TDT4173: Machine Learning and Case-Based Reasoning

Assignment 1

January 11, 2019

Amar Jaiswal
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

- **Delivery deadline: February 01, 2019** by 23:59.
- **This assignment counts towards 4 % of your final grade.**
- Solutions must be submitted individually.
- Deliver your solution on *Blackboard* before the deadline.
- Please upload your assignment as a PDF file, and package your code into an archive (e.g. zip, rar, tar).
- The programming tasks may be completed in the programming language of your choice.
- Your code is part of your delivery, so please make sure that your code is well-documented and as readable as possible.
- For each programming task you need to give a brief explanation of what you did, answer any questions in the task text, and display results, e.g. plots, in the report. It will be appreciable if the code is well commented (such as: input and output parameters, the task of a function if any exists and critical/complex logic explanatory comments).

Objective: Gain insight into (a) essential machine learning concepts, and (b) how simple learning problems can be solved using linear and logistic regression.

1 Theory [1.5 point]

1. [**0.1 points**] What is *concept learning*, explain with an example?
2. [**0.1 points**] What is function approximation and why do we need them?
3. [**0.4 points**] What is *inductive bias* in the context of machine learning, and why is it so important? Decision tree learning and the candidate elimination algorithm are two different learning algorithms. What can you say about the inductive bias for each of them?

4. **[0.3 points]** What is *overfitting*, and how does it differ from *underfitting*? Briefly explain what a validation set is. How can *cross*-validation be used to mitigate overfitting?

5. **[0.6 points]** Apply candidate elimination (CE) algorithm on the data given below in Table 1, where $\{TreatmentSuccessful\}$ is the target attribute. The tabular data given below is based on physiotherapy questionnaire results for patients having pain concerning musculoskeletal disorders and its treatment successfulness. 'Problem Area' indicates region of the pain, 'Activity Level' describes the current physical activity level of the patient, 'Sleep Quality' indicates the level of sleep quality of the patient and 'Treatment Successful' indicates whether the treatment was successful in lowering the pain or not. The task is to learn to predict the value of Treatment Successful for an arbitrary values of the questionnaires. Describe the version space, specific hypothesis and general hypothesis boundary for this task (represent the version space starting from the initial boundary sets corresponding to the most specific and most generic hypotheses. You must represent the version space when CE algorithim visits a new negative or positive sample/example). The representation for "no value is acceptable" is **'Ø'**, and "any value is acceptable" is **'?'**. Also, the hypothesis space should be restricted to include only conjunctions of the attribute values.

Sex	Problem Area	Activity Level	Sleep Quality	Treatment Successful
Female	Back	Medium	Medium	Yes
Female	Neck	Medium	High	Yes
Female	Shoulder	Low	Low	No
Male	Neck	High	Medium	Yes
Male	Back	Medium	Low	Yes

Table 1: Physiotherapy questionnaire and treatment outcome

2 Programming [2.5 points]

Note on Linear Models

Common to all linear models is that the incoming data is integrated to create a signal computed by taking a linear combination of the input plus a bias. This can be seen in Equation 1, where $\mathbf{x}$ is a d-dimensional column vector $(x_1, x_2, \ldots, x_d)^T$.

$$h(\mathbf{x}) = b + \sum_{i=1}^{d} w_i x_i \tag{1}$$

To simplify notation, most textbooks merge the bias unit b with the weights w_i to get a single weight vector $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_d)^T$, where $w_0 = b$. Consequently, an extra dimension must be prepended to the input vector, i.e. $\mathbf{x} = (1, x_1, x_2, \ldots, x_d)^T$. The

simplification can be seen in Equation 2. Geometrically, this real-valued function can be interpreted as a hyperplane (or simply a line if $d = 1$).

$$h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \begin{bmatrix} w_0 & w_1 & \cdots & w_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \mathbf{w}^T \mathbf{x} \tag{2}$$

This is the representation that we will use for the rest of the assignment.

Task 1: Linear Regression [1.5 points]

A linear regression model learns a real-valued function $y = f(\mathbf{x}) \in R$. We will call the learnt model $h(\mathbf{x})$ and it can be seen defined in Equation 2. In other words, the goal of linear regression is to fit a hyperplane to our data such that $h(\mathbf{x}) \approx f(\mathbf{x})$.

In this assignment, the data is a set of N input-output vectors $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is the input and y_i is the real-valued output. The dataset will be stored in a comma-separated values (CSV) file[1] and will **not** include the prepended 1 as required by the simplifications made in Equation 2. Keep this in mind when loading the data.

