2nd mandatory assignement : different methods for classification

Thibaut ELOY

How to run my code :

There are 5 files :

- *loadData* : contains the definition class of a data point, the functions that creates the 3 sets and plot them.
- *SimpleClassifier* : logistic Regression, linear Regression, perceptron, one vs rest
- *NeuralNetWork*
- *kNN*
- *tests*

Open the file test, there are several functions which run the different algorithms and print the results. At the end of this file there are several commented lines that call this functions.

For simple classifiers and neural network the loss and the succeed percentage (computed on validation set) are printed at each iteration so that we see the evolution. If you do not want it to be printed, set the variable *printLossWhileTraining* to False. There is one such parameter at the top of file *simpleClassifier* and at the top of file *neuralNetwork*.

1 Part 1: Comparing classifiers

1.1 Data sets

I chose a different data representation than what was suggested. Each data is represented by an object of the class *labelled data* which has 3 attributes :

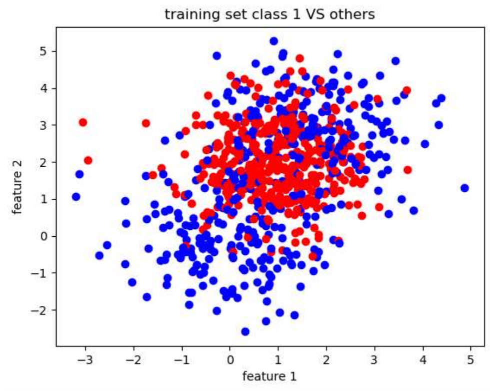
- *Vector* : a vector which contains the features representing the data point. In our case it is a 2 dimensional point
- *Label* : an integer which tells the class the data point belongs to. Each class is represended by an integer between 0 and n-1 with n the number of classes.
- *LabelVector* : a n-dimensional vector such that *LabelVector*[Label] = 1 and *LabelVector*[i] = 0 with i != label.

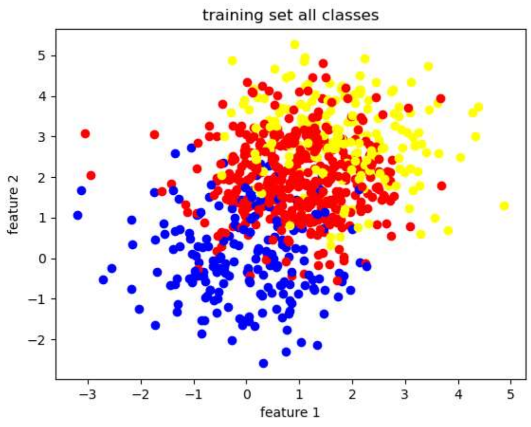
The function getValue(targetClass) will return 1 iff the data point belongs to class targetClass by using *LabelVector*. If targetClass = « » is given it will return the ID of the class. (in our case 0, 1 or 2 ie blue, red or yellow)

There are several advantages of using this representation instead of what was suggested :

- First to implement kNN algorithm I needed to sort the data so if the vectors are separated from their labels it implies that when we sort the list of vectors we lose the correspondance to their labels because the list of labels is not sorted at the same time. Thus, I found more relevant to use the same representation for all the algorithms even if for others we do not need to sort the data.
- Secondly, for each class, the labelVector gives the information wether it belongs to it or not. Therefore we have all the information that we need to train the different neural networks for *one versus rest* algorithm, with only one training set.

Bellow you can see the two training sets :





1.2 <u>Binary Classifiers</u>

a) Linear Regression, Logistic Regression, Perceptron

Since this three classifier are very similar I used the same class *simpleClassifier* for representing them. An object of this class has the following attributes :

- *type :* can be « logReg » or « linReg » or « perceptron »
- *targetClass :* the output of the classifier is 1 iff the input belongs to *targetClass* (0 otherwise). In our case *targetClass* = 1 = red. This parameter will be useful for OneVsRest algorithm.
- *nbFeatures :* the dimension of inputs data (2 in our case)
- *weights :* a *nbFeatures*-dimensional vector which represents the connections between the input vector and the output node.
- *Biais*

Training attributes :

- *etha :* the learning rate
- *batchSize* : the number of data that we put in the classifier between each update.
- *diff* : a parameter that tells us when to stop training. Between each update we compute the loss and if the difference between the loss before and after update is less than *diff* we stop training.
- *nbMaxIteration :* guarantees the end of the training.

