CIFAR-10 Multiclass Classification

Machine Learning

Master's degree in Computer Science and Engineering [LM-32]
Master's degree in Artificial Intelligence [LM-18]

Francesco Marastoni
Daniele Pasotto

AY 2024/2025

Contents

1 Motivation and rationale **1**

2 State Of The Art **1**

3 Objectives **1**

4 Methodology **2**
4.1 Dataset 2
4.2 Preprocessing 3
4.3 Pipeline Composition 4
4.4 Grid-Search Cross-Validation 4
4.5 Models 5
4.5.1 SVM 5
4.5.2 Random Forest 6
4.5.3 Logistic Regression 6
4.5.4 K-Nearest Neighbors 7
4.5.5 Gaussian Naive Bayes 7

5 Analytical and Computation Tools **8**

6 Results **9**
6.1 Confusion Matrix 9
6.2 Comparisons 12

7 Conclusions **13**

1 Motivation and rationale

The project will study the multi-class classification problem using the most well-known Machine Learning models. By doing so, one can see the strengths and improvement points, that these approaches have to offer. The motivation behind this project is twofold. On the one hand, the project assesses the capabilities of different Machine Learning techniques in handling multi-class classification. On the other hand, the project aims to deepen our understanding of their strengths and limitations in real-world scenarios. By comparing their effectiveness, one can gain insights into their real-world applicability and identify areas for improvement.

2 State Of The Art

Multi-class classification is done using the most common state-of-the-art machine learning approaches. The best result can be obtained using the SVM classifier combined with the dimensionality reduction technique done by PCA. Even if the result is poor in terms of accuracy, an improvement can be done for sure using a deep learning method, for example, to make feature extraction.

3 Objectives

The goals of this project are to analyze the CIFAR-10 [1] dataset, and train different machine learning models using a supervised technique, to have a comparative view. Finally, the report will draw conclusions from the data obtained. In summary, the points that cover this work are:

- Analyze the CIFAR-10 dataset;
- Train different machine learning models for comparison;
- Compare the performance and other statistics on the models;
- Draw conclusions from the obtained data.

4 Methodology

4.1 Dataset

The dataset used is CIFAR-10, which consists of 60000 colour images of the size of 32x32. This dataset is often used as a benchmark dataset [2] for various models, both in deep learning and machine learning. The dataset is composed of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), with 6000 images per class. In total, there are 50000 training images and 10000 test images. The classes are completely mutually exclusive (there is no overlap between classes that may be similar).

Figure 1: CIFAR-10 dataset

A graphical representation of a subset of the dataset can be found below, using the T-SNE algorithm to visualize the data in two dimensions. It can be seen that the classes are very scattered and that there is no immediate separation possible between the ten classes. Considerable overlap between different classes is also clearly visible.

