ULB FACULTÉ DES SCIENCES

INFO-H501 MACHINE LEARNING PROJECT - CHRISTOS DAOULAS

1. Introduction

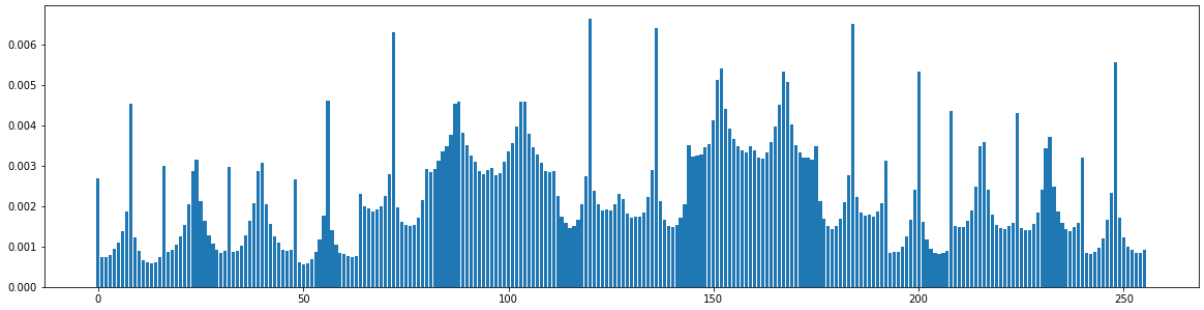
In this report, I recount the methods followed in comparing four algorithms in the label recognition of 18000 images in the CIFAR dataset. As the dataset is divided between a train and a test set, the latter will be used for final validation after hyperparameters of the algorithms have been fine-tuned by relying on the training set. In what follows, I will describe the Data Exploration, Data Description, Pre-Processing and Algorithm Evaluation and Algorithm Comparison amongst the four chosen algorithms, namely: K-nearest neighbours, Decision Tree, Random Forest, Neural Networks, and Support Vector Machines.

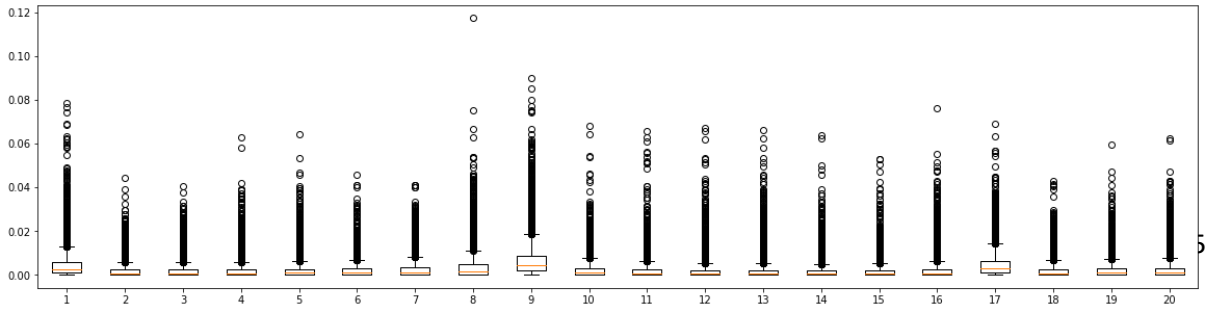
2. Dataset Exploration

After loading a subsample of the CIFAR 10 image dataset, we start by taking a look at their histograms of gradients. The images are 32x32 8-bit RGB, so the extracted Histogram of Gradient vector has 16 orientations x 16 blocks = 256 values. We can see that these three classes, namely 'Airplane', 'Bird', 'Horse', are quite balanced in the training set consisting of 5000 instances each, therefore we will not have to add any weights to the different images during training. This point is also important for the latter evaluation metrics. Since we are dealing with a classification problem with a balanced dataset, “accuracy” would be an interesting metric that we will be using.

