Assignment 1

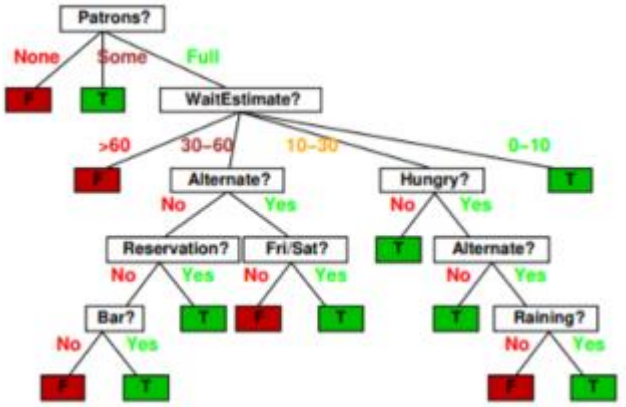
Theory

1. Concept Learning is the art of finding the attributes, or combinations of attributes and their impact on a problem domain. The course book defines it as the art of searching through all the hypothesis in a problem space in order to find the best fitting hypothesis. This can be done though several machine learning algorithms, all of them by using training data with known expected values.[i]

   Example: Will I wait for a table at the restaurant?

Example	Attributes										Target
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	T	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

The table above shows an instance at every row with the corresponding attribute values in the columns. The target value is indicating whether a given person would wait for a table at the restaurant. From this training data one can learn which parameters is important to the given person when evaluating if he/she should wait for a table or perhaps search for another restaurant. Using this dataset, one could for instance construct a decision tree like the one below.

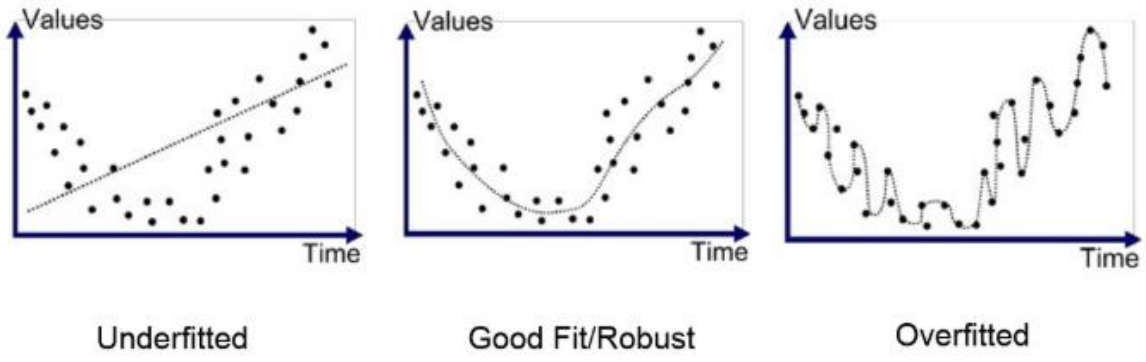


PS: The example, data table and the decision tree is copied from Helge Langseths lecture slides from the course TDT4171.

2. In mathematics, function approximation is often called regression. One has data from an unknown function and wants to find a function that approximates the given datapoints well enough for a given task. Machine learning is very much like this, because when machine learning systems learn from big data sets we don't expect them to be correct 100% the of the time (most of the time), but we want them to be right most of the time (within our specified threshold for the given task). In short, the function approximation is the process of training

the machine learning system to be able to approximate the real function that is the foundation of the problem in the real world.

