INF8245E – KAGGLE COMPETITION FALL 2023

Achille Saint-Hillier
achille.saint-hillier@polymtl.ca
2081283

Michel Lominy
michel-1.lominy@polymtl.ca
2074487

Malaurie St-Amour-Bilodeau
malostamourbilodeau@gmail.com
2075534

ABSTRACT

This paper delves into the practical aspects of an image classification competition involving a dataset composed of 100 classes and 1024 features. We provide a clear overview of our data preprocessing techniques and the considerations guiding our model choices. The successes and challenges of selected models are candidly discussed. Furthermore, we present the outcomes of our participation in a Kaggle competition, highlighting the tangible results of our approach.

1 FEATURE DESIGN

The feature design and pre-processing phase equip the data with essential transformations to enhance its suitability for modeling. These practices collectively enhance model performance, reduce noise, and improve interpretability. We will apply various pre-processing techniques to establish a solid foundation for our model training. We're given 25,515 labeled samples for our training, and a test set of 10,000 unlabeled observations.

1.1 OUTLIERS DETECTION

Identifying and addressing outliers is a critical step in the data analysis process. Outliers, or data points significantly deviating from the majority, can distort statistical analyses and adversely impact model performance. In our case, we're only going to set aside outliers belonging to classes with more than 25 samples, given the limited data available for these smaller classes. Handling outliers in such cases may risk introducing bias or inaccuracies, as the smaller sample size makes it challenging to reliably distinguish between genuine anomalies and data variability.

Using an Isolation Forest Algorithm, we reduced the number of samples from 20412 to 19959, removing 453 data points identified as outliers.

1.2 DATA NORMALIZATION

We can normalize the data to ensure that all features are on a consistent scale. Normalization is a crucial preprocessing step that helps mitigate the impact of varying magnitudes among different features. In our case, the focus is on achieving consistency in feature scales. This process ensures that each feature has a mean of 0 and a standard deviation of 1, promoting stable and effective learning across different machine learning models. We explored different strategies for normalization. Usually, the same normalization parameters are then applied to the validation and test sets to maintain uniformity in the scaling process across all subsets of the data. However, we observed that normalizing solely the train data gave us better results than the usual approach.

1.3 Oversampling and Undersampling

Next, we looked at class distribution. This process involves assessing the distribution of instances across different classes to identify any imbalances.

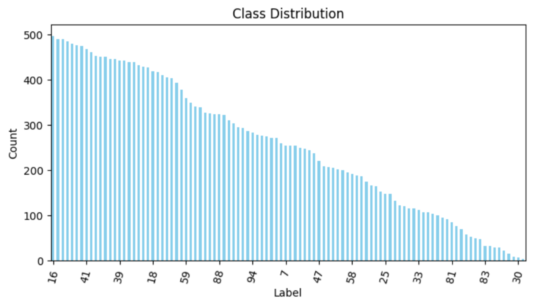

Figure 1: Class distribution before pre-processing

As we can see (Figure 1), our dataset is highly unbalanced, with 498 samples for the class 16 and only 3 samples for the class 82. We need to address this issue, since our model may struggle to effectively learn from minority classes with fewer instances.

Oversampling is generally done using SMOTE - Synthetic Minority Over-sampling Technique, creating synthetic minority class samples. However, our data contains so few samples for certain classes (our training set only contains two samples of the class 82 for example) that the use of this technique resulted in poorer prediction than a model that used no oversampling.

This is the reason why we opted for a custom oversampling technique: for each feature, we'll compute the average distribution across all samples. For a given feature, after identifying the optimal distribution for each class, we choose the predominant distribution type and calculate the average parameters. This process is repeated for each feature, resulting in a distribution array of length 1024. This array allows us to generate synthetic ”average data points” by sampling observations for each distribution. Using these synthetic points, we perform a weighted average with actual data points from the class we want to sample. This technique leads to oversampling that is centered around the actual data points of the class, skewed to resemble the average distribution of the specific feature.

We identified that most features follow a Gaussian or an exponential distribution, thus why we use norm, expon and exponnorm as possible distributions. We compute the Akaike Information Criterion (AIC), which is a method for evaluating how well a model fits the data it was generated from.

This leaves us the 100 distributions that best fit the corresponding feature for each class.