Loss function :

For linear regression I used MSE and for logistic resgression cross-entropy loss function as suggested in the book. Perceptron algorithm has no loss function since the output node is not a continious variable.

Ouput :

To get the output of the classifier given the input we compute the dot product between the input vector and the weights vector, we add the biais and put the result into an actictivation function which is :

- a sigmoid for logistic regression
- identity function for linear regression
- heaviside function for perceptron

modification of weights :

the three classifier update the weights in the same way given the loss functions mentionned above, we do not need to make a distinction.

Training :

At each iteration we fed *batchSize* data to the classifier, then we update the weights according to the differences between the outputs and what is expected. For linear and logistic regression we stop when the improvement of the loss function is not more than diff. However for percepron there is no loss function so we do a given number of iteration and store the solution that has given the best succeed percentage during the training.

b) K-nearsets-neighboor algorithm (both for binary and multiclass)

The arguments of the algorithm which classifies one vector are :

- *trainingSet*
- *targetVector :* the vector we want to classify
- *k :* the *k* nearest neighboors are looked to do the vote for classifying *targetVector*
- *targetClass*. If *targetClass = « »* then all classes are considered. If *targetClass* is a class ID (0,1 or 2) it means that we consider *targetClass* vs others.
- *listClass.* The list of all possible classes. [0,1,2] if we consider all classes [0,1] otherwise.

It returns the class predicted by the classifier.

There are three steps involved. First the training set is sorted according to the distance of its vectors to *targetVector*. I used the *sort* function of python with the key equals to the funcion which evaluates the distance between the target vector and one particular vector of the training set.

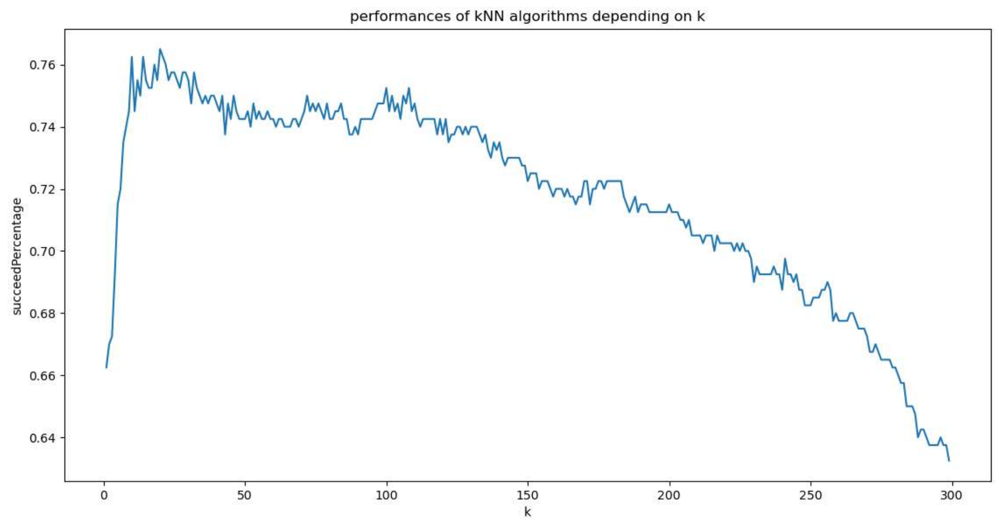
Secondly we count the number of points belonging to each class. To do that I first initialize a dictionnary. The keys are the classes and the values the number of points belonging to it. Then I Iterate on the k first values of the sorted training set and update the values of the dictionnary. Finally I return the class which has the highest number of vote.