To measure how well our hypothesis, or model, $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ approximates $f(\mathbf{x})$ we will use *squared error*: $(h(\mathbf{x}) - f(\mathbf{x}))^2$. The in-sample squared error over all our *training* data is called *mean squared error* (MSE) and can be seen in Equation 3.

$$E_{\text{mse}}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (h(\mathbf{x}_i) - y_i)^2 = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 \tag{3}$$

The mean squared error can be further simplified by recognising that the training data can be put into matrices; see Equation 4 and Equation 5. The input matrix $\mathbf{X}$ is often called a *design matrix* and consists of one (training) example per row – similar to how the data is laid out in the aforementioned CSV file.

$$\mathbf{X} = \begin{bmatrix} -\mathbf{x}_1^T - \\ -\mathbf{x}_2^T - \\ \vdots \\ -\mathbf{x}_N^T - \end{bmatrix} \qquad \text{and} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{4}$$

$$E_{\text{mse}}(\mathbf{w}) = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 \tag{5}$$

Now, to get a good approximation, we need to select weights $\mathbf{w}$ so that the error $E_{\text{mse}}(\mathbf{w})$ is minimised. This is commonly called *ordinary least squares* or OLS. There are several ways to do this, for example, gradient descent, however, in this assignment we will simply take

[1] There will be one example, or input-output vector, per line, where the last number in each line will be the target value y_i.

the derivative of $E_{\text{mse}}(\mathbf{w})$ with respect to $\mathbf{w}$ and then compare it to zero to get the closed-form solution. First though, we need to expand the mean squared error representation so that we can differentiate it. The steps can be seen in Equation **??**; notice that the constant $\frac{1}{N}$ is thrown out as it will not alter the final result.

$$\begin{aligned} E_{\text{mse}}(\mathbf{w}) &= \frac{1}{N}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 \\ &= \frac{1}{N}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= (\mathbf{X}\mathbf{w} - \mathbf{y})^T T(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= ((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= (\mathbf{X}\mathbf{w})^T\mathbf{X}\mathbf{w} - (\mathbf{X}\mathbf{w})^T\mathbf{y} - \mathbf{y}^T(\mathbf{X}\mathbf{w}) + \mathbf{y}^T\mathbf{y} \\ &= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2(\mathbf{X}\mathbf{w})^T\mathbf{y} + \mathbf{y}^T\mathbf{y} \end{aligned} \tag{6}$$

We can now compute $\frac{\partial E_{\text{mse}}(\mathbf{w})}{\partial \mathbf{w}}$ by differentiating each of the three terms in Equation **??** separately, then add them back together – remember, differentiation is a linear operation. This can get quite hairy so with respect to time I will just write down the final derivatives in Equation 7.

$$\frac{\partial \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}}{\partial \mathbf{w}} = 2\mathbf{X}^T\mathbf{X}\mathbf{w} \qquad \frac{\partial 2(\mathbf{X}\mathbf{w})^T\mathbf{y}}{\partial \mathbf{w}} = 2\mathbf{X}^T\mathbf{y} \qquad \frac{\partial \mathbf{y}^T\mathbf{y}}{\partial \mathbf{w}} = 0 \tag{7}$$

Putting the pieces together we get Equation 8.

$$\begin{aligned} \frac{\partial E_{\text{mse}}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}}{\partial \mathbf{w}} - \frac{\partial 2(\mathbf{X}\mathbf{w})^T\mathbf{y}}{\partial \mathbf{w}} + \frac{\partial \mathbf{y}^T\mathbf{y}}{\partial \mathbf{w}} \\ &= 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 0 \\ &= 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} \end{aligned} \tag{8}$$

Finally, let's throw away the constant terms, compare against zero, and solve for $\mathbf{w}$ in Equation 9. This will yield the correct set of weights because $E_{\text{mse}}(\mathbf{w})$ is a convex function. In the last step both sides are multiplied by $(\mathbf{X}^T\mathbf{X})^{-1}$ to get the final result.

$$\begin{aligned} \frac{\partial E_{\text{mse}}(\mathbf{w})}{\partial \mathbf{w}} &= 0 \\ \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} &= 0 \\ \mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\ \mathbf{w} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \end{aligned} \tag{9}$$

And there we have it, the closed-form solution for ordinary least squares. It requires that $\mathbf{X}^T\mathbf{X}$ be non-singular, however, there are ways to circumvent this issue, for example, by using the Moore-Penrose pseudoinverse instead – typically `pinv()` in your favourite linear algebra package, such as NumPy or MATLAB.

