machine learning model.

However, it's important to note that while the use of dimensionality reduction techniques like PSA and t-SNE can lead to improved model performance, they also result in transformed features that lose their original meaning and interpretability. This is a trade-off that needs to be considered when deciding on the appropriate preprocessing steps for a given dataset.

Overall, this project underscores the importance of thoughtful data preprocessing and the selection of appropriate dimensionality reduction techniques. It serves as a reminder that while the choice of model is important, the way we prepare and transform our data can be equally, if not more, crucial to achieving high-performing models.

References

1. van der Maaten, L., & Hinton, G. E. (2008). Visualizing data using t-SNE. Journal of Machine Learning Research, 9(26), 2579-2605
2. Dembla, G. (2017). Intuition behind Log-loss score. Medium. Medium
3. Olah, C. (2014). Visualizing MNIST: An Exploration of Dimensionality Reduction. Colah's blog. Cola's blog
4. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.