3. During supervised learning, the machine learning system is trained on a set of data with corresponding expected values. How well will such a system perform on data it has never seen before? In order to increase the performance for unseen data, one can apply *inductive biases*, prior knowledge that is taken for granted, even before training begins.
   The inductive bias in the **Candidate-Elimination algorithm is the assumption that the target function is part of the set of all hypothesis** (that among all the hypothesis constructed in the beginning, at least one of them is fitting the target function perfectly).
   For the **decision trees**, usually they are programmed to choose **smaller trees over deep ones**, so this is their inductive bias. It is usually approximated by calculating entropy and information gain when deciding which attribute to split on next.
4. Overfitting is the concept of training a system on a finite training set so much that it starts to fit the training set more than grasping the general characteristics of the problem domain. One can think of the analogy if memorizing vs understanding. When one trains a system on a finite training dataset it is custom to split the data into at least 2 parts. One of the parts, usually the bigger one is used to actually train the system, the second part is used to validate the systems performance over time during training. This second data set is called the *validation set***.** It is usually a given that as long as one trains a system long enough on the same data set, its performance on that same data set will improve over time, but whenever the performance on the validation set starts to decrease it is a good idea to stop the training further in order to avoid overfitting. **Cross validation** is about mixing up which parts of the available data sets is used for training and which is used for validation. Usually one wants to train the system on all data available, so taking out a part of the available data only to use it for validation can lead to the system not being exposed to important features. When changing which part of the data set one uses for validation and training, usually also shuffling the data, hopefully the system gets exposed to all available features in the domain, but can still use the validation-technique to perform early stopping etc.



5. The initial version space is any combination of all the parameters and their legal values. The idea is to squeeze the hypothesis between the most specific and the most general hypothesis one can guarantee for. Initially, the most general hypothesis one can guarantee is that everything is allowed, $\mathbf{G_0 = \{<?, ?, ?, ?>\}}$. the most special hypothesis makes no sense initially, no values allowed is initially set; $\mathbf{S_0 = <\emptyset, \emptyset, \emptyset, \emptyset>}$
   i. After observing <Female, Back, Medium, Medium> : True, the General Hypothesis is kept equal to G_0, but $\mathbf{S_1 = <Female, Back, Medium, Medium>}$.

ii. When observing <Female, Neck, Medium, High> : True, the specific hypothesis must be updated to allow for this record to be classified as true as well; **S_2 = {<Female, {Back, Neck}, Medium, {High, Medium}>}**
iii. In record 3 the first negative result comes, this leads to us having to update the general hypothesis to exclude this record from being classified as true; **G_3 = {<Male, ?, ?, ?>, <?, {Back, Neck}, ?, ?>, <?, ?, {High, Medium}, ?>, <?, ?, ?, {High, Medium}>}** All the examples observed till now fits in between these two hypothesis boundaries.
iv. Another positive label and we need to update the special hypothesis accordingly; **S4 = {?, {Back, Neck}, {High, Medium}, {High, Medium}}.** The general hypothesis also needs some correction because now we have that the special hypothesis opens up for generalization between genders that the general hypothesis does not ; **G4 = {<?, {Back, Neck}, ?, ?>, <?, ?, {High, Medium}, ?>, <?, ?, ?, {High, Medium}>}.**
v. Again, with this positive target, the special hypothesis needs updating; **S5 = {?, {Back, Neck}, {High, Medium}, ?}.** The general hypothesis that has (High, Medium) as its most special attribute can now be eliminated because the special hypothesis is more general in this case; **G5 = {<?, {Back, Neck}, ?, ?>, <?, ?, {High, Medium}, ?>}.**

Now we know that, if the training data is consistent with the real domain, the real hypothesis lies between the boundaries of the special hypothesis and the general ones.

Programming

All the code is delivered in attached zip-file. Numpy and MatPlotLib are the two external libraries that are used. Numpy is used for vector maths, which makes the code much more concise and efficient. Matplotlib is used for the plotting.

