Supervised Learning Report

I6222534 Luka Bowen
I6219496 Alexander Leonidas

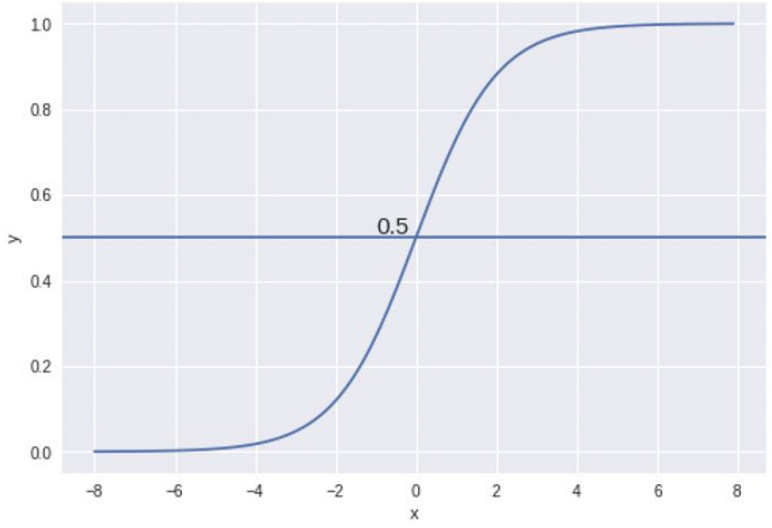
Naive-Bayesian classifiers

First we found a discrete dataset that we used for this mini project from the UCI machine learning repository (Breast Cancer Data Set). The task that Alexander and Luka worked on was to perform a naive bayesian classifier to this dataset. We wrote the code in octave and used some code from github in order to replace matlab functions which are not yet implemented in octave. In the *main.m* file we load the dataset into an array using octave's *dlmread()* function. We split the data into features and labels and removed the first row from the dataset, which were patient ID's. We created a function, *Naive_Bayesian(),* which takes an input of features and labels. The parameters were split into testing and training sets using an 80-20 split. After this, we got the labels using octave's *unique()* function on the labels array from the data, where the length of this variable is the number of classes present in our data. Then the class probabilities were calculated, which is the number or elements of a specific class divided by the total number elements. Then we calculated the mean and the standard deviation for the elements of a specific class using the built in octave functions, *mean(), std()*. Then we used octave's *normcdf()* function to calculate the normal cumulative distribution for the testing data using the mean and standard deviation from the training data. We had to create 2 cumulative normal distributions in the range [x+alpha, x-alpha], where alpha = 0.5, and take the absolute difference between them in order to make sure that the computed probability belongs to the range of testing data +-alpha, instead of computing the probability that belongs to the range [-inf, x]. Then we got the index of the maximum value of each row in the probability array which tells us which label the test data more likely belongs to. The function outputs a matrix with two columns, where the first column is the predicted label, and the second column is the actual label. We then analysed this process to see how effective it was. We made a confusion matrix, using code from github which mimics the *confusionmat()* matlab function which is not implemented in octave. The confusion matrix outputs a 2x2 matrix, the element (1,1) being true positives, element(1,2) being false negatives, element (2,1) being false positives and element (2,2) being the true negatives. We found that our algorithm has a precision of 98%, where precision is true positives divided by the sum of true positives and false positives, and a recall of 1, where recall is true positives divided by the sum of true positives and false negatives. A recall value of 1 means that our algorithm did not assign any false negatives to the testing data. We also computed the F1 score, which represents the balance between precision and recall, which we found to be 99%, which is to be expected from the precision and recall. Also computed was the mean squared error, which was very small(0.02) due the high accuracy(98%) of our algorithm. The script *main.m* outputs a plot of the confusion matrix using the *plotconfmat()* function, also taken from github to implement an existing matlab function into octave. We have a few suggested improvements that could be implemented, we linearly chose the training data(1:first 80%) and testing data(the last 20%), it would probably be better to randomly select 80% of data points and use the remaining 20% as testing data. We would have to keep track of the indexes to make sure no data overlaps or is duplicated.