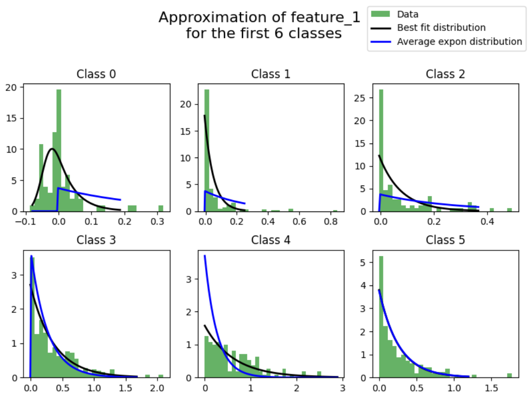

Figure 2: Best approximation of the first feature for classes 0 to 5

The classes of the training dataset that have less than 250 samples will be oversampled and the ones with more than 250 samples will be undersampled to ensure a balanced distribution.

1.4 Dimensionality reduction

Our last step of preprocessing is dimensionality reduction, which is aimed at mitigating the curse of dimensionality and enhancing the efficiency of subsequent modeling processes. By reducing the number of features while retaining essential information, we seek to address potential issues such as overfitting, computational complexity, and the ”noise” associated with high-dimensional datasets. We’re going to use Principal Component Analysis (PCA). Using 768 components (75% of the original 1024 features), we reach a cumulative explained variance of 99.3%.

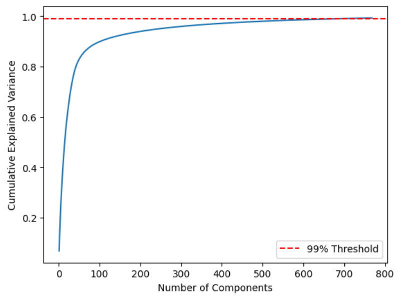

Figure 3: Influence of the number of features over the Cumulative Explained Variance

This reduces the training time by a quarter without losing too much information about the data. However, our models didn’t perform any better with dimensionality reduction, whether it be on the validation set or the test set, so we didn’t use the reduced data to train our models.

2 Algorithms

In this section, we’ll delve into the different algorithms we used. We’ll talk about the ones that gave the best performance, and that we used for our final model. The ones that didn’t perform as good, but that we still tried, are depicted in Appendix A. For each algorithm, we’ll try to explain why they performed the way they did, as well as giving our prior intuition on their performance.

2.1 Logistic Regression

Given the high dimensionality of our dataset, we initially had reservations about the efficacy of logistic regression, as it typically excels when dealing with low-dimensional datasets. Logistic regression is well-suited for scenarios where the number of features is moderate and there is a linear relationship between the features and the target variable. In our case, however, the dataset’s high dimensionality posed a challenge for logistic regression to capture complex relationships effectively. Despite these initial concerns, we decided to experiment with logistic regression to assess its adaptability to our dataset. The primary motivation behind this decision was its ease and speed of training, particularly when employing a one-vs-rest method. Logistic regression, with its simplicity and efficiency, allows us to quickly iterate and experiment with different configurations, making it a pragmatic choice for initial assessments.

To our surprise, logistic regression demonstrated notable performance, showcasing its resilience even in high-dimensional spaces. It managed to capture essential patterns within the data, making it a valuable contender in our algorithmic ensemble. The success of logistic regression in our context might be attributed to the inherent characteristics of our dataset or the presence of linear relationships within specific feature subsets that we failed to recognize in our analysis of the data.

2.2 Bagging

Combined with Logistic Regression, we utilized Bagging, or Bootstrap Aggregation. Bootstrap datasets are randomly created from the original dataset. They possess the same number of data points, but some are randomly duplicated to ensure differences between the datasets. This algorithm uses ‘M’ bootstrap datasets to form a committee that predicts the output. Bagging is particularly effective in our case due to its ability to reduce overfitting and variance. By creating diverse datasets through random sampling with replacement, Bagging mitigates the impact of noisy data points. It performs really well in combination of our custom oversampling that generates intended noise, that might be inappropriate for certain classes. The ensemble of models generated from these datasets works collaboratively to provide a more robust and generalized prediction, enhancing the overall performance and reliability of our model.

3 Methodology

3.1 Data analysis

Our approach commenced with an analysis of the dataset. We scrutinized its structure, variables, and distributions, aiming to unearth insights into the characteristics of the data. This involved exploring summary statistics, visualizing key features, and identifying potential challenges such as missing values, imbalances or outliers. We also made a diligent effort to identify any distinctions between the training and testing datasets, as an early observation suggested disparate behaviors. However, our thorough examination did not reveal any substantial differences worth noting. Although normalizing the data led to a slight improvement in our results, pinpointing a clear and discernible difference was not possible.

3.2 Training and Validation Split

The training set serves as the foundation for model learning, the validation set aids in fine-tuning and selecting the best model without overfitting on the training data. The data was partitioned, with 80% dedicated to the training set and 20% to the validation set. To maintain consistency in our evaluations, we employed a predefined random seed throughout our work, ensuring reproducibility in our analyses and results.