Linear Regression

1. For task 1 I simply used numpy's .dot() and .transpose() functions to implement the equation given in the assignment paper

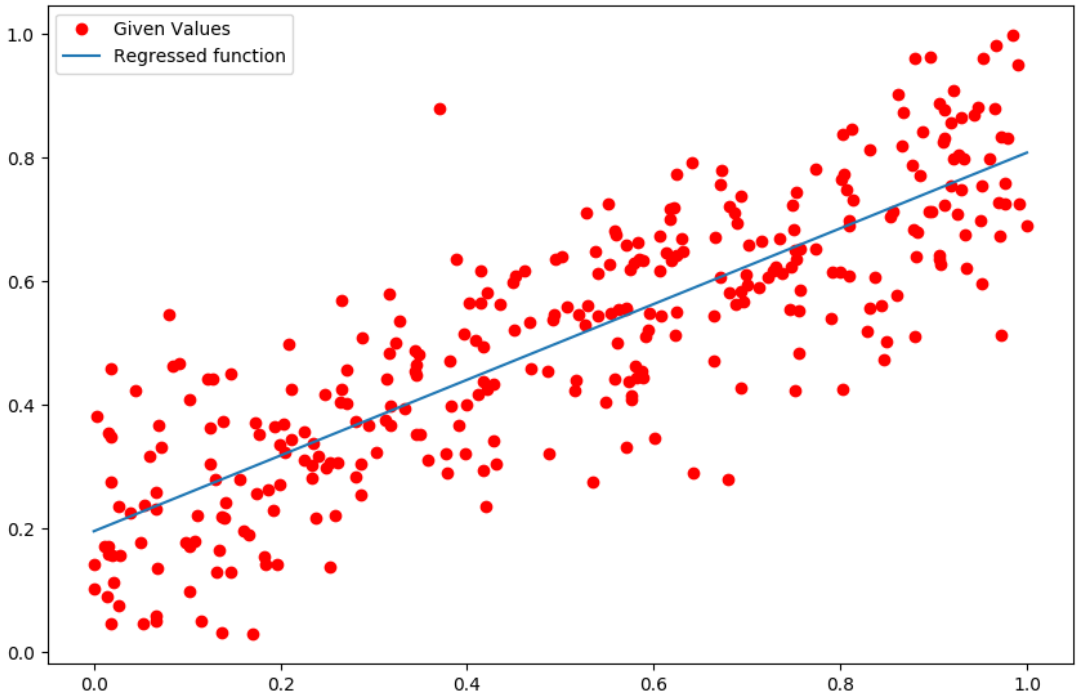
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

2. Used the function created in task 1 and implemented the mean squared error function given in the assignment paper.

$$= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2(\mathbf{X}\mathbf{w})^T\mathbf{y} + \mathbf{y}^T\mathbf{y}$$

The results say that the error was in fact smaller on average when applied on the test data than when applied on the training data. Though I cant see the exact reason why, I still believe it points in the direction that it does not overfit, but in fact generalizes OK. The algorithm produced the weights [0.24079271 0.48155686 0.0586439], where the first element is the bias-element ant the 2 others are x1 and x2 respectively. The mean squared error for the training set as a whole resulted in 3.3757264877252027 and the mean squared error for the test set resulted in 1.667708778858298.