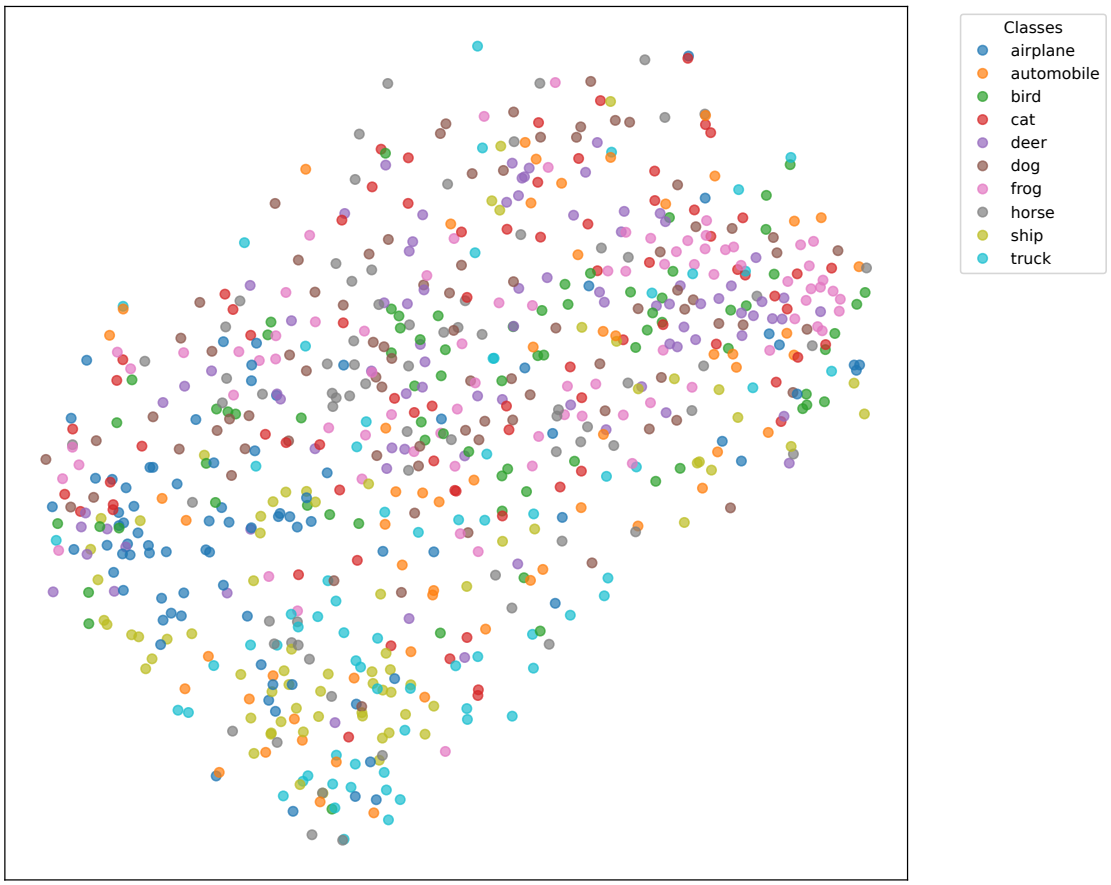


Figure 2: T-SNE Visualization of CIFAR-10 Training Data (750 Samples)

4.2 Preprocessing

The training and test sets are preprocessed by applying data augmentation. In particular, on training images normalization, random crop, horizontal flip, rotation and colour jitter are applied; whereas on test images only normalization is applied. The mean and standard deviation used for normalization come from values calculated on this dataset and well known to everyone.

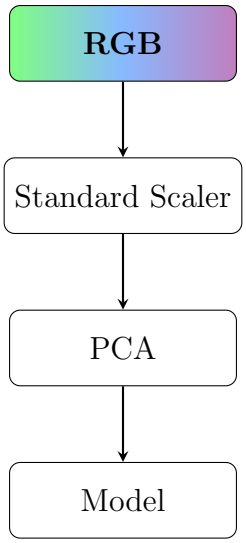
After data augmentation, images' pixel values are scaled between values of [0, 1] to ensure more stable and efficient training. Finally, the data is flattened because all models are fed with one-dimensional input.

4.3 Pipeline Composition

To avoid redundancy in the code, a pipeline is created for every model to train and test on the dataset. The pipeline is composed by:

- **Standard Scaler**: transforms the entire dataset to zero mean and unit variance to ensure stability during training and testing;
- **Principal Component Analysis (PCA)**: every image is composed of 32x32x3 (3072) features, which are a large amount of data for training. To avoid long processes in terms of time, PCA is applied to reduce the dimensionality of the data preserving 95% of the variance. This process also removes redundant features;
- **Classifier**: the model used for the classification.

The simplified pipeline used is as follows:



4.4 Grid-Search Cross-Validation

Every model is composed of many parameters and the goal of this project is to find the best classifiers and compare them. To make this a grid search method is applied to automatize the tuning of hyperparameters for every classifier. This method takes in input a grid of hyperparameter values to test and then train and evaluate the model using k-fold cross-validation (5 folds) for each combination. The best hyperparameters are selected based on performance (accuracy score).