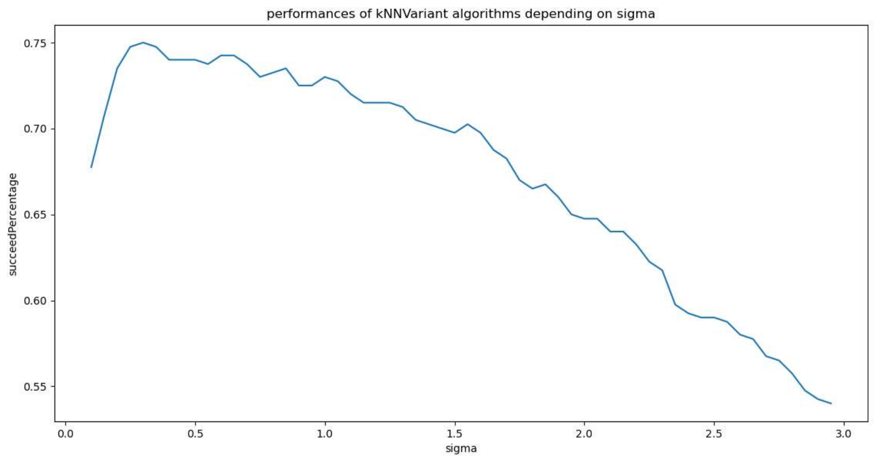
In addition to the classic kNN algorithm I also coded a variant of it which is based on soft vote. Instead of looking at the k nearest neighboors and take the majority class, it looks at all the vectors and give a weight according to the distance to the target vector. I chose a gaussian expression for this weight such that

$$w(d) = e^{-(\frac{d}{\sigma})^2}$$

With σ a parameter to adjust, as k for classic kNN. Then we take the class that has the highest sum of weights.

I found that the best parameters for kNN and soft kNN are k=20 and sigma =0.3 by computing the accuracy on validation set for various values of k and sigma. You can see bellow the variation of the accuracy of these two algorithms depending on k and sigma.





c) Summary

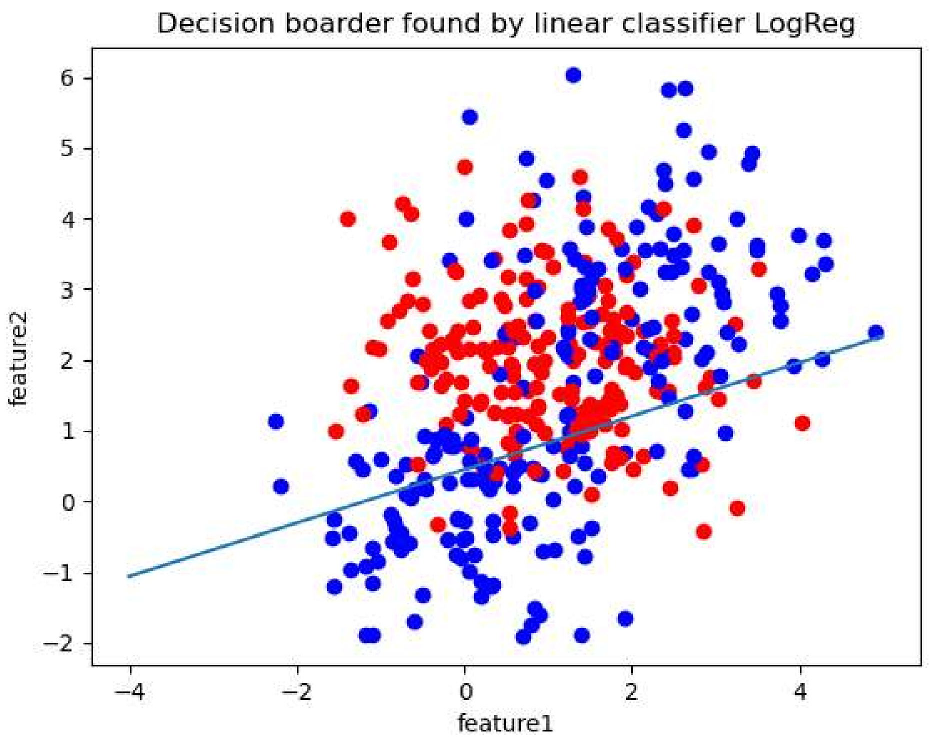
Here are the results obtained on the validation set for these four algotithms :

