INF554 - Machine Learning I

Lab 2: Supervised Classification

1 Logistic Regression and Gradient Descent

In linear regression, we got a real-valued response

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

i.e., a linear combination of inputs, where $y \in \mathbb{R}$.

In classification, we want an *indication* of how likely an instance is to belong to a particular class; a probability $\in [0, 1]$.

Given a real valued a, we can squish it to range $\sigma(a) \in [0, 1]$ by feeding it through the **logistic function** aka **sigmoid function**[1]:

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{1}$$

Therefore we can treat this as a probability, i.e.,

$$\begin{aligned} P(y = 1|\mathbf{x}) &= \sigma(\boldsymbol{\beta}^\top \mathbf{x}) \\ P(y = 0|\mathbf{x}) &= 1 - \sigma(\boldsymbol{\beta}^\top \mathbf{x}) \end{aligned}$$

where we omit the bias term and suppose that both $\boldsymbol{\beta}$ and $\mathbf{x}$ are column vectors.

Task 1
Implement the sigmoid function Eq. (1).

To fit the model, we want weights (i.e., parameters) $\boldsymbol{\beta}$ to reduce error. Our error function is

$$E_i(\boldsymbol{\beta}) = \begin{cases} -[1 - \sigma(\boldsymbol{\beta}^\top \mathbf{x}_i)] & \text{if } y_i = 0 \\ -\sigma(\boldsymbol{\beta}^\top \mathbf{x}_i) & \text{if } y_i = 1 \end{cases} \tag{2}$$

for a given instance (i.e., i-th instance). Note that, assuming iid data, we may write, for n training instances,

$$E(\boldsymbol{\beta}) = \prod_{i=1}^{n} E_i(\boldsymbol{\beta}) \tag{3}$$

Task 2
Implement the cost (error) function, Eq. (3)

As in OLS, we want to find values of weights which minimize the error.

[1] `https://en.wikipedia.org/wiki/File:Logistic-curve.svg`

Ques 1
Derive the gradient of the cost function with respect to the model's parameters $\nabla_{\boldsymbol{\beta}} E(\boldsymbol{\beta})$. Hint 1: It's easier to take the log first. Hint 2: You can rewrite the case notation of Eq. (2) using exponents: $\sigma_i^{y_i}(1-\sigma_i)^{1-y_i}$ where $\sigma_i \equiv \sigma(\boldsymbol{\beta}^\top \mathbf{x}_i)$. Hint 3: $\sigma' = (1-\sigma)\sigma$.

You'll notice that, unlike under OLS, it's not a closed form expression; you have $\boldsymbol{\beta}$ on the right hand side of the equation. Nevertheless, we can use numerical methods to find optimal $\hat{\boldsymbol{\beta}}$.

Task 3
Implement the gradient-of-the-cost function (which you just derived above).

What we have now is a vector $\mathbf{g} = \nabla_{\boldsymbol{\beta}} E(\boldsymbol{\beta})$; the gradient of the function. Note that the dimensionality is the same as the input. We can move in the direction of the gradient and thus descend the function. This is the idea of **Gradient Descent**: we iteratively follow the gradient down the error surface. We repeatedly carry out

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \alpha \nabla_{\boldsymbol{\beta}} E(\boldsymbol{\beta}_t) \tag{4}$$

where α is the **learning rate**; for $t = 1, 2, \ldots, T$, such that $\hat{\boldsymbol{\beta}} \leftarrow \boldsymbol{\beta}_T$.

Task 4
Use the functions implemented so far to implement gradient descent, Eq. (4), for a fixed number of steps (say, $T = 500\,000$) and learning rate to (say, $\alpha = 0.005$).

Task 5
Implement a function to provide predictions $\hat{y} \in \{0, 1\}$ for any given $\mathbf{x}$ and $\hat{\boldsymbol{\beta}}$ (from the previous task) by assigning $\hat{y}_i = 1$ whenever $\sigma_i \geq 0.5$.

You have have now turned logistic *regression*, into a classifier.

There is data provided in the `data/` folder. Suppose that each column represents an exam score during a course, with the class label indicated `admitted` or not into a Masters program at some university. We want a model to give the probability that a student will be admitted based on the two grades of two courses. Some plotting code is already provided.

Task 6
Make a train-test split of the data provided and evaluate your classifier (similarly to as in the first lab). Output and/or plot the error $E(\boldsymbol{\beta}_t)$ both on the training *and* test set, for $t = 1, \ldots, T$. Hint: You may want to plot the average of a moving window.

Ques 2
Knowing that the cost function is convex, are you convinced that with a suitable learning rate, gradient descent will always converge to the minimum? What are some examples of poor learning rates?

Task 7
Following your considerations in the previous question – can you find a better learning rate, such that you reduce error more efficiently (for smaller T?).

Task 8
(Bonus) Instead of the whole training set, select a random subset (i.e., minibatch) of 10 examples for each iteration. This is known as **stochastic gradient descent**.

Ques 3
(Bonus) What are the advantages **Stochastic gradient descent** over classical **Gradient descent** ? Especially with a non-convex loss function.

2 k-Nearest Neighbours

The method kNN takes a different approach to modeling $P(Y = 1|\mathbf{x})$. It is a non-parametric lazy method, and does not have a "model" as such. It predicts with

$$P(Y = 1|\mathbf{x}) \approx \frac{1}{k} \sum_{\mathbf{x}_i \in \mathsf{Ne}_k(\mathbf{x})} y_i$$

where $\mathsf{Ne}_k(\mathbf{x})$ is the *neighbourhood* of the k training examples closest to $\mathbf{x}$ (typically measured by Euclidean distance).

Ques 4
What is the effect of different values of k? Hint: Draw on paper and/or empirically test different values.

Ques 5
What is the complexity of making a prediction with a naive implementation of kNN? When is this likely to be a problem in practice?

Task 9
(Bonus) Implement this function to create a kNN classifier and evaluate it on the data provided. Hint: there is not really any training stage here; simple store the training instances to search over later.