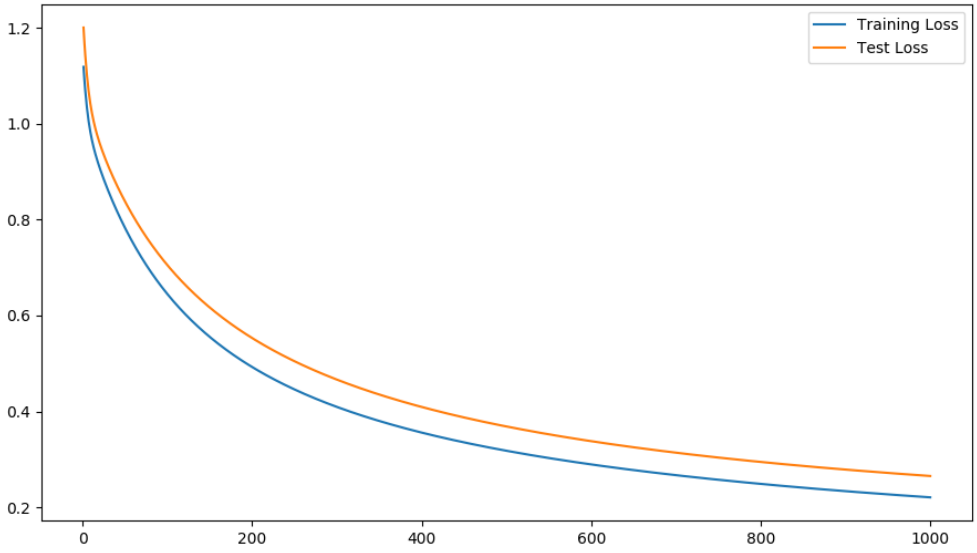
3. Again, the test dataset gave a smaller average error than the training data. Below the regressed function is plotted alongside the given values in the *training set*. Without a hardcore overfitted function with way more weights, it seems hard to make this function approximate the given values better than it already does. So in conclusion, the least squared regression works perfectly within the limitations.



To elaborate a bit further on what was mentioned above; One can make a function that hits every single one of the given values, but this is of course not possible for a function with 2 weights. One could create a more complex function with more weights, but the problem with this is that I might just work for the training data and won't generalize better than the basic function shown above (overfitting).

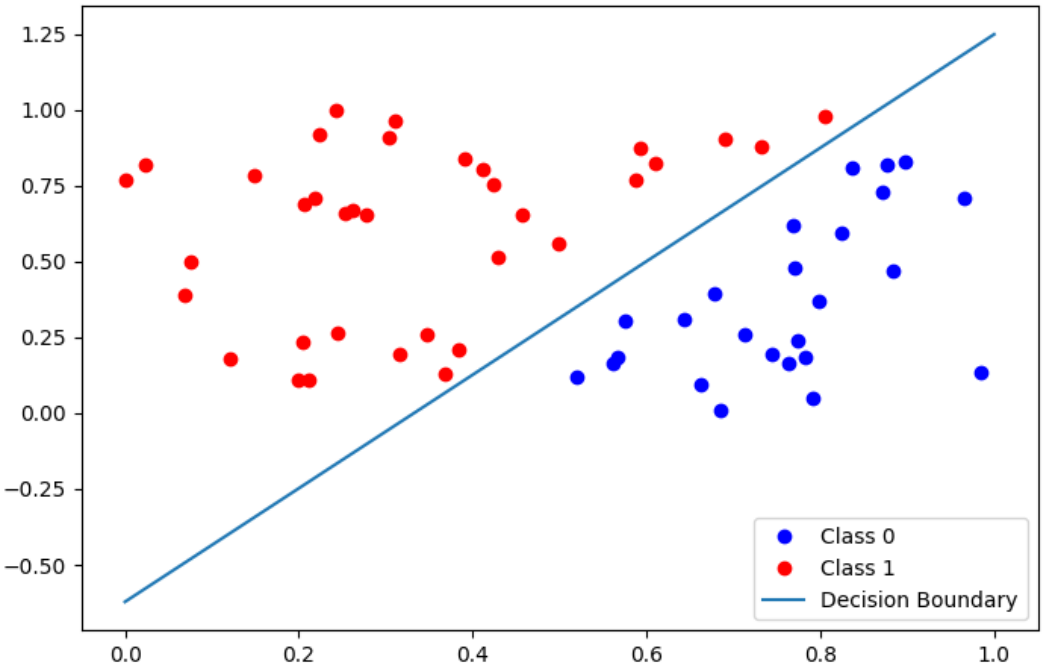
Logistic Regression

1. After 1000 epochs of training on the training set, the model seems to classify the training data and the test data perfectly. The average cross-entropy error of the last iteration was ca 0.2215 for the training data and 0.2660 for the test data. This example was calculated when the learning rate was set to 0.005. The weights including the bias is set random between 0 and 1, which is usual when initializing neural networks. The small difference between the error for the training data and the test data leads me to believe that the model manages to generalize well.