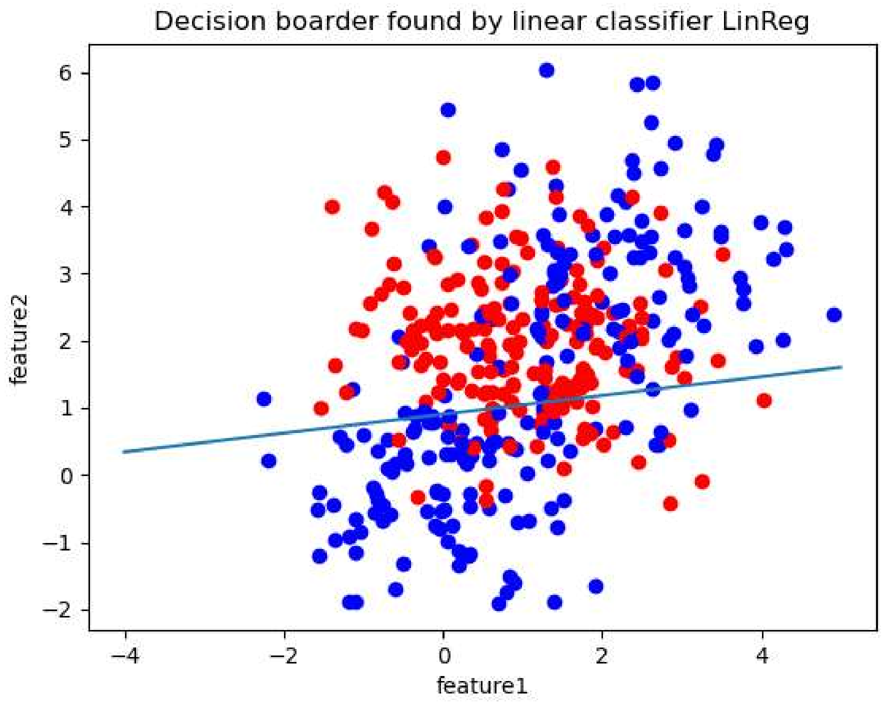
Algorithm	Linear regression	Logistic regression	perceptron	kNN	kNN soft
accuracy	0.61	0.615	0.61	0.7575	0.765

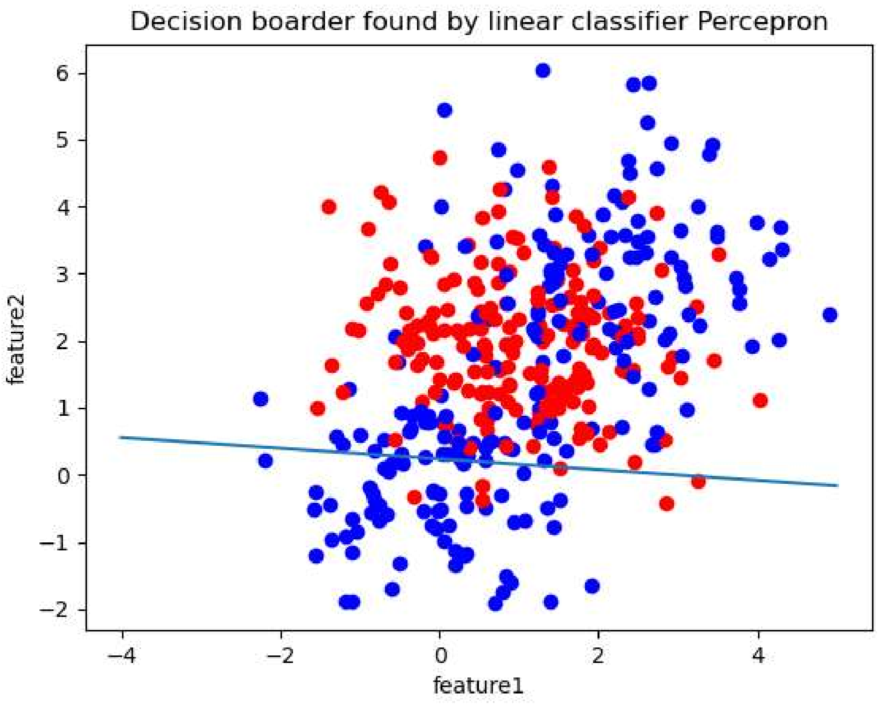
Perceptron, linear regression and logistic regression give similar results on the validation set with an accuracy around 0.61. This is a quite bad accuracy as a random classification would give 0.5 accuracy. But it is not suprising because the goal of this 3 algorithms is to find the best straight line that separate the two classes. However the 2 classes are far from being lineary separables. In fact the points of class 0 are « in the middle » of points of class 1. That's why this 3 classifier are not appropriated.