4.5 Models

The multi-class classification task is made by using two different strategies:

- **One-vs-Rest classification**: it splits the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident. In total there will be as many classifiers as the number of classes;
- **One-vs-One classification**: it splits a multi-class classification dataset into binary classification problems. The one-vs-one approach splits the dataset into one dataset for each class versus every other class. In this case, the number of classifiers is calculated with this formula:

$$\frac{n\_classes \cdot (n\_classes - 1)}{2}$$

4.5.1 SVM

The most important hyperparameters to set in an SVM with an **RBF kernel** are the **regularization parameter C** and the **kernel coefficient gamma**. The C hyperparameter controls the trade-off between maximizing the margin and minimizing the classification error. The gamma hyperparameter controls how much influence a single training example has on the decision boundary. If these parameters are too big the risk is to lead in overfitting; on the other hand, if the parameters are too small the risk is to lead in underfitting.

The best parameters found for these models were:

- One VS Rest (Best score during the train: 56.85%):
  - C: 10
  - Gamma: `scale`
- One VS One (Best score during the train: 55.52%):
  - C: 10
  - Gamma: `scale`

4.5.2 Random Forest

The most important hyperparameters to set in a Random Forest are the **n_estimators**, the **max_features** and the **min_samples_leaf**. The first hyperparameter controls the number of trees in the forest, and if it is a lower value there is the risk to lead in underfitting. The second one controls the number of features to use at each tree split and for higher values the risk is to lead to overfitting. The last hyperparameter sets the minimum number of samples to form a leaf of the tree and for lower values, the risk is to lead to overfitting.

The best parameters found for these models were:

- One VS Rest (Best score during the train: 45.83%):
  - n_estimators: 100
  - max_features: `sqrt`
  - min_samples_leaf: 4
- One VS One (Best score during the train: 46.42%):
  - n_estimators: 100
  - max_features: `sqrt`
  - min_samples_leaf: 4

4.5.3 Logistic Regression

The most important hyperparameters to set in Logistic Regression are the **inverse of regularization strength C**, the **penalty** and the **solver**. The C hyperparameter controls the amount of regularization applied to the model (works like SVM). The penalty hyperparameter specifies the norm of penalty applied to prevent overfitting. The last hyperparameter specifies the algorithm to use in the optimization problem.

The best parameters found for these models were:

- One VS Rest (Best score during the train: 40.85%):
  - C: 0.1
  - Penalty: `l1`
  - Solver: `liblinear`
- One VS One (Best score during the train: 41.64%):
  - C: 0.1
  - Penalty: `l1`
  - Solver: `saga`

4.5.4 K-Nearest Neighbors

The most important hyperparameters to set in K-Nearest Neighbors are the **n_neighbors**, the **weights** and the **metric**. The number of neighbours hyperparameter specifies how many nearest neighbours are involved when making a prediction; if it is a lower value there is the risk to lead in overfitting and for higher values the risk is to lead in underfitting. The weights hyperparameter determines how much influent is each neighbour for the prediction. The last hyperparameter specifies what metric is used to calculate the distance between points.

The best parameters found for these models were:

- One VS Rest (Best score during the train: 37.20%):
  - n_neighbors: 5
  - weights: `distance`
  - metric: `minkowski`
- One VS One (Best score during the train: 36.71%):
  - n_neighbors: 3
  - weights: `distance`
  - metric: `minkowski`

4.5.5 Gaussian Naive Bayes

The most important hyperparameter to set in Gaussian Naive Bayes is the **var_smoothing**. This hyperparameter adds to the variance of each feature a small smoothing parameter to prevent division by zero or very small values. For lower values of this parameter, there is a risk to lead to overfitting.