3.3 Evaluation Metrics

For assessing the performance of our machine learning model, we employed the weighted F1 Score as a key evaluation metric. The weighted F1 Score takes into account both precision and recall, offering a joint measure that considers the influence of class imbalances. The same metric is used for ranking the different predictions in the Kaggle competition. This choice ensures consistency between our model assessments and the competition’s evaluation criteria.

3.4 Model Exploration

The process of model exploration involves experimenting with various machine learning models to identify the most suitable architecture for the given task. This phase allows for an assessment of model performance, enabling the selection of the most effective approach based on the dataset and problem at hand. During our exploration, we first tried the models we initially thought would perform the best. We chose the best model based on the metrics outlined in the Evaluation Metrics section for further evaluation and tuning.

3.5 Hyperparameter Tuning

Fine-tuning the hyperparameters is a critical step in optimizing the performance of machine learning models. It involves adjusting the configuration settings that are external to the model itself, impacting its learning process. Techniques such as grid search or randomized search were employed to systematically explore the hyperparameter space and identify the optimal combination that enhances model performance. This intensive tuning process contributed significantly to achieving the best possible predictions for our model.

4 Results

Our predictions resulted in a score of 0.80515 on the public leaderboard, elevating us to the 6th position.

It's important to note that we didn't realize one of the functions employed in our oversampling process utilized a random seed, leading to a situation where our predictions are not entirely replicable. Despite this, the results remain highly consistent, with only minor variations observed.

Through grid search, our most successful model emerged as a bagging classifier combined with logistic regression, employing 35 estimators for bagging and a regularization strength of 0.5 for logistic regression. This configuration yielded a notable F-Score of 0.9401 on the validation set. However, it is noteworthy that the F-Scores of the majority of our alternative models hovered within the range of 0.92 to 0.94 on the validation set.

Table 1: Performance of Different Models

Model	**Best validation F-Score**
SVC (Support Vector Classification)	0.9279
RF (Random Forest)	0.9310
CNN (Convolutional Neural Network)	0.9393
LR (Logistic Regression)	0.9352
Bagging (with LR)	**0.9401**
MLP (Multi-layer Perceptron)	0.9231
KNN (K-Nearest Neighbors)	0.8561

We can do a quick comparison between the best hyperparameters we found. The Bagging algorithm with 35 estimators demonstrates a slight improvement compared to 10 estimators, achieving a cross-validation score of 0.951 with the whole dataset, as opposed to 0.949. Additionally, it exhibits superior performance on the training set, indicating effective learning without overfitting. Notably, the Bagging algorithm with 35 estimators achieves better scores while requiring less data compared to the configuration with 10 estimators.

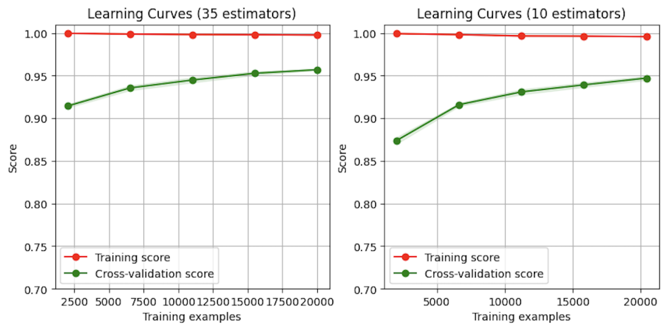

Figure 4: Comparison between 35 and 10 estimators for bagging

A comparable analysis can be conducted for two logistic regression models—one with a regularization strength of 0.5 and the other with 1.0. Similar to our observations with bagging, the logistic regression classifier employing a regularization strength of 0.5 outperforms its counterpart, yielding higher cross-validation scores while requiring fewer training examples.

Opting for a stronger regularization is particularly advantageous, considering the distinctive behavior of the test set and the training set. The presence of a slight underfit further amplifies the performance

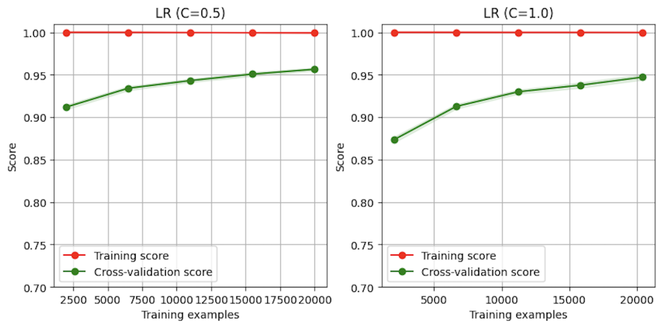

Figure 5: Comparison between regularization strength for logistic regression

of our model, contributing to its effectiveness in handling variations between the training and test datasets.

