Bichara Rania (345616)
El Idrissi Dafali Jad (346195)
Imghi Mamoun (346722)

Report - Milestone 1 of the *Introduction to Machine Learning course*, Spring 2023

Introduction

This report presents the Milestone 1 of the CS-233(b)'s course project done by the students Bichara Rania, El Idrissi Dafali Jad, and Imghi Mamoun. We will here discuss the different algorithms implemented for this Milestone and the different challenges encountered when designing them and testing them on our HASYv2 dataset.

Main function

We updated our main.py function to call the corresponding algorithm to the one given as argument, and we added some data processing before giving it as an argument to our algorithms to train and run. We first normalized our data to uniformize the vectors we have and then added a bias term to it. We then created a validation set to test our data and optimize our hyperparameters, which we did using 30% of our training dataset and assigning it to the test dataset. Finally, we call the corresponding algorithm on our modified dataset.

K-Means algorithm

We first describe the K-Means algorithm and what it does to our data. The goal of our algorithm is to regroup our data into K clusters, and is essentially useful for data which is nicely distributed around a certain mean value, called cluster center.

The idea behind the algorithm is to choose at random k elements of data from our data set, which will be our cluster centers, and iterate over our data to compute the Euclidean distance between our current centers and each element of our data, assign each element to its corresponding center, and then using the mean of each cluster as its new center. We iterate until one of two conditions is reached: we attain our number of maximum iterations or the values of the old cluster centers and the newly computed one is equal.

After designing our algorithm, we tested it on our dataset with the given value of 3, which produced a measly 28.549% of accuracy for our test data, then we tested with k = 10, and each multiple of 10 after until reaching 40, for which we found an accuracy of 85.307%. We then decided to decrement our k number by 5 until reaching the value of 30 clusters where we consistently obtained an accuracy ranging between 79.5% and 83.0% for our test data:

```
python main.py --data dataset_HASYv2/ --method kmeans --K 10
```

Logistic regression algorithm

We begin by describing the logistic regression algorithm and what it does to our data. The goal of our algorithm is to classify the data into C classes using a weight matrix where each column corresponds to the class k.

We first define a softmax function which computes the probability for each element of our data to belong to each class using the given weight matrix learned during training, before computing the loss function. Then, we compute the gradient of the entropy that we will use to update the weight matrix by using the gradient descent method. It is especially efficient for linearly separable data.

After designing our algorithm, we tested it on our dataset with a learning rate of 0.00001 and a maximum number of iterations of 100. We obtained an accuracy for our test data of 75.039%. We then tried a slower learning rate of 0.000001 which resulted in an exponential decrease of our accuracy (20.281%). We then tried with the values 0.0001 and 0.001, the latter yielding an accuracy for our test data ranging between 87.0% and 92.0%.

```
python main.py --data dataset_HASYv2/ --method logistic_regression --lr 1e-3
                              --max_iters 100
```

SVM algorithm

The goal of the Support Vector Machine (SVM) algorithm is to classify the data by creating a line or hyperplane that separates it. It is unfit for large datasets because of its high polynomial complexity.

We here call the sklearn.svm module and use the SVC class of this library, which can be translated to Support Vector Classification. We support multiple types of problems, namely linear and non-linear problems, with multiple kinds of inputs (for example, our kernel function supports up to three parameters). We then fit and predict our model using this same library.

After designing our algorithm, we tested it on our dataset with a ‘rbf’ kernel, a gamma of 0.01, and a penalty term of 1, which resulted in an accuracy ranging between 17.0% and 22.0% for our test data. We then tried a lower gamma value of 0.001 for which we obtained an accuracy ranging between 91% and 95%. Trying lower gamma values result in a slightly lower accuracy for our test data (between 88.0% and 92.0%). We therefore decided to go with a gamma value of 0.001. We then tried a lower value for our penalty term which resulted in a lower accuracy of ~85%, and values higher than 1 which had no influence whatsoever on our test data accuracy:

```
python main.py --data dataset_HASYv2/ --method svm --svm_c 1.
          --svm_kernel rbf --svm_gamma 0.001
```

We tested it then on our dataset with a ‘poly’ kernel using different values of degree and coef0. We settled on the value 2 for degree and 1 for coef0 which gave a yield of ~94%.

```
    python main.py --data dataset_HASYv2/ --method svm --svm_c 1.
--svm_kernel poly --svm_gamma 0.001 --svm_degree 2 --svm_coef0 1
```

Finally, we tested the ‘linear’ kernel which gave a yield of ~93%.

```
python main.py --data dataset_HASYv2/ --method svm --svm_c 1.
                --svm_kernel linear
```

Conclusion

This project allowed us to test various algorithms which all serve the purpose of classifying a given dataset, each of them presenting a different way of classification and we noticed that some of them were more efficient than others (the ‘rbf’ implementation of SVM for example).