Report for project 3

Jon Audun, Mikael Ravndal and Adam P W Sørensen

December 11, 2018

In this project we analysed some cooking data and did fairly well.

1 Introduction

The concept of machine learning have gained a hughe popularity boost over the last couple of years. And this is no wonder: the techniques have a wide range of applications and can be a major asset if you know when to use what.

When to use what is exactly what we're going to have a brief peek into in this project. We will evaluate the performance of three different methods for classification on a data set consisting of recipies labeled with the cuisines they belongs to. That is we are looking at a classification problem with multiple classes. The methods we will look at are logistic regression, support vector machines and random forests.

> All the code we have implemented can be found at `https://github.com/???`. A print of the jupyter notebook is also attached at the end of this report.

2 Methods

First we need to establish some notation. In the following we assume that we are given $N \in \mathbb{N}$ samples consisting of $p \in \mathbb{N}$ features and one target. Further we let $K \in \mathbb{N}$ be the number of classes in our classification problem. We then denote by $x_{i,j}$ the j-th feature of the i-th sample and by $t_i \in \{1, 2, \ldots, K\}$ the target of the i-th sample.

2.1 Logistic Regression

Let $\boldsymbol{x}_i := (1, x_{i,1}, x_{i,2}, \ldots, x_{i,p})$ be the row-vector in $\mathbb{R}^{p+1}$ consisting of a one followed by the features of sample i. Also let $\boldsymbol{\beta_k}$ be a column-vector in $\mathbb{R}^{p+1}$ for $k \in \{1, 2, \ldots, K-1\}$ and let $\boldsymbol{\theta} := [\boldsymbol{\beta}_1\ \boldsymbol{\beta}_2\ \ldots\ \boldsymbol{\beta}_{K-1}]$ be the matrix with the $\boldsymbol{\beta}_k$'s as it's columns. In the case of multinomial logistic regression our model is then given by

$$\begin{aligned} P_k(\boldsymbol{x};\boldsymbol{\theta}) &= \frac{\exp(\boldsymbol{x}\boldsymbol{\beta}_k)}{1+\sum_{l=1}^{K-1}\exp(\boldsymbol{x}\boldsymbol{\beta}_l)} \qquad k \in \{1, 2, \ldots, , K-1\} \\ P_K(\boldsymbol{x};\boldsymbol{\theta}) &= \frac{1}{1+\sum_{l=1}^{K-1}\exp(\boldsymbol{x}\boldsymbol{\beta}_l)} \end{aligned} \tag{1}$$

where the $\boldsymbol{\beta_k}$'s are the coefficients we wish to estimate. $P_k(\boldsymbol{x};\boldsymbol{\theta})$ is then the probability that a sample with features $\boldsymbol{x}$ belongs to category k.

The way we fit our model in the case of logistic regression deviates slightly from the usual proceedure with the introduction of a loss function. We now instead define a function we wish to maximize, namely the log-likelihood function given by

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N}\sum_{k=1}^{K}\chi_{\{k\}}(t_i)\log(P_k(\boldsymbol{x}_i;\boldsymbol{\theta})) \tag{2}$$

However the difference doesn't go further than the fact that this function shouldn't be interpreted exactly as a loss function. Other than that we proceed as usual by minimizing the negative of this function in order to maximize the original function. To do this we first need to compute the derivative of this function with respect to $\boldsymbol{\theta}$. Some computations gives us

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \beta_{k,j}} = \sum_{i=1}^{N}\chi_{\{k\}}(t_i)x_{i,j}(1-P_k(x_i;\theta)) \tag{3}$$

We are now all set to use the gradient method of your choice to minimize $-L$ as a function of $\boldsymbol{\theta}$.

2.1.1 Gradient descent

The perhaps simplest approach to minimizing a real valued function f iteratively, is to set some starting point $\boldsymbol{x}_0$, compute the gradient ∇f in $\boldsymbol{x}_0$, and move $\eta > 0$ in the opposite direction of this gradient in order to obtain $\boldsymbol{x}_1 = \boldsymbol{x}_0 - \eta\nabla f(\boldsymbol{x}_0)$. Then one iterates by setting $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \eta\nabla f(x_0)$ for $i = 1, 2, ...$ This is the framework of gradient descent. For small enough η this implies $f(\boldsymbol{x}_i) > f(\boldsymbol{x}_{i+1})$. The problem with this approach is to find out what η is small enough. If we choose η too small, it might take days for our algorithm to converge. If we on the contrary choose η to big, it might not converge at all.

2.1.2 Newton-Raphson

The Newton-Raphson method is a kind of gradient descent method, except we choose the learning rate in a more clever way. We thus start out by choosing a random starting point $\boldsymbol{x}_0$, just as in the case of gradient descent. The update step however is a bit different:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \frac{\partial^2 f(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_i^T} \frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i} \tag{4}$$

The result is a method that converges way faster than normal gradient descent. The draw back is however the need to compute the Hessian of your objective function. For a thourough derivation of this method consult [**?**]

2.1.3 Stochastic gradient descent

To avoid overfitting when training neural networks, training on randomly picked batches of the training data for each iteration is a good regularization method. If we let b be our batch size, the first step of this method consists of dividing the training data into $N//b$ batches. One does this by picking from the N data-points without replacement. The second step is then to perform a gradient descent as described in section 2.1.1 with a new batch for each iteration.

2.2 

To call a Support Vector Machine classifier with no kernel we use the following Python code:

2.3 Decision trees

3 Model Selection and Verification

4 Results

5 Conclusion

We did good.