1. Henvis til src-code
2. Find weights for training data first, and then test with those weights on the test data. Print both results. This means that there must be another metod that calls the Error-function WITHOUT training again for the second csv.

1. **[0.3 points]** Implement linear regression with ordinary least squares (OLS) using the closed-form solution seen in Equation 9. *Tip: This can be done in a single line of code!*

2. **[0.5 points]** Load the data in `train_2d_reg_data.csv` (training data) and `test_2d_reg_data.csv` (test data) and use your OLS implementation to find a set of good weights for the training data. Show the weights as well as the model error $E_{\text{mse}}(\mathbf{w})$ for the *training* and *test* set after training. Is your model generalizing well?

3. **[0.7 points]** Load the data in `train_1d_reg_data.csv` and `test_1d_reg_data.csv` and use your OLS implementation to find a set of good weights for the training data. Using these weights, make a plot of the line fitting the data and show this in the report. Does the line you found fit the data well? If not, discuss in broad terms how this can be remedied. *Tip: Remember, for this dataset there are only two weights: the first is the bias, while the second is the slope.*

Task 2: Logistic Regression [1 point]

Logistic regression is, despite its name, a model that can be used for binary classification. Namely, it is a model that fits a probability distribution $\Pr(y|\mathbf{x})$ where $y \in \{0, 1\}$. That is, the model outputs the probability that the inputs $\mathbf{x}$ belong to the *class* $y = 1$. This probability, $\Pr(y = 1|\mathbf{x})$, can be captured by a nonlinear function called the *logistic* function, which can be seen defined in Equation 10.

$$\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}} \qquad \text{where} \qquad z = h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \tag{10}$$

It is a soft threshold with a distinct S shape, otherwise known as a sigmoid function. This is illustrated in fig:sigmoid.

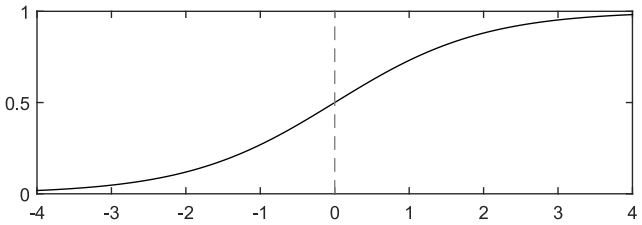

Figure 1: The logistic function with a vertical line placed at zero indicating 0.5 probability.

As we can see, the logistic function can be interpreted as a probability because it is bounded between zero and one. Thus, when a linear signal $z = h(\mathbf{x})$ is large $\Pr(y = 1|\mathbf{x})$ is high and $\Pr(y = 0|\mathbf{x})$ is low, and vica versa when z is low. This can be codified by letting $\Pr(y = 1|\mathbf{x}) = \sigma(z)$ and $\Pr(y = 0|\mathbf{x}) = 1 - \sigma(z)$. A more compact representation of this probability distribution is shown in Equation 11.

$$\Pr(y|\mathbf{x}) = \sigma(z)^y(1-\sigma(z))^{1-y} = \sigma(\mathbf{w}^T\mathbf{x})^y(1-\sigma(\mathbf{w}^T\mathbf{x}))^{1-y} \tag{11}$$

Now, just like with linear regression, we need some way to ascertain which hypothesis to select. Logistic regression learns its weights by *maximising the likelihood* that a given weight vector $\mathbf{w}$ can result in a correct prediction for the data. The likelihood over all N training examples is depicted in Equation 12.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \prod_{i=1}^{N} \Pr(y_i|\mathbf{x}_i) \\ &= \prod_{i=1}^{N} \sigma(z)^{y_i}(1-\sigma(z))^{1-y_i}\end{aligned} \tag{12}$$

It is generally easier to work with the log-likelihood instead of the likelihood. The logarithmic function is monotonically increasing which means that the maximum remains the same. Moreover, $\sigma(z)$ is a positive function so taking the logarithm is safe. We alter the likelihood computation by taking the natural logarithm as seen in Equation 13.

$$\begin{aligned}\ell(\mathbf{w}) &= \ln \mathcal{L}(\mathbf{w}) \\ &= \sum_{i=1}^{N} (y_i \ln \sigma(z) + (1-y_i)\ln(1-\sigma(z)))\end{aligned} \tag{13}$$