5 Discussion

Our methodology demonstrated strengths in meticulous data analysis, addressing potential issues, including but not limited to data imbalance, outliers, and non-normalized data. Additionally, we started to submit a lot of predictions from different model pretty early in the competition. However, this led us to try and explore alternative models when faced with poor performance, rather than investing more time in refining and optimizing the existing models

An area for improvement is the need for rigorous hyperparameter fine-tuning for each model. This step could optimize model performance by better aligning with the dataset's nuances.

In retrospect, a more thorough consideration of the time and computational resources required for model selection and optimization is essential, as our initial underestimation posed challenges in navigating these processes efficiently.

Recognizing the significant difference between our local training performance (0.94) and the test set performance (0.8) is crucial for future considerations. The consistent behavior of various models highlights the importance of understanding the distinctions between the test and train sets. It's essential to ensure our models don't overfit to the training set specifics. Addressing this difference is a priority, as it could impact the reliability of our models with unseen data. Moving forward, focusing on factors contributing to this divergence will guide us in refining our models for better performance and generalization.

A Appendix

A.1 Other algorithms tried

A.1.1 K Nearest Neighbor

We tried to visualize our data points in 2 and 3 dimensions, as you can see in Figure 4 and Figure 5. We used t-distributed Stochastic Neighbor Embedding (t-SNE), which is an unsupervised non-linear dimensionality reduction technique for data exploration and visualizing high-dimensional data. We expected the K Nearest Neighbor (KNN) approach to work well, given how locally smooth our visualization looked. However, KNN was not as effective in our case, primarily due to the high dimensionality of our data. We learned the hard way not to trust too much t-SNE visualization. The inherent challenge posed by numerous features in our dataset hindered KNN's ability to discern meaningful patterns. Despite our attempts to address this by employing dimensionality reduction techniques, the performance of KNN did not meet our expectations. The complexity introduced by the multitude of features might have led to a less effective utilization of the nearest neighbors in the high-dimensional space.

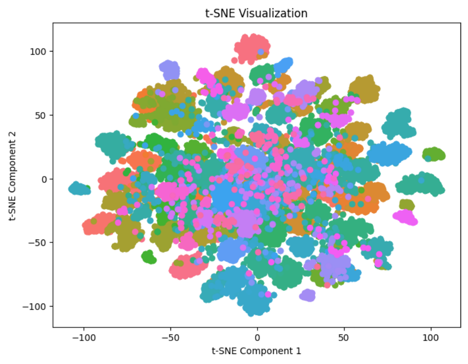

Figure 6: t-SNE visualization of the training data in 2D

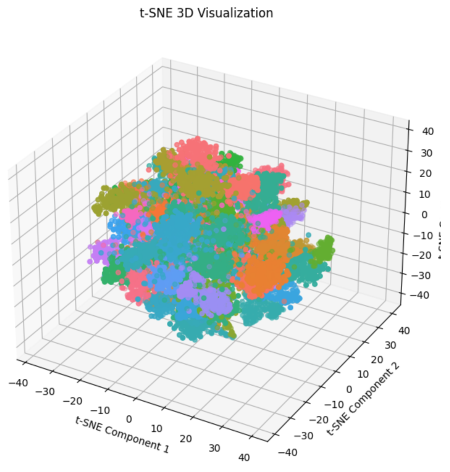

Figure 7: t-SNE visualization of the training data in 3D

A.1.2 Random Forest

We used a Random Forest classifier, a robust ensemble technique that combines the predictions of N diverse decision trees, each trained on subsamples of the dataset. Additionally, we enhanced our results by applying Adaboost to boost the performance of the Random Forest. We thought that boosting through Adaboost could amplify the efficacy of the Random Forest in capturing intricate patterns within the data, but the results were poor.

A.1.3 CNN

Utilizing TensorFlow, we used a Convolutional Neural Network (CNN) because it is adapted for image. CNNs are specialized neural networks designed to excel in image processing by employing convolutional layers to scan the input image for patterns and features. However, in cases where there's not a lot of ambiguous feature or pattern, this type of model may overfit to noise present in the training data. This can lead to poor generalization and reduced performance on new, unseen images.

A.1.4 Multi Layer Perceptron

We tested a multi layer perceptron algorithm to classify the data. The MLP algorithm, being a feedforward neural network, employs activation functions to introduce non-linearity to the model. Despite our initial expectation that MLP would excel in capturing complex patterns, it was tested alongside bagging and yielded predictions inferior to those obtained with logistic regression.

A.1.5 Stacking

Stacking combines the predictions of different base models with a metamodel that outputs the final prediction. It is quite useful when the data is complex and is better captured by multiple models rather than a single one. We tested this algorithm on multiple models to take advantage of all their characteristics but it did not improve the predictions.