The best parameters found for these models were:

- One VS Rest (Best score during the train: 30.16%):
  - var_smoothing: $1e^{-11}$
- One VS One (Best score during the train: 30.52%):
  - var_smoothing: $1e^{-11}$

5 Analytical and Computation Tools

The major libraries used for the project are:

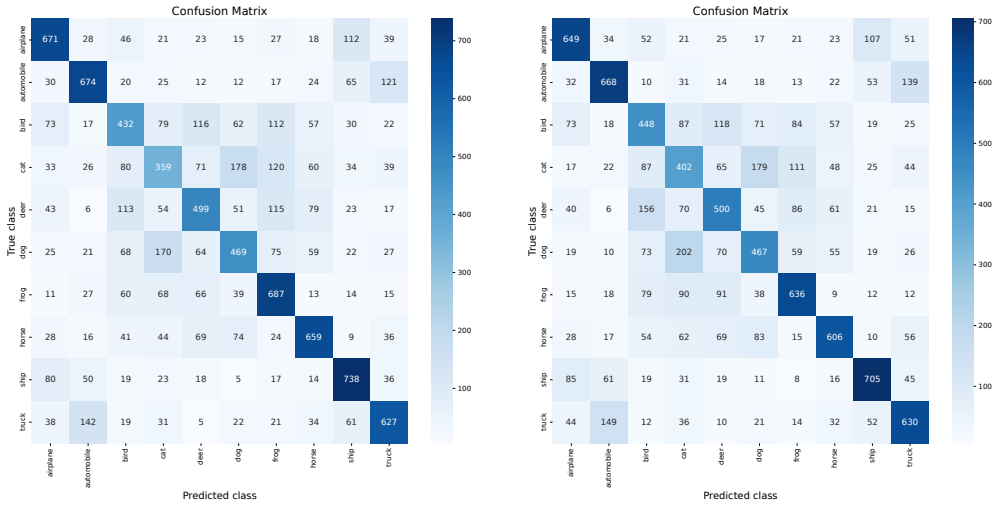
- `scikit-learn`: Fixed version v1.3.2. Used for implementing all the machine learning algorithms
- `scikit-learn-intelex`: Enables `scikit-learn` to use the full power of the processor. Used for SVMs' intense training.
- `torchvision`: Used for dataset gathering.
- `matplotlib`: Used to plot results in the form of graphs.
- `pandas`: Used for tabular vision of the final scores.
- `seaborn`[3]: Used for plot confusion matrices.

6 Results

6.1 Confusion Matrix

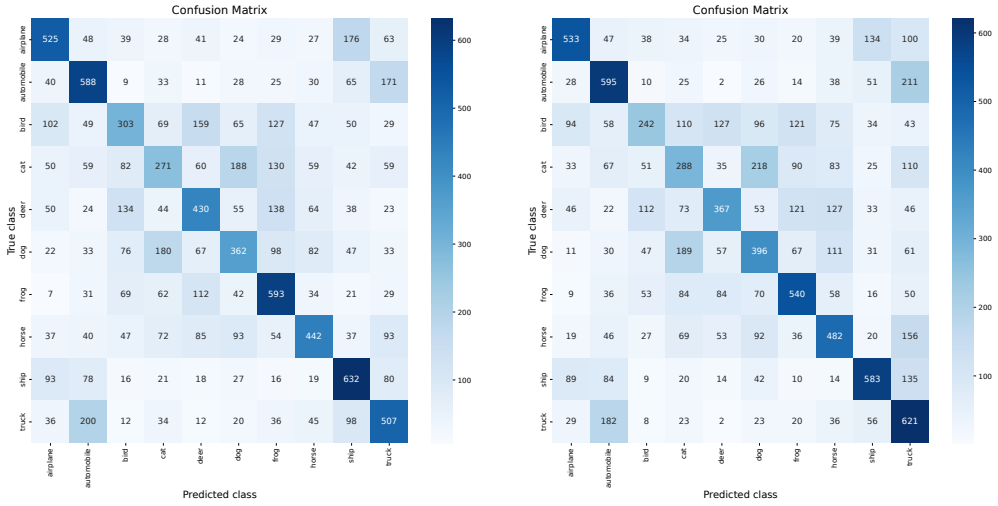
The confusion matrix is used to visualize the performance of the classifier by displaying the true labels vs. predicted labels.