Minimising the *negative* log-likelihood function is equivalent to maximising the log-likelihood. This particular form is called the *cross-entropy* error function, and it is the error measure we will be using to see how good the model is. It can be seen in Equation 14. Notice that we also divide by the number of examples; this just scales the result.

$$\begin{aligned}E_{\text{ce}}(\mathbf{w}) &= -\frac{1}{N}\ell(\mathbf{w}) \\ &= -\frac{1}{N}\sum_{i=1}^{N} (y_i \ln \sigma(z) + (1-y_i)\ln(1-\sigma(z)))\end{aligned} \tag{14}$$

Now, if you have done the linear regression task in the previous section your immediate thought might be to differentiate $E_{\text{ce}}(\mathbf{w})$ with respect to $\mathbf{w}$, set the derivative equal to zero, and solve the equation. Sadly, there is no closed-form solution so we will not be able to do this. We can however find an approximate solution numerically. We will be using an iterative optimisation algorithm called *gradient descent* to do this. In general, it works by taking the derivative of an error function – sometimes called a loss function – with respect to the parameters $\mathbf{w}$ of the model, and then alter the parameters in the direction of the *negative* gradient. This can be summarised as: $\mathbf{w}(k+1) \leftarrow \mathbf{w}(k) - \eta\frac{\partial E_{\text{ce}}(\mathbf{w})}{\partial \mathbf{w}}$, where $\mathbf{w}(k)$ signifies the state of the model parameters at iteration k, and η is known as the *learning rate* and decides how much the parameters should change for each application of the rule. This *update rule* is repeated until convergence or until the maximum number of iterations

has been reached. With that in mind, let's find the derivative of the cross-entropy error function with respect to $\mathbf{w}$. This can be done by applying the chain rule as seen in Equation 15.

$$\frac{\partial E_{\text{ce}}(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial E_{\text{ce}}(\mathbf{w})}{\partial \sigma(z)} \frac{\partial \sigma(z)}{\partial z} \frac{\partial z}{\partial \mathbf{w}} \tag{15}$$

As with linear regression, we will do each derivative separately and then combine the results afterwards. First off, the derivative of the cross-entropy with respect to its inputs can be seen in Equation 16. Again, the constant term is removed as it will not impact the end result.

$$\begin{aligned}
\frac{\partial E_{\text{ce}}(\mathbf{w})}{\partial \sigma(z)} &= \frac{\partial(-\frac{1}{N}\sum_{i=1}^{N} y_i \ln \sigma(z) + (1-y_i)\ln(1-\sigma(z)))}{\partial \sigma(z)} \\
&= -\sum_{i=1}^{N}(\frac{\partial y_i \ln \sigma(z) + (1-y_i)\ln(1-\sigma(z))}{\partial \sigma(z)}) \\
&= -\sum_{i=1}^{N}(\frac{\partial y_i \ln \sigma(z)}{\partial \sigma(z)} + \frac{\partial (1-y_i)\ln(1-\sigma(z))}{\partial \sigma(z)}) \\
&= -\sum_{i=1}^{N}(\frac{y_i}{\sigma(z)} - \frac{1-y_i}{1-\sigma(z)}) \\
&= \sum_{i=1}^{N} \frac{\sigma(z) - y_i}{\sigma(z)(1-\sigma(z))}
\end{aligned} \tag{16}$$

The second term to differentiate is the logistic function $\sigma(z)$. This can be seen in Equation 17. The result has the remarkable property that it can be expressed in terms of the logistic function.

$$\begin{aligned}
\frac{\partial \sigma(z)}{\partial z} &= \frac{\partial}{\partial z}\left[\frac{1}{1+e^{-z}}\right] \\
&= \left[\frac{-1}{(1+e^{-z})^2}\right]\left[\frac{\partial}{\partial z}(1+e^{-z})\right] \\
&= \left[\frac{-1}{(1+e^{-z})^2}\right]\left[\frac{\partial}{\partial z}(1) + \frac{\partial}{\partial z}(e^{-z})\right] \\
&= \left[\frac{-1}{(1+e^{-z})^2}\right]\left[0 + (-1)e^{-z}\right] \\
&= 0 + \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \left[\frac{1}{1+e^{-z}}\right]\left[\frac{e^{-z}}{1+e^{-z}}\right] \\
&= \sigma(z)(1-\sigma(z))
\end{aligned} \tag{17}$$