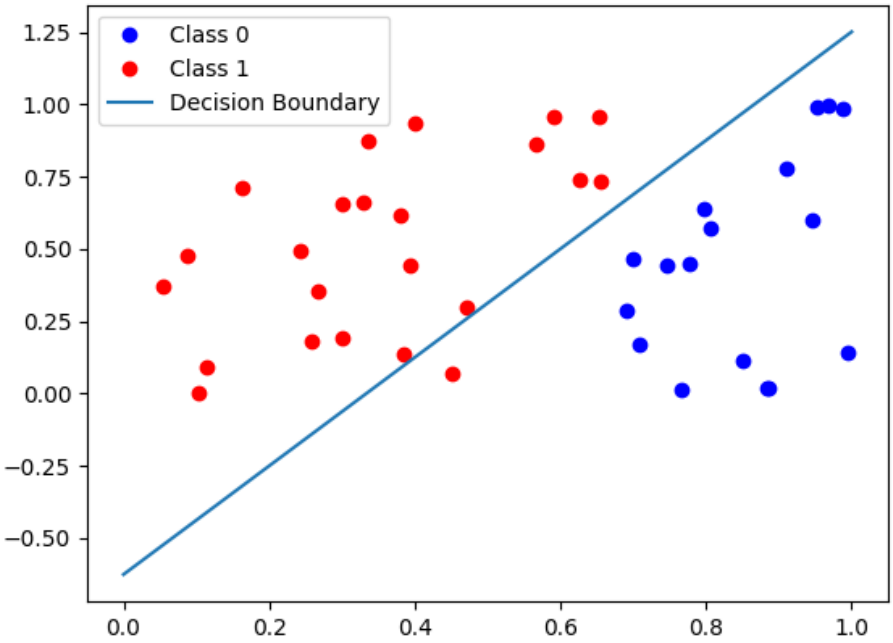


The graph above shows the error rate over 1000 epochs. This graph also confirms that the

model generalizes rather than memorizes, which is good. Since the cross-entropy error is low, I suspect that the domain is also linear separable. If the model manages to get 50% of the classifications right (complete random guessing for instance), then the cross entropy will be 1.0, pushing this number down towards 0 means that it manages to split the two classes more often than not. The graph below will confirm this theory.