From these confusion matrices can be seen that there aren't big differences between the one-vs-rest and the one-vs-one approach.



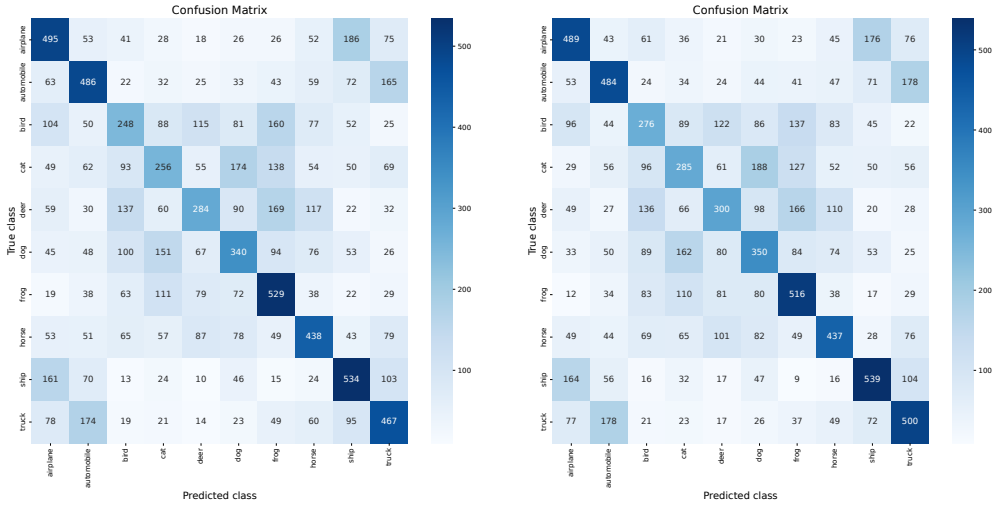
(a) Support Vector Machine (OvR) (b) Support Vector Machine (OvO)

Figure 3: Comparison of Support Vector Machine methods



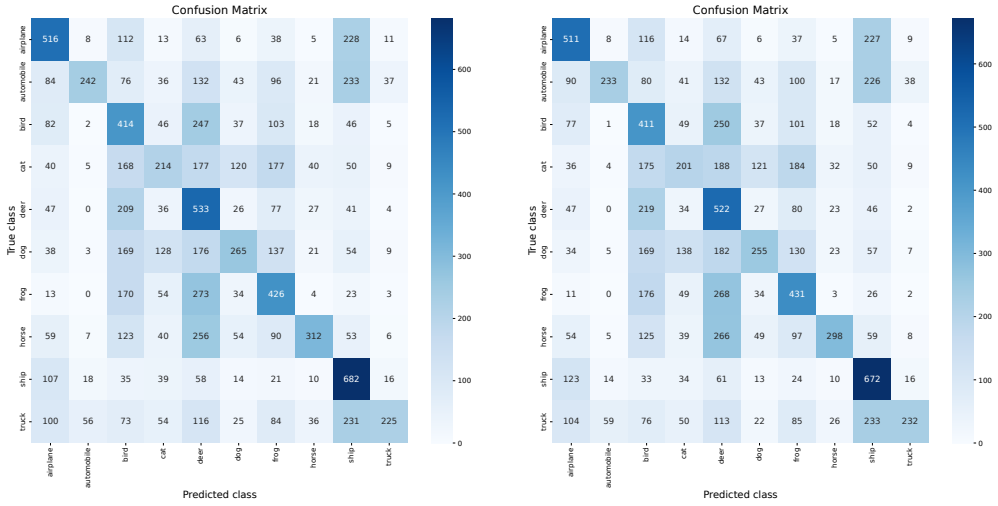
(a) Random Forest (OvR) (b) Random Forest (OvO)

Figure 4: Comparison of Random Forest methods



(a) Logistic Regression (OvR) (b) Logistic Regression (OvO)

