Machine Learning for Agricultural Applications

Assignment 2

Prof. Dr. Niels Landwehr
Dr. Julian Adolphs

Summer Term 2020

Release: 07.05.2020
Discussion: 14.05.2020

Task 1 – Linear Regression [10 points]

Write down the objective function $L(\Theta)$ for linear regression with squared loss and L_2-Regularization based on the linear model with implicit bias:

$$f_{\mathbf{w}} = \mathbf{w}^T \mathbf{x}\,, \qquad \mathbf{w} \in \mathbb{R}^d, \quad \mathbf{x} \in \mathbb{R}^d.$$

Calculate the gradient of the objective function.

$$L(\mathbf{\Theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(f_{\mathbf{\Theta}}(\mathbf{x}_i), y_i\right) + \lambda\, R(\mathbf{\Theta}) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \sum_{i=1}^{n} \mathbf{w}_i^2$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \frac{2}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)\, \mathbf{x}_i + 2\,\lambda\, \mathbf{w}$$

Task 2 – Linear Regression, Gradient Descent [15 points]

a) Write a python program, that calculates the linear regression with gradient descent for $d = 2$ (i.e. 2 features). Plot the results (the points and the resulting linear function). Generate the points using suitable random generators.

b) Also calculate the closed form solution and compare with the result from a).

`Solution:    ex2_LinReg.py`

Task 3 – Classification [10 points]

Show that the objective function for classification with cross-entropy loss (without regularization) based on the linear model without bias

$$\mathbf{f}_{\mathbf{W}}(\mathbf{x}_i) = \mathbf{W}\mathbf{x}_i\,, \qquad \mathbf{W} \in \mathbb{R}^{k\times d}, \quad \mathbf{x}_i \in \mathbb{R}^d$$

can be written as

$$L(\mathbf{W}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\,\log\left(\frac{\exp(\mathbf{W}_j\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}\right).$$

where $\mathbf{W}_j$ is the j-th row of matrix $\mathbf{W}$ (that is related to class c_j), and $\delta_{y_i c_j}$ is the Kronecker symbol with

$$\delta_{y_i c_j} = \begin{cases} 1, & \text{if} \quad y_i = c_j \\ 0, & \text{if} \quad y_i \neq c_j\,. \end{cases}$$

Show that the gradient descent for cross-entropy can be calculated iteratively by

$$\mathbf{W}_j^{(k+1)} = \mathbf{W}_j^{(k)} + \alpha\frac{1}{n}\sum_{i=1}^{n}\left(\delta_{y_i c_j} - \frac{\exp(\mathbf{W}_j\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}\right)\mathbf{x}_i\,,$$

where $\mathbf{W}_j^{(k)}$ is the k-th iteration step of the j-th row of matrix $\mathbf{W}$.

Hint: For the differentiation of the loss function, you should take advantage of the logarithm rule $\log\left(\frac{a}{b}\right) = \log a - \log b$. With log we mean the natural logarithm.

$$\ell(f_{\mathbf{W}}(\mathbf{x}_i), y_i) = -\sum_{j=1}^{k}\delta_{y_i c_j}\log p(y_i = c_j|\mathbf{x}_i, \mathbf{W})$$

$$p(y_i = c_j|\mathbf{x}_i, \mathbf{W}) = \frac{\exp(f_{\mathbf{W}}(\mathbf{x}_i)_j)}{\sum_{l=1}^{k}\exp(f_{\mathbf{W}}(\mathbf{x}_i)_l)}$$

$$\Longrightarrow \ell(f_{\mathbf{W}}(\mathbf{x}_i), y_i) = -\sum_{j=1}^{k}\delta_{y_i c_j}\log\left(\frac{\exp(f_{\mathbf{W}}(\mathbf{x}_i)_j)}{\sum_{l=1}^{k}\exp(f_{\mathbf{W}}(\mathbf{x}_i)_l)}\right)$$

$$L(W) = \frac{1}{n}\sum_{i=1}^{n}\ell(f_{\mathbf{W}}(\mathbf{x}_i), y_i)$$

$$\Longrightarrow L(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\log\left(\frac{\exp(f_{\mathbf{W}}(\mathbf{x}_i)_j)}{\sum_{l=1}^{k}\exp(f_{\mathbf{W}}(\mathbf{x}_i)_l)}\right)$$

$$\hookrightarrow L(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\log\left(\frac{\exp(\mathbf{W}_j\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}\right) \qquad \textbf{qed}.$$