The kNN algorithm gives an accuracy of 0.76 which is better but there are still a high error rate. But this is not surprising because some points that are very close to each other can have different classes, especially in the middle.

I plot the decisions boarders found by the first three algorithms :


Decision boarder found by linear classifier LogReg
feature1
feature2


Decision boarder found by linear classifier LinReg
feature1
feature2



1.3 Multi-class classifiers

1.3.1. kNN : it is the same algorithm as for binary classification

1.3.2 One versus Rest :

We initialize 3 different classifier, the only difference between them is the *targetClass* which is 0,1 and 2 for the first second and third. Then we train all of them. The prededicted class of a data is the class associated to the classifier which gives the highest output. We predict a class for each data of the test set and compare it to the real class, we compute like that the succeed percentage.

1.3.3 Summary :

Algorithm	One vs Rest (logReg)	kNN	kNN soft
Accuracy (validation set)	0.695	0.7575	0.765

On multy class classification kNN also gives better results than one versus rest logistic regression however the difference is less significant. (0.75 vs 0.70).

One versus rest algorithm find 3 straight lines. It will succeed to find good separtion for class 0 vs other (0.80 accuracy) and class 2 vs others (0.85 accuracy) but not for class 1 vs others (0.60 accuracy). The global accuracy is in between.

kNN algorithm gives exactly the same accuracy as for binary classification.In fact it never classifies class 0 as 2 and class 2 as 0 because they are far from each other. It is shown by the confusion matrix in part 3. So the mistakes that the algorithm makes are the same as for binary classification.

1.4. Adding non-linear features :

Algorithm	linReg	logReg	pergceptron	kNN	kNN soft
normal	0.61	0.615	0.61	0.7575	0.765
Non-linear features	0.71	0.735	0.6825	0.7475	0.71

Adding non-linear features gives significantly better results for perceptron, linear and logistic regression . This can be explained by the fact that points of class 1 (in blue) tend to have higher absolute values both for feature 1 and feature2. Then the values of x1^2 and x2^2 are higher for class 1 than for class 0. Then this attributes are closer to be lineary separable than x1 and x2.

However for kNN it doesnt give better results. In fact kNN is based on neighboorhood and if we add other features it can change significantly the distance between two points. 2 points that are closed could then become far away.