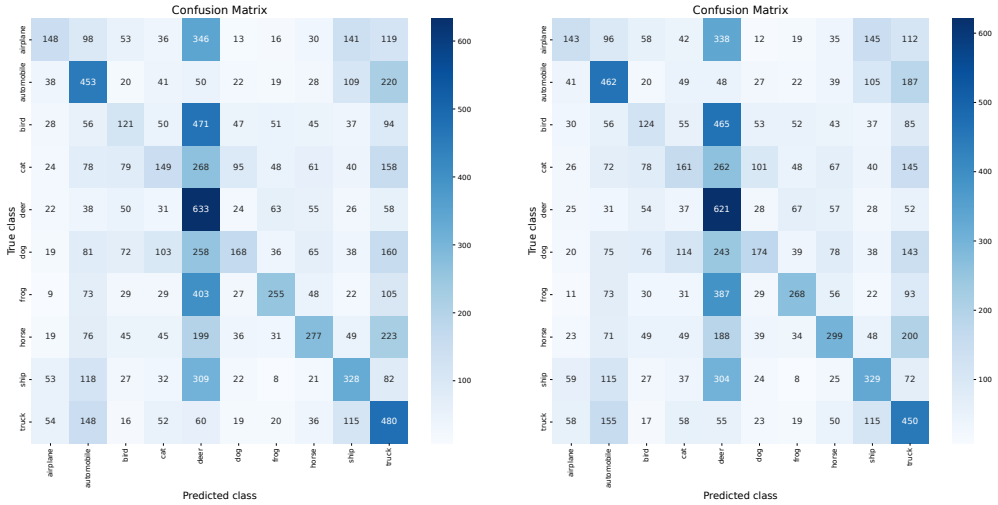
Figure 5: Comparison of Logistic Regression methods



(a) K-Nearest Neighbor (OvR) (b) K-Nearest Neighbor (OvO)

Figure 6: Comparison of K-Nearest Neighbor methods

A classifier whose classification is very poor is the KNN. From its matrix can be seen that classify very well one class (ship), meanwhile, the others have a lower prediction score.



(a) Gaussian Naive Bayes (OvR) (b) Gaussian Naive Bayes (OvO)

Figure 7: Comparison of Gaussian Naive Bayes methods

Another classifier which classifies very badly on the test set is the GNB. Focus on its matrix, only one class (deer) is classified more correctly with respect to the other classes.

6.2 Comparisons

Model	Accuracy	F1 Score	Precision	Recall
Support Vector Machine (OvR)	0.5815	0.5778	0.5766	0.5815
Random Forest (OvR)	0.4653	0.4596	0.4595	0.4653
Logistic Regression (OvR)	0.4077	0.4020	0.4011	0.4077
K-Nearest Neighbors (OvR)	0.3829	0.3769	0.4544	0.3829
Gaussian Naive Bayes (OvR)	0.3012	0.2875	0.3321	0.3012

Table 1: Performance comparison for One Vs Rest classifiers on test data

Model	Accuracy	F1 Score	Precision	Recall
Support Vector Machine (OvO)	0.5711	0.5708	0.5710	0.5711
Random Forest (OvO)	0.4647	0.4595	0.4657	0.4647
Logistic Regression (OvO)	0.4176	0.4143	0.4132	0.4176
K-Nearest Neighbors (OvO)	0.3766	0.3709	0.4535	0.3766
Gaussian Naive Bayes (OvO)	0.3031	0.2907	0.3272	0.3031

