MS1 Report

AIGUERPERSE Ambroise (341890),
GIRAUD Louis (334749), ROCHE Louis (345620)

1 Introduction

This report tests and compares the perfomance of three machine learning models on the HASYv2 dataset. The machine learning models used were K-Means, Logistic Regression and SVM. Prior to fitting and testing, the data was normalized using the formula $x_n = \frac{x-\mu}{\sigma^2}$. We also appended a bias term to the data. The validation set used was made with a 80/20 ratio of the data set. So 80% of the data was used for training, and 20% for validation. The method we used consisted in fitting on the training data, then testing on either the validation or the test data. The results are presented below:

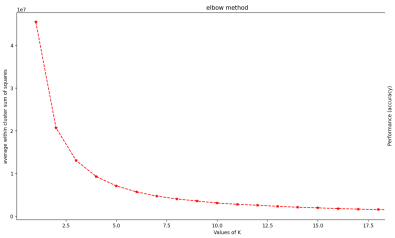
2 Kmeans

For Kmeans, we first tried to get a stable version using the classical Kmeans algorithm, using euclidean distance metric and random center initialization. To find the best number of clusters k, we used two methods: the elbow method and K-folds cross validation with 5 folds. We obtained the respective results in figure 1 and 2.

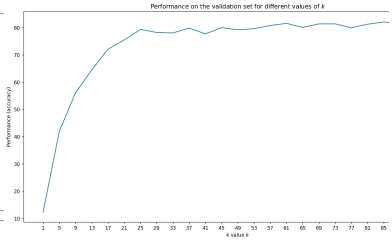
We can see that using the elbow method, we would pick K to be 4-5, which doesn't seem to be the best K, since we can see on figure 2 that bigger values of K result in better accuracy on the validation set, reaching 82.2% for K = 85 (also around 82% on the test set). The fact that values of K, bigger than the actual number of classes, perform better could indicate that some sub classes (sub clusters) could be formed among the classes (clusters).

We then tried to improve the results. First, we tried to change the distance metric from Euclidean to Manhattan. Our second idea was to change the method to initialize centers. Instead of doing it randomly, we tried to implement Kmeans++. The algorithm is described **here**.

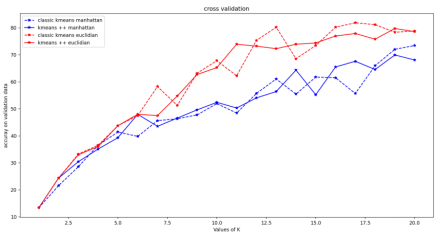
When using Kmeans, it is in the end possible to choose both the distance metric and whether kmeans++ should be used or not, giving four possibilities whose performances with cross validation is shown in figure 3. We can first see that euclidean metric performs better than Manhattan metric, possibly because the data has homogeneous dimensions. Also, Kmeans++ has similar performance as random center initialization. In the end, the classical kmeans algorithm with euclidean distance metric is to be recommended for its simplicity and performance.



Elbow method



K-folds (5 folds)

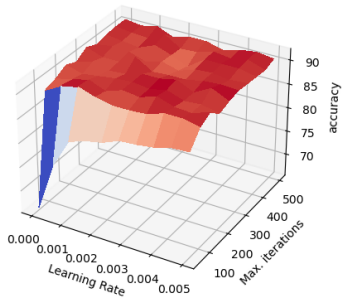


K-Means(++) with the 2 metrics

3 Logistic regression

For logistic regression, we mainly used the homework on the jupyter notebook for our source code. We first tried to use logisitic regression with the given data as is without normalizing it. It did not work. Normalizing the data yielded satisfying results. We appended a bias term to the data. Though we noticed that it does not change much in terms of results. We encountered a few errors such as overflows and undesired function behaviors. That was due to unwieldy function calls. We searched for the best learning rate and the best max iterations count by implementing a method `visualize`, which takes parameters the training and test data and labels, the minimum and maximum learning rate, the minimum and maximum maximum number

of iterations, and the number of steps to go from a minimum parameter to a maximum parameter. One can call it by putting the main argument `--visualize` to true. The function arguments concerning the hyper-parameters have a default value. It fits a logistic regression model using hyper parameters and the training set and predicts the test set. It finds the best hyper-parameters by grid-searching them, prints them, and produces a 3D plot of the accuracy in function of the learning rate and the maximum number of iteration. It is further documented in the code. Running it with the set of default parameters yields $\eta_{best} = 2 \cdot 10^{-3}, iter_{best} = 180$ with 91.8% accuracy along with the following plot.



Accuracy on test data

Discussion: It is important to note that, since the weights are initialized at random, some seeds are to perform better than others. Thus, the result can vary. In practice, these settings get around 90% accuracy, which is along the norms established by the project description.

4 SVM

For the SVM, we will try three different kernels : Linear, Polynomial and RBF using the SciKit library. Each kernel type has different hyperparameters we can adjust : C for Linear, C and γ for RBF and for polynomial, we can choose the degree of the polynomial, C, γ and $coef0$. To find the best possible accuracy on the testing set with all these different methods, we used the validation set.

For the linear kernel, cross-validation yielded maximum accuracy on the validation set with $C = 0.006$ and an accuracy of 94.159%.

For the RBF kernel, we had two parameters to adjust so we did a grid search for a maximum. This method ,although being much slower and less accurate than a 1D maximum search, allowed us to find what seemed to be a global maximum at $\gamma = 0.00075, C = 3$ with an accuracy of 95.327% on the validation set.

For the polynomial kernel, there are four parameters to adjust so it was pretty hard to find the best possible value, but we found pretty good results with $\gamma = 1$ and $C = 45$, $coef0 = 750$. We tried these hyper-parameters with different degree polynomial ranging from 2 to 10 and we observed the best accuracy at degree 2. These parameters allowed us to get a 95.327% accuracy on validation. Although this method isn't really accurate, as we didn't optimize the parameters all at once, once for each different degree this was much quicker and still allowed us to get pretty good accuracy.

5 Conclusion

The obtained results are satisfactory and within the norms established by the project description. The best accuracy on testing set is attained with the SVM method using a RBF kernel, yielding a 95.327% accuracy on the testing set. Furthermore, this method is well-optimized in terms of implementation in the projet. Thereby, it is faster and more convenient to use overall.