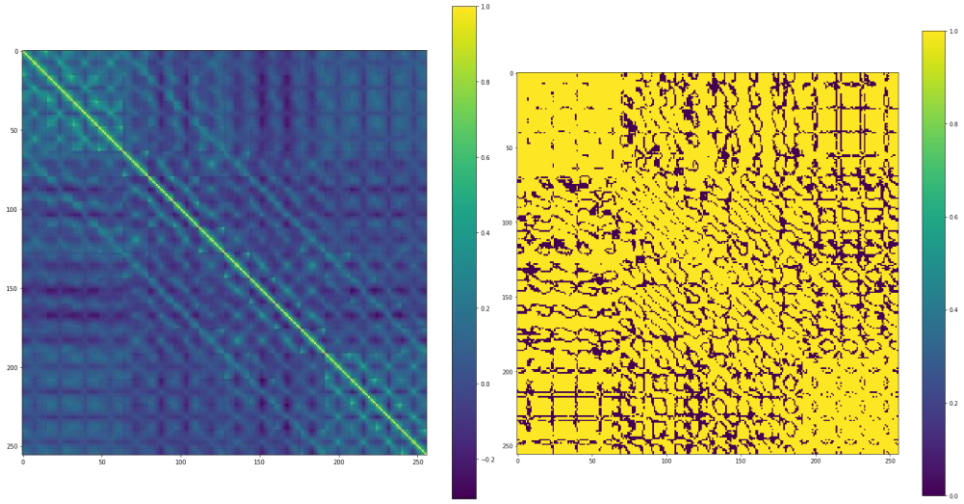
3. Descriptive Data Analysis

When we compute the histogram of gradients, we have a relative proportion of the gradients present in the image that have a specific orientation. It is normalised across one image, so the sum of histogram of the gradients of the features of one image are equal to one. We can start by looking at the median value for each feature across the entire dataset. Apparently, we have small values, so in order to see the distribution we can use a boxplot. The vast majority of features have a median value close to zero, with the exception of a small number of outliers like and we have some outlier value like 0.12





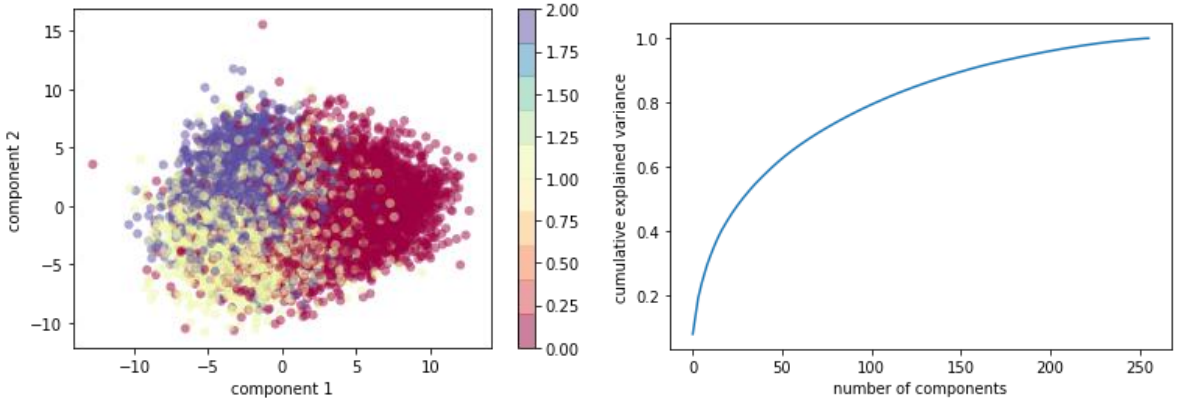
In order to ease computations, we will see if we can reduce dimensionality of the data by dealing with collinearity. For that reason, we will be using the spearman and Pearson test to determine the existence of correlated features. In the second graph, we have used a p-value<0.005 threshold:



We can obviously discern a pattern, revealing correlation between orientations together in the same cell or between variables with the same orientation in the neighbouring cells, as such there is a fair amount of correlation.