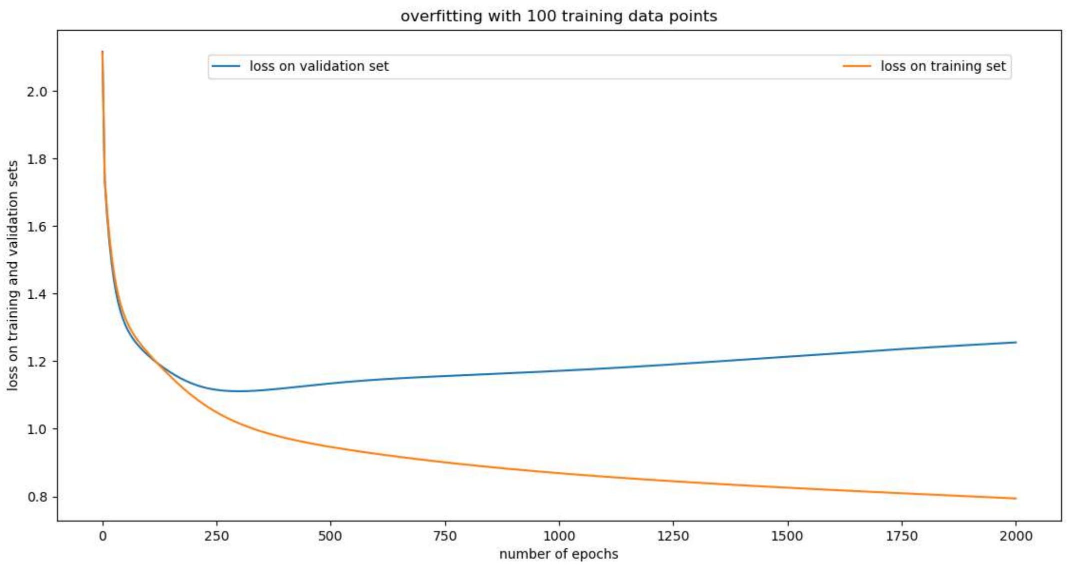
2 Part 2: Neural Networks

caracterised by
- An integer number of layers 'n'
- An integer's list 'lengths' such that lengths[i] is the number of neurons from layer i.
- A list of matrices 'weights' (n-1 matrices). Each matrix contains the weights of all connexions between 2 consecutive layers
The size of weights[i] is lengths[i]*lengths[i+1].

- A list of layers 'layers'. Each layer is an array of size lenghts[i]. If an input batch is given it will have a size of batchSize*lenghts[i]
- A learning rate 'etha' which is a float number.
- other training parameters e, t, nbMaxEpochs, batchSize
- meanSquaredLossFunction is a boolean which is true iff we want to use meanSquared error while training otherwise cross entropy is used.

The accuracy found by the neural network with one hidden with 10 neurones in the hidden layer on the validation is 0.765. I tried different values of number of neurones and I also tried to add more than one hidden layer but it didn't improve the accuracy. Note that it is a similar accuracy that the one found by kNN.

To illustrate overfitting I plotted the graph bellow :

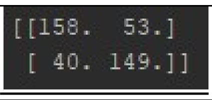
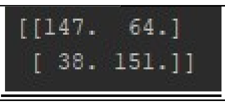


We can see that the loss on training set always decreases but on validation set it starts to increase after 200 epochs, so we need to stop training after 200 epochs. Here the neural network was trained on only 100 data points, in fact with 800 data points there is overfitting but it is so negligeable that it is not visible on a graph.

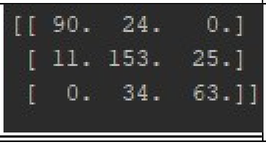
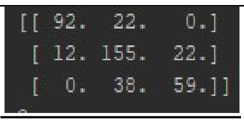
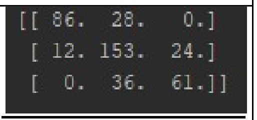
3 Part 3: Final testing

The best results for binary classification on validation set are found by kNN and soft kNN and for all-classes classification it is kNN and neural networks.

a) Binary classification results on test set

Algorithm	kNN	Soft kNN
Accuracy validation set	0.7525	0.765
Accuracy test set	0.7675	0.745
precision	0.749	0.697
recall	0.798	0.794
Confusion matrix	 `[[158.  53.]` `[ 40. 149.]]`	 `[[147.  64.]` `[ 38. 151.]]`

b) <u>All-classes Classification</u>

Algorithm	Neural network	kNN	Soft kNN
Accuracy validation set	0.765	0.7575	0.765
Accuracy test set	0.765	0.765	0.75
Confusion matrix	 `[[ 90.  24.   0.] [ 11. 153.  25.] [  0.  34.  63.]]`	 `[[ 92.  22.   0.] [ 12. 155.  22.] [  0.  38.  59.]]`	 `[[ 86.  28.   0.] [ 12. 153.  24.] [  0.  36.  61.]]`

The results on training set and validation set are slightly different but the difference is not significant. We can compute the uncertainty for example for binary kNN :

Δa =1.96* $\sqrt{a * (1-a)/n}$ = $1.96\sqrt{0.76 * (1-0.76)/400}$ = 0.04. A 95% confidence intervall is the

[0.76-0.04, 0.76+0.04] = [0.72, 0.80]. Both values are in the intervall.

We can notice that kNN and neural networks give the same accuracy and it might be not far from the best accuracy t we can get. In fact it is very unlikely that any classifer would be able to find 100% accuracy for this probem. It is due to the way the data points are generated. There is a deterministic part (the centers of the class) and a probabilistic part (the distance and angle to the center). The clasifiers are only trying to guess the deterministic part. So if there is too much randomness it is impossible to classify all the points correctly no matter how good our classifier is.