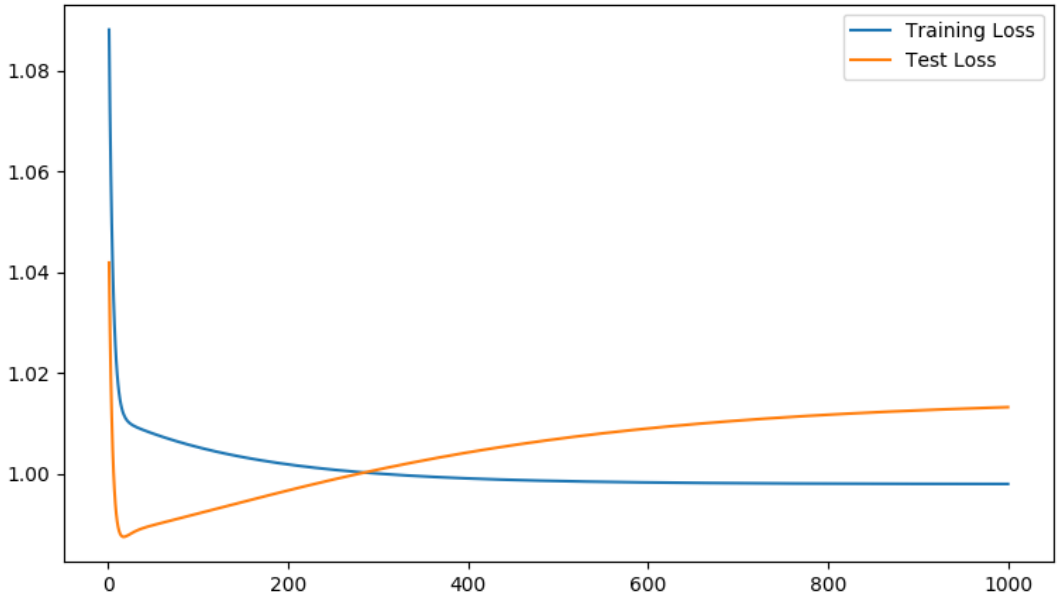


So the graph above shows the class 0 objects as blue dots, class 1 objects as red dots and the decision boundary is also plotted. When plotting the same graph with the test data, similar result occurs, but a few of the objects are slightly on the wrong side of the boundary, which is usually to be expected when dealing with classifications. At this point it is no doubt that the domain is linearly separable and that the model produced generalizes well.

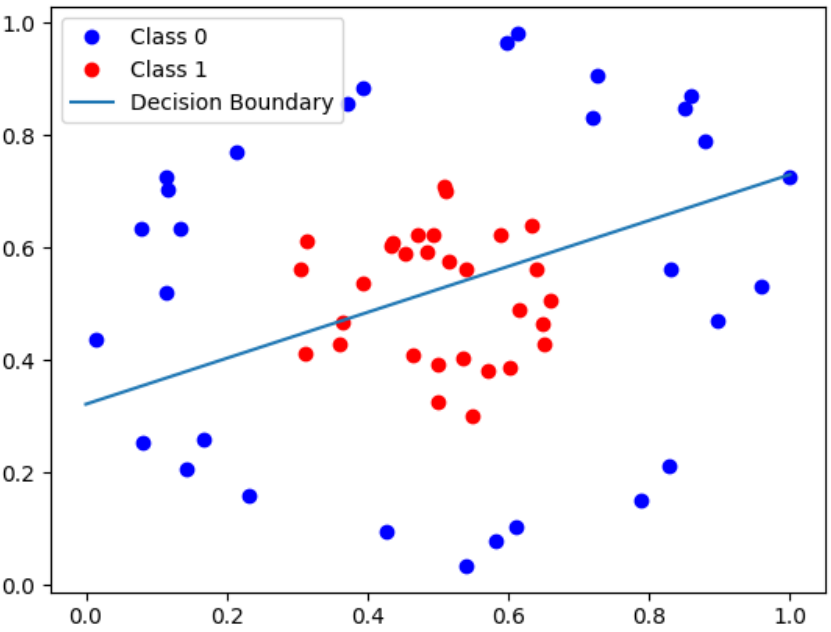


Same graph for the test data points.

2. For this domain I used the same learning rate of 0.005 and initialized the weights randomly as in task 1. The final cross-entropy error is pretty close to 1.0 which is already a sign that the data is not linearly separable. It doesn't make sense to talk about the degree of generalization when the model struggles this bad with classifying the training data, but I can mention that the error is pretty much the same for the test data.



The graph above is the cross-entropy error over 1000 epochs. Both the training loss and test loss is centred around the 1.0 mark. Please not that the characteristics of this graph will change a lot for each time one runs the training, because a small variation in the weights will perhaps push some of the test objects over on the other side and same goes for the training objects, they will always revolve around the 1.0 error mark, but the characteristics of the plot will change.



The graph above shows the classes, color-coded and the decision boundary. Its easy to see why this simple model wasn't able to classify the objects better than with a 50/50 chance, not only is it not linearly separable, but the absolute best the model could achieve was in fact

50/50.

Same plot for the test data points.

[i] Machine Learning, Tom M. Mitchell, Chapter 2