Alexander De Pauw i6210128
Eva Ruzicka i6214718

Logistic Regression

In nature a classification problem is called a problem when independent variables are going to be continual. In this case the dependent variable has to be in categorical state. As a negative or as a positive output.
One of the most common Machine Learning classification algorithms is Logistic Regression. This algorithm is used to predict the probability of a variable that is categorical dependent. Logical regression uses binary variables.
Oppositely, from the linear regression, where output is the weighted sum of inputs, in logistic regression machine does not output the weighted sum of inputs directly, but it has been passed through a special function. That function is able to map all real values between 0 and 1.
In case if the result of binary code is 1, the output is “yes”, “success”,” true”, etc. However, if the result is 0, the output will be “no”, “failure” “false”.
The special function that is used is known as the sigmoid function. This function is represented as an “S” shaped curve. The plot of the sigmoid function looks like:



(Swaminathan, 2019)

Values of sigmoid function always lines between 0 and 1. If x is an extremely negative value it will be collated to y that is very close to 0. Whereas, values close to 1 are outputted as positive values.

The hypothesis for logistic regression then becomes

$$h(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$h(x) = \begin{cases} > 0.5, & \text{if } \theta^T x > 0 \\ < 0.5, & \text{if } \theta^T x < 0 \end{cases}$$

,

If the weighted sum of inputs is greater than zero, the predicted class is 1 and other way around.
Dataset that will be used is based on the idea that we are given scores of the final exam. The goal of our classifier is to separate and allocate students into two categories. Categories will be based on applicant's scores. One group will be called Class (1), applicants that can be admitted to the university are placed into this group. However, if students are put into Class (0), admission cannot take place. The dependent value takes on binary values 1,0. Value 1 states that applicant is accepted to the university, while 0 stands for negative outcome.

We now decided to take 70% of our data as training data and leave the rest as test data. We initialize theta for our costFunction. The costFunction takes as input the initialized theta, the training data and the training labels. We used cost function in order to evaluate our design requirements while using our modeled variable values. Furthermore the Sigmoid function is used in the costFunction.
Sigmoid function is represented by sigmoid formula, which will further be correlated to our final graph. It is showing us the connection between our data set and helping us to separate the given dataset. More precisely, it is helping us to visualize whether applicants will be accepted to the university or not.

Now we have to minimize the cost function to obtain the optimal theta. This will return theta and its cost.

We use predict to compute accuracy on our training set. The predict function takes the training data and the optimal theta as input and returns the probability of the data being positive. More close our probability gets to 1, our predicted model is more likely to be allocated in class 1.

Finally we apply the sigmoid function on our data_test to see if our algorithm works. The labels_test are there to compare with the result given by the machine.

Intuition on how to continue

Furthermore, to make a step forward our solution it would be necessary to implement Sigmoid function to our readdata. That would lead us to the graph where the Sigmoid curve would be represented as a pass(1) or fail(0).
Then we would need to take an initial guess for theta, use that to find the optimal theta by minimizing the cost function. At this point we could use the optimal theta to predict the labels of the data_test.

Interpretation of the results

Compare labels given by algorithms with test_labels, this could give us an indication on how good our algorithm actually is.

2nd attempt: Because our code was not working we tried to redo the algorithm using gradient descent as an optimisation tool. The problem we ran into in that code was the wiring of all the different functions.Separately, our functions had different roles and that was

supposed to lead us to our main code. However we could not find the way of connecting all of them and put them in one function.
Our second attempt of running this code consists of: “classif”, “CostFunc”, “decisionbound”, “GradDescent”, “Predict”, “Sigmoid” and “training”.
‘Training” has an input of feature, leabl, weights, step size and number of iterations. It is responsible for giving
Weights are going to be values that are supposed to control the strength of the connection between input and output. The value of how much influence would input points have on output points. The closest a data point is to the decision boundary the more weight that point has because by moving it slightly the whole boundary could shift.

I6222534 Luka Bowen
I6219496 Alexander Leonida

We continued where Alex and Eva left off, we tried to implement their logistic regress functions into a main, and train a model based on their functions, which we ran into trouble completing due to time constraints and some errors in the algorithms. We realized that they did not normalize their data, which we implemented in order to allow the sigmoid function to produce a proper output. This allowed us to use gradient descent to calculate optimal weights for each iteration, though something again seems to be wrong with the algorithm as the weights do not change per iteration, and our gradient descent algorithm appears to be maximizing the cost function as opposed to minimizing it. We made a second attempt to perform a logistic regression classifier which outputs a decision boundary graph for the data, however we were unable to plot the decision boundary function alongside the predicted outputs due to octave’s symbolic package not correctly detecting my python installation, which is required to produce the decision boundary function. We used a source (https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#cost-function) which helped us to understand and implement the logistic regression technique.