We further convert $L(\mathbf{W})$

$$L(\mathbf{W}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\left(\log\left(\exp(\mathbf{W}_j\mathbf{x}_i)\right) - \log\left(\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)\right)\right)$$

$$\Longrightarrow L(\mathbf{W}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\left(\mathbf{W}_j\mathbf{x}_i - \log\left(\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)\right)\right)$$

Now we need the partial derivative of $L(\mathbf{W})$:

$$\begin{aligned}
\frac{\partial L(\mathbf{W})}{\partial \mathbf{W}_m} &= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\frac{\partial \mathbf{W}_j}{\partial \mathbf{W}_m}\mathbf{x}_i + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\delta_{y_i c_j}\frac{\frac{\partial}{\partial \mathbf{W}_m}\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\underbrace{\sum_{j=1}^{k}\delta_{y_i c_j}\delta_{jm}}_{j=m}\,\mathbf{x}_i + \frac{1}{n}\sum_{i=1}^{n}\underbrace{\sum_{j=1}^{k}\delta_{y_i c_j}}_{=1}\underbrace{\frac{\exp(\mathbf{W}_m\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}}_{\text{because no } j \text{ here}}\mathbf{x}_i \\
&= -\frac{1}{n}\sum_{i=1}^{n}\delta_{y_i c_m}\,\mathbf{x}_i + \frac{1}{n}\sum_{i=1}^{n}\frac{\exp(\mathbf{W}_m\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}\mathbf{x}_i \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left(\delta_{y_i c_m} - \frac{\exp(\mathbf{W}_m\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}\right)\mathbf{x}_i
\end{aligned}$$

Now we get for the gradient descent steps:

$$\begin{aligned}
\mathbf{W}_j^{(k+1)} &= \mathbf{W}_j^{(k)} - \alpha\,\frac{\partial L(\mathbf{W})}{\partial \mathbf{W}_j} \\
\hookrightarrow \mathbf{W}_j^{(k+1)} &= \mathbf{W}_j^{(k)} + \alpha\,\frac{1}{n}\sum_{i=1}^{n}\left(\delta_{y_i c_m} - \frac{\exp(\mathbf{W}_m\mathbf{x}_i)}{\sum_{l=1}^{k}\exp(\mathbf{W}_l\mathbf{x}_i)}\right)\mathbf{x}_i \qquad \textbf{qed.}
\end{aligned}$$

Task 4 – Classification, Gradient Descent [15 points]

Write a python program, that calculates a linear classification with gradient descent for $d = 2$ features and $k = 3$ classes. That can be 3 clusters of points, described by their coordinates (x_1, x_2) and their class labels $y_{\text{label}} \in \{1, 2, 3\}$. Use one-hot-encodings for the class labels $\{1, 2, 3\}$, i.e. class 1: $y = (1, 0, 0)^T$, class 2: $y = (0, 1, 0)^T$ and class 3: $y = (0, 0, 1)^T$. You can use the data given below or create your own data manually or with a random number generator.

```
X = np.array([(-1.,2.5),(-2.,5.),(-1.5,4.),(-1.,2.3),(-2.5,6.5),(-1.8,4.),
     (-1.2,-2.5),(-2.3,-3.),(-1.8,-4.),(-1.9,-2.3),(-2.9,-3.5),(-1.7,-4.),
              (1.,-4.5),(0.2,5.),(0.5,-3.),(1.3,2.3),(2.5,-1.0),(1.8,3.)])

y = np.array([(1,0,0), (1,0,0), (1,0,0), (1,0,0), (1,0,0), (1,0,0),
              (0,1,0), (0,1,0), (0,1,0), (0,1,0), (0,1,0), (0,1,0),
              (0,0,1), (0,0,1), (0,0,1), (0,0,1), (0,0,1), (0,0,1)])
```

Hints: It could be useful to define a function `sigma(x)`. If possible avoid loops, prefere vectorization!

a) Plot the progress of the gradient descent by plotting the loss function.

b) Try to visualize the classification boundaries.

c) Use the matrix $\mathbf{W}$ for a class prediction for 5 test points (new points, that are not part of the training set above) and interpret the result.

Solution: ex2_SoftmaxReg.py

and: ex2_sklearnClass.py