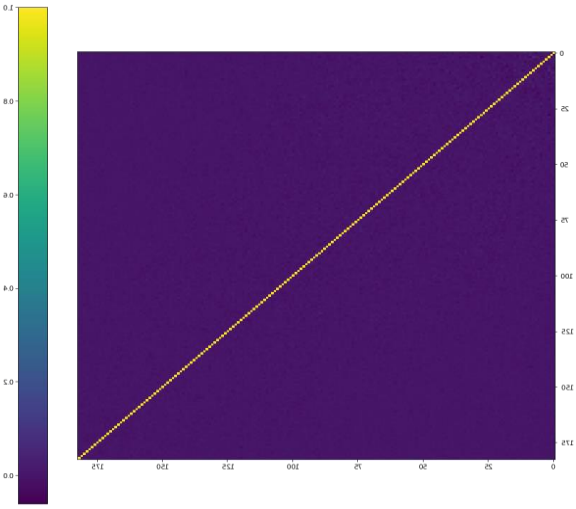
4. Data Pre-processing

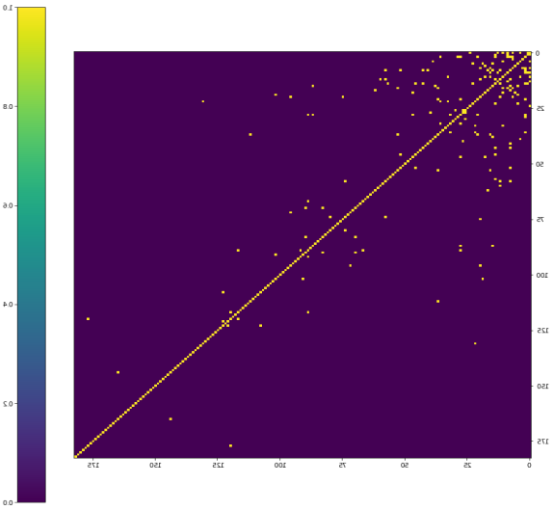
Given the aforementioned correlation, we will be using PCA in order to reduce dimensionality and increase computational efficiency and accuracy. Prior to that we will have to normalise the data, as since PCA is based on SVD thus relies on the standard deviation of the features, higher standard deviations will have a higher weight for the calculation of projection axis of the Principal Component, so normalisation harmonises standard deviations. First, we normalise that training data and test the two first principal components. We then compare explanatory power amongst number of components to see if we need to choose more PCs:



Indeed, we see that the two first components explain only a very small part of the variation. Therefore, we need to find more principal components. We discern that for a 90% explanation of variability, we can reduce dimensionality to 183 from 256, which increases efficiency by almost 30%. We apply the same techniques for the test set, i.e. normalising and choosing the same number of

components for the PCA algorithm and then repeat correlation tests: Looks like we are in the clear! Now we will evaluate the hyperparameters for our chosen algorithms.

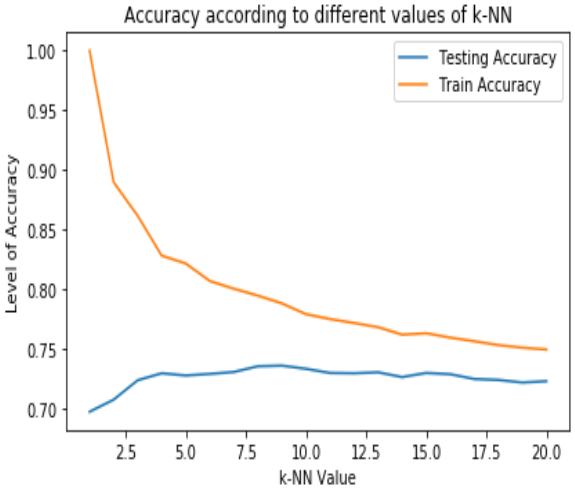


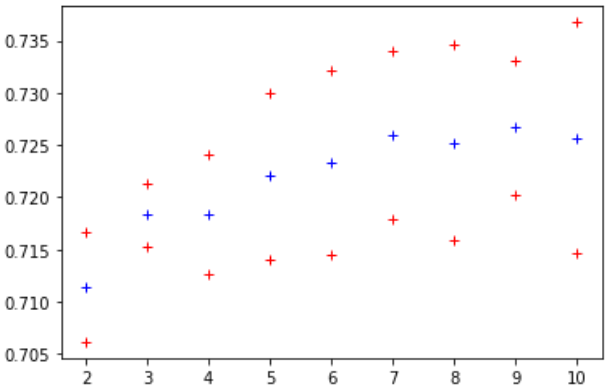


5. Hyperparameter tuning for the chosen algorithms

K-nearest Neighbours

In order to evaluate the hyperparameters, we will tune them using the training set and then choose the estimator with the best performance. After choosing the best estimator, we will then use the chosen hyperparameters to validate the test set. One option is splitting the sample by randomly choosing a subsample of X by ourselves by allocating a part of the data as a validation set and the rest as a test set through randomised indexes. In the following graph, we have the accuracy score for each value of the neighbour parameter:





We can discern that optimal performance, i.e. t he max test accuracy, where the test set in this case is a random partition of the training set dat aset.train['hog'] for a factor of 0.25, the best ch oice of neighbour on terms of accuracy would be n_neighbours = 8. For that number we then proceed with cross validation, a technique that decides on a pre-specified number of partitions for the allotted training set and then proceeds b y iteratively evaluating these partitions – in the end, all partitions will have been used as a test set or been part of the training set. We proceed with the following graph, where we can discern that for cv = 9 we have the highest average cross validation score of 0.726 and the best estimator is `KN eighborsClassifier(n_neighbors=8)`. Alternatively, we can rely on the confusion matrix so lely, notwithstanding cv, which give us a **Train score of 0.79315555 and a Test score 0.511.**

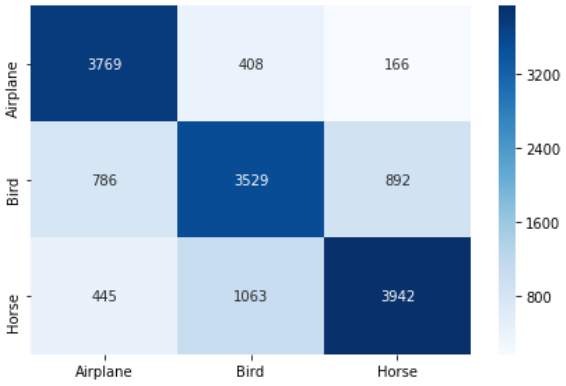
Decision Tree

On terms of pre-processing, it is pretty straightforward since we are using ordinal data. For the decision of the train and validation sets, in this case and from now on we will perform cross-validation with GridSearchCV, which is automatic - as such, instead of randomly choosing a validation set in our explanatory variables, the package allows for also assessing combinations of various hyperparameter values, enabling us to find the one that gives the best results. For the given number of the parameter neighbours and cv level, we can then finetune hyperparameters using GridSearchCV. In the next algorithms we will choose the default number of folds which is 5, but for the time being we can allow a GridSearchCV iteration in a for loop for various folds. The hyperparameters investigated are the number of neighbours, the weights, and the algorithm, according to the code below: When we directly apply the DecisionTreeClassifier() for a pre-determined and replicable randomized state, we naturally get overfitting (Train score 1.0, Test score 0.5). As such, we will focus on pre- and post- pruning for the decision tree. We start with by limiting the parameters max_depth, min_samples_split, min_samples_leaf. In this instance, we will be using the default 5-fold cv technique in the interest of time efficiency for the computations. The parameters tested are:

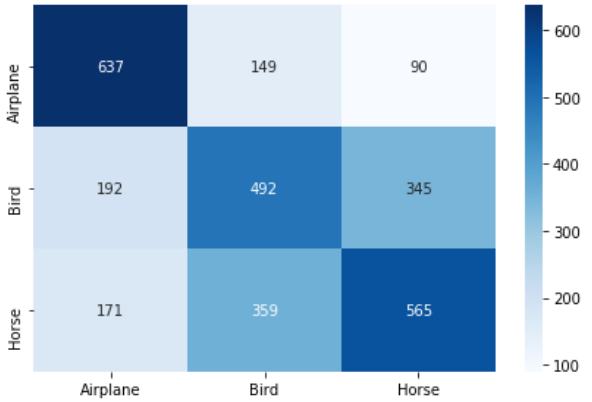
```
params = {'criterion':["gini","entropy"],'max_depth': [2,4,6,8,10,12],
'min_samples_split': [2,3,4], 'min_samples_leaf': [1,2]}
```