Table 2: Performance comparison for One Vs One classifiers on test data

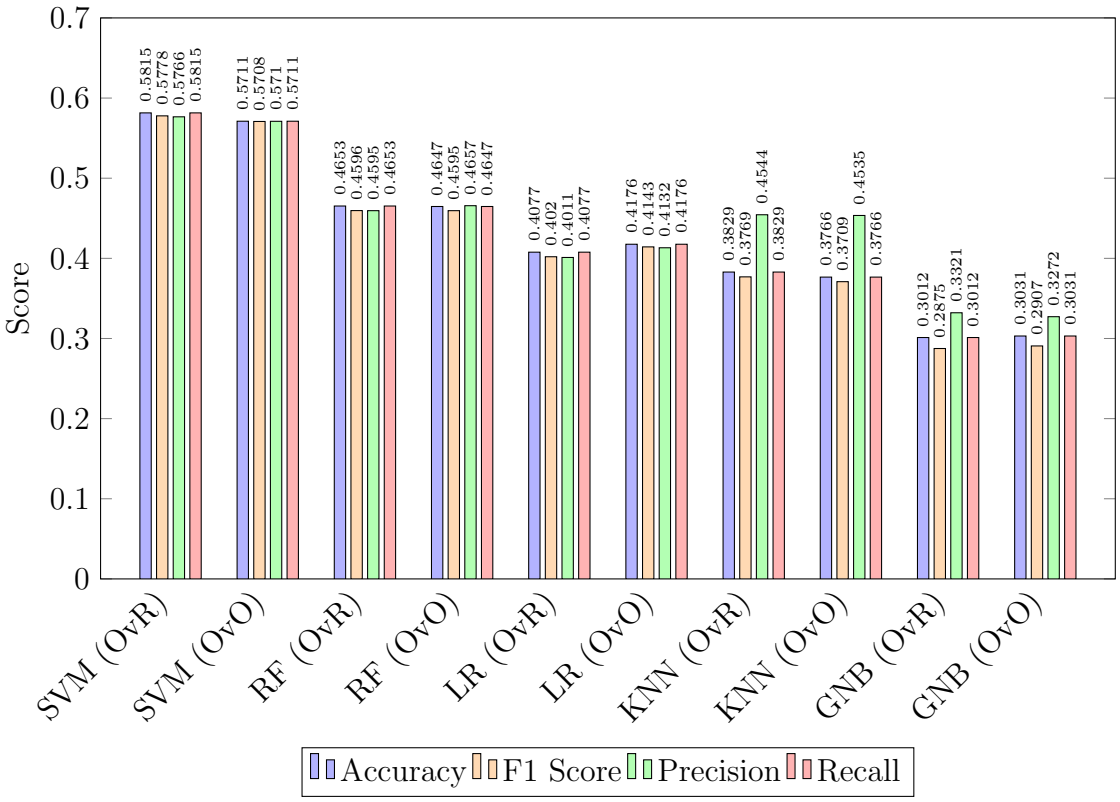


Figure 8: Comparison between models

From the plot can be seen that the KNN model has an unusual precision, different from the rest of its other metrics. The dataset might have non-local patterns, such as global correlations or dependencies between features, this makes a local method like KNN fail, but makes an algorithm like Random Forest prevail. For such high precision, positive data is expected to be sparsely distributed.

It can also be seen that, for the linear Logistic Regression model, the results obtained through the OvO technique are slightly better than those obtained through OvR. This is not true for SVMs, a kernel-based model, which perform better through the OvR technique than OvO. It is therefore possible to assume that, the SVM model, found instability in the vote aggregation phase, thus concluding that boundaries between some classes are intricate.

7 Conclusions

The project thus aimed at image classification using Machine Learning models. The report analyzed five models in one-vs-rest and one-vs-one setting with different hyperparameter tuning methodologies. It can be seen that the best-performing models are the ones expected from the predicted results, namely SVM and Random Forest. The major problem with this dataset may be the resolution of the images, which is as low as 32x32 may give problems. The project has room for improvement, through the use of Deep Learning methodologies.

References

[1] University of Toronto, *The CIFAR-10 dataset*. Available at this link: https://www.cs.toronto.edu/~kriz/cifar.html

[2] Papers With Code, *CIFAR-10 Benchmark Overview*. Available at this link: https://paperswithcode.com/sota/image-classification-on-cifar-10

[3] Seaborn, *Seaborn for data visualization*. Available at this link: https://seaborn.pydata.org