The final term is the derivative of the linear combination of the inputs defined way back in Equation 2, namely $z = h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}_i$. This simple derivative can be seen in Equation 18.

$$\begin{aligned} \frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial \mathbf{w}^T\mathbf{x}_i}{\partial \mathbf{w}} \\ &= \mathbf{x}_i \end{aligned} \tag{18}$$

Now that we have all the pieces, let's put them back together in Equation 19 to get the derivative of the cross-entropy error function with respect to the model parameters $\mathbf{w}$. As seen by the summation, this version of the derivative is taken over all of the N training examples.

$$\begin{aligned} \frac{\partial E_{\text{ce}}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial E_{\text{ce}}(\mathbf{w})}{\partial \sigma(z)} \frac{\partial \sigma(z)}{\partial z} \frac{\partial z}{\partial \mathbf{w}} \\ &= \sum_{i=1}^{N} \frac{\sigma(z) - y_i}{\sigma(z)(1-\sigma(z))} \sigma(z)(1-\sigma(z))\mathbf{x}_i \\ &= \sum_{i=1}^{N} (\sigma(z) - y_i)\mathbf{x}_i \end{aligned} \tag{19}$$

The final update rule for gradient descent for logistic regression can be seen in Equation 20. This is the core equation that you must implement in the tasks for this section.

$$\begin{aligned} \mathbf{w}(k+1) &\leftarrow \mathbf{w}(k) - \eta \sum_{i=1}^{N} (\sigma(z) - y_i)\mathbf{x}_i \\ &\leftarrow \mathbf{w}(k) - \eta \sum_{i=1}^{N} (\sigma(\mathbf{w}(k)^T\mathbf{x}_i) - y_i)\mathbf{x}_i \end{aligned} \tag{20}$$

Now the observant reader might ask: "After training, how can we make a prediction?". Let's say we have picked an initial weight vector, trained for a few iterations using the update rule in Equation 20, and now want to know the class of a new, unseen data point. Intuition says that we simply have to input our data point to $\sigma(z)$, i.e. the logistic regression model. However, this function will yield a number betwixt 0 and 1, rather than 0 and 1 exactly. Referring back to fig:sigmoid, the indicated vertical line at 0 is one common *decision point* for which class to yield. In other words, when the probability of class $y = 1$ is greater than or equal to 0.5 we can predict $y = 1$, otherwise we predict $y = 0$. A second way to interpret this is to look at z since when $z \geq 0$ the probability will be greater than or equal to 0.5. This means that we can predict $y = 1$ when $\mathbf{w}^T\mathbf{x} \geq 0$ and $y = 0$ otherwise.

The data for this part of the assignment can also be found in CSV files, and they are organized exactly like they were for the linear regression task: one training example per line. Each line contains a set of features, where the last number is the class label $\in \{0, 1\}$. Thus,

for a d-dimensional dataset with N examples, there will be N lines with $d+1$ numbers per line. Remember, that logistic regression still uses the representation presented in Equation 2 which means that when loading the data you must prepend each data vector with the number 1.

For the purposes of this assignment, usage of libraries that perform automatic differentiation is **not** allowed. This includes libraries such as Theano, Autograd, and TensorFlow.

1. [**0.5 points**] Load the data in `cl_train_1.csv` and `cl_test_1.csv` and use your logistic regression implementation to train on the data in the training set. Is the data *linearly separable*? Explain your reasoning. Additionally, show a plot of the cross-entropy error for both the training and test set over 1000 iterations of training. What learning rate η and initial parameters $\mathbf{w}$ did you select? Is your model generalising well? *Keep in mind that you may have to test several different learning rates to get a good classification. It may be helpful to plot the decision boundary*[2] *to get a visual representation of the fit. This will allow you to see how well you are doing.*

2. [**0.5 points**] Load the data in `cl_train_s2.csv` and `cl_test_2.csv` and use your logistic regression implementation to train on the data in the training set. Is the data *linearly separable*? Explain your reasoning. Plot the *decision boundary* as explained in the previous task as well as the data points in the training and test set. Discuss what can be done for logistic regression to correctly classify the dataset. Make the necessary changes to your implementation and show the new decision boundary in your report.

[2] A decision boundary is the set of points that fall on the line where $\mathbf{w}^T\mathbf{x} = 0$. For the datasets in these tasks, this will look like a line separating the two classes of data points.