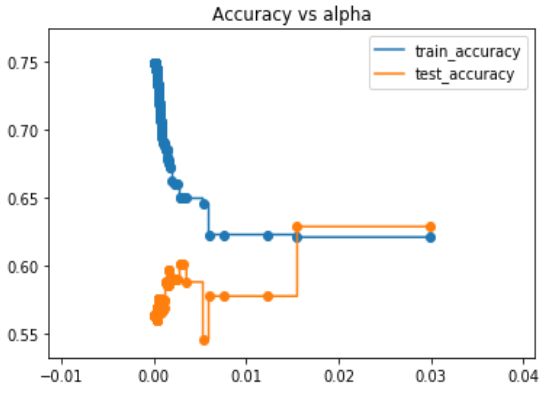
This gives us the following best estimator: `DecisionTreeClassifier(max_depth=8, min_samples_leaf=2)`. We thus have the following classification matrices for the train and test sets:

Train Confusion matrix, Score = 0.7493

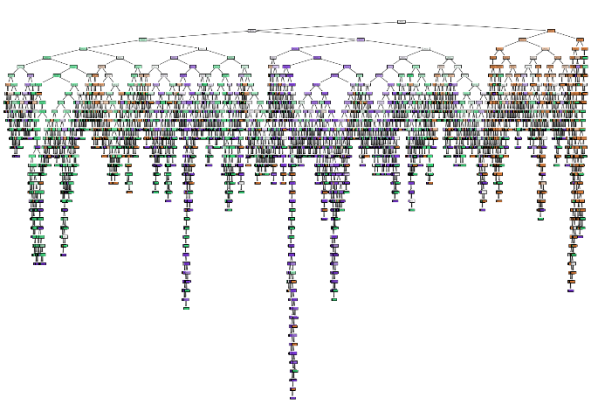


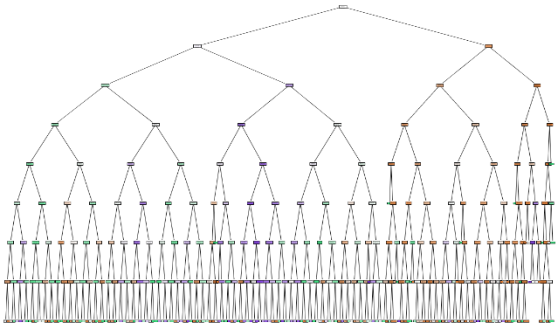
Test Confusion matrix, Score 0.5647





Performance has increased. We can see if we can finally further increase performance by post-pruning, in order to decide for a better alpha through cost complexity pruning in order to decrease entropy. We see that for alpha = 0.02 we have the highest test accuracy. By applying post-pruning, we are finally able to increase the performance of the test set (**Train score 0.621267, Test score 0.629**). The best algorithm is `DecisionTreeClassifier(max_depth=8, min_samples_leaf=2, ccp_alpha=0.02)`. We can finally dwell in our accomplishment by comparing the tree graphs on pruning:





The non-pruned algorithm looks like a mangrove tree, whereas the second looks more like the conceptual abstract the algorithm's graph was supposed to be, with much less branches and levels.

Random Forest

This algorithm due to its randomised ensemble nature requires virtually no pre-processing. As such, we will be evaluating the following parameters:

```
param_grid = {'min_samples_split': [3, 5, 10], 'n_estimators' : [100, 300], 'max_depth': [3, 5, 15, 25], 'max_features': [3, 5, 10, 20]}
```

The best estimator is `RandomForestClassifier(n_jobs=-1, max_depth = 25, max_features = 10, min_samples_split = 3, n_estimators= 300)` and with **Train score = 1.0 and Test score = 0.638**

Neural Networks

On terms of pre-processing, we have already normalised the sample and also the output is categorical, so no label encoding will be necessary. As such, we focus on batch size and number of maximum_iter which after experimenting we were able to reach a balance between use of memory and accuracy for max_iter =3000 and batch_size = 512. As such, the remainder of hyperparameters to be examined is the following:

```
parameters= [{"activation":["identity", "logistic", "tanh", "relu"], "solver": ["lbfgs", "sgd", "adam"], "alpha": [0.001,0.005,0.01], "learning_rate": ['constant','invscaling','adaptive']}]
```

The best estimator is `MLPClassifier(activation='relu', alpha= 0.01, learning_rate = 'constant', solver= 'adam', batch_size=512, max_iter=3000, random_state=1)` and with **Train score = 1 and Test score = 0.5293**

Support Vector Machines

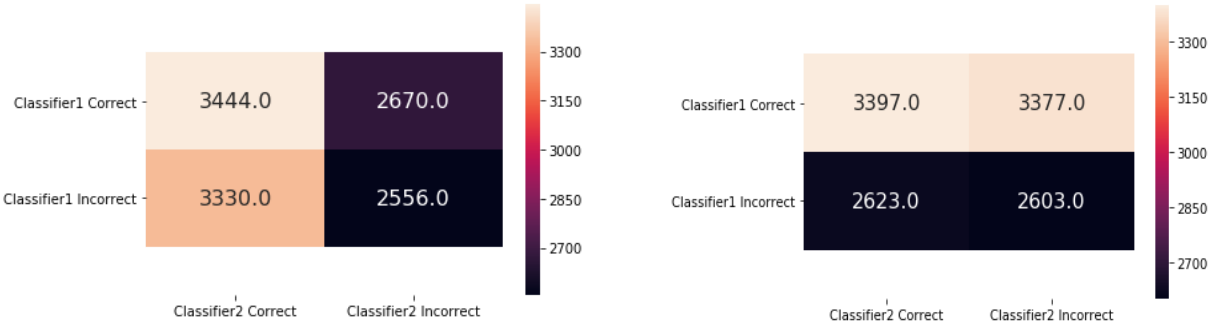
Discretization is not necessary on terms of pre-processing since we have numerical attributes and encoded categorical features, and on top of that, unnecessary discretization through binning may induce a bias that may affect the determination of the support vectors. In addition, order to remove any bias towards outliers we have already standardized the data, as scaling can have a large effect on the accuracy of SVM when defining the support vectors' distance from the decision boundary, we have correctly applied PCA to reduce dimensionality, using the same scaling factors on the training, and test sets. We thus focus on the kernel and choose a big gamma=2 that is large enough to not

constrain the model too much to allow it to capture the complexity of the shape on one hand, but small enough to allow proper regularization with c in order to have the capacity to prevent overfitting on the other. For the parameters evaluated (`[{'C':[0.1, 1], 'kernel':['linear', 'poly', 'rbf', 'sigmoid']}]`), the best estimator is `svm.SVC(C= 0.1, kernel= 'poly')`and with **Train score = 0.684 and Test score = 0.502**

6. General Comparison of Algorithmic performance

In order to find the Algorithm with the best performance we will be using the Mcnemar criterion, which relies on the contingency tables between classifiers that check the balance of disagreements on terms of correspondence between predicted and actual values between two classifiers. We will first rule out classifiers that evidently have lower performance and then compare the rest of the best estimators by algorithms by two where applicable. Evidently, Neural Networks are ruled out due to overfitting, along with the random forest. From the ones remaining, Nearest neighbours and SVM have close test scores so we will perform a Mcnemar test and then compare with the remaining classifier, the Decision tree. By comparing the first two, the McNemar test allows us to reject the Null (statistic=1.442, p-value=0.230, same proportions of errors)

On the other hand, when comparing Neighbours and the Decision Tree, the McNemar gives us statistical significance as with a statistic=72.380, p-value=0.000 we reject the null any and the observed differences on the test set performance is probably real, as we have different proportions of errors. This result shows the superiority of the Decision tree. We find similar results for a comparison between SVM and the Decision tree as well. The contingency tables for the last two cases respectively, are the following:



7. Conclusion

We have evaluated four algorithms following proper pre-processing methods, namely normalisation and PCA for the training and test sets, reducing dimensionality by 30%. This processing was useful for all algorithms and particularly for Random Forests and SVM. Additionally, we performed pruning techniques on Decision Trees and a balanced between batch sizes and number of iterations for NN for higher efficiency and accuracy. We fine-tuned selected hyperparameters on the training sets and evaluated performance on the test set, using cross-validation enabled GridSearchCV with cv = 5. Based on resulting McNemar tests, we proved that the Decision Tree estimator tree.DecisionTreeClassifier(max_depth=8, min_samples_leaf=2, ccp_alpha=0.02) had the best test set performance with 0.629.