Week 10 Tutorial: Stochastic Gradient Descent

POP77032 Quantitative Text Analysis for Social Scientists

Tom Paskhalis

Exercise 1: Boolean OR

- Having considered the Boolean AND function in the lecture, let's now look at the Boolean OR function.
- The OR function is defined as follows:

$$\text{OR}(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 = 1 \text{ or } x_2 = 1 \\ 0 & \text{if } x_1 = 0 \text{ and } x_2 = 0 \end{cases}$$

- We will start by defining the complete dataset and applying a simple linear regression to it.
- First, try implementing the OR function using a single-layer perceptron (SLP) using stochastic gradient descent (SGD) with the MSE loss function, as we did for the AND function in the lecture.
- As we discussed in the lecture, the MSE loss function isn't ideal for binary classification problems, so, as an alternative, we can try implementing it using the negative log-likelihood.

Negative Log-Likelihood (NLL)

- Consider the likelihood function for data drawn from a binomial distribution

$$\ell(\pi) = \prod_{i=1}^{N} \pi^{y_i}(1-\pi)^{1-y_i} = \pi^{\sum_{i=1}^{N} y_i}(1-\pi)^{N-\sum_{i=1}^{N} y_i}$$

- Taking the logarithm of the likelihood function gives us the log-likelihood:

$$L(\pi) = \log[\pi^{\sum_{i=1}^{N} y_i}(1-\pi)^{N-\sum_{i=1}^{N} y_i}] = (\sum_{i=1}^{N} y_i)\log(\pi) + (N - \sum_{i=1}^{N} y_i)\log(1-\pi)$$

$$= \sum_{i=1}^{N} y_i \log(\pi) + (1-y_i)\log(1-\pi)$$

- The negative log-likelihood (NLL) is then defined as:

$$\text{NLL}(\pi) = -L(\pi) = -\sum_{i=1}^{N} y_i \log(\pi) + (1-y_i)\log(1-\pi)$$

NLL with Logistic Regression

- Previously we considered π as just $P(y_i = 1)$, that is the probability of success.
- But in practice we want to model that probability of success as a function of the input features $\mathbf{x}_i$.
- While we could pick different functional forms for modelling $P(y_i = 1|\mathbf{x}_i)$, the most common approach is to use the logistic function (also knows as the sigmoid function in the ML litearture):

$$\pi = P(y_i = 1|\mathbf{x}_i) = \sigma(\mathbf{x}_i^\top \boldsymbol{\beta}) = \frac{1}{1 + e^{-\mathbf{x}_i^\top \boldsymbol{\beta}}}$$

- Therefore, the NLL for a logistic regression model can then be expressed as:

$$\text{NLL}(\boldsymbol{\beta}) = -\sum_{i=1}^{N} \left[y_i \log(\sigma(\mathbf{x}_i^\top \boldsymbol{\beta})) + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})) \right]$$

Derivation

- Since we are interested in minimizing the NLL, we need to compute its gradient with respect to the parameters $\boldsymbol{\beta}$:

$$\nabla_{\boldsymbol{\beta}} \text{NLL}(\boldsymbol{\beta}) = -\sum_{i=1}^{N} \left[y_i \frac{1}{\sigma(\mathbf{x}_i^\top \boldsymbol{\beta})} \sigma'(\mathbf{x}_i^\top \boldsymbol{\beta}) \mathbf{x}_i + (1 - y_i) \frac{1}{1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})} (-\sigma'(\mathbf{x}_i^\top \boldsymbol{\beta})) \mathbf{x}_i \right]$$

- As the derivative of the sigmoid function is given by:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

- Substituting this into the gradient expression, we get:

$$\nabla_{\boldsymbol{\beta}} \text{NLL}(\boldsymbol{\beta}) = -\sum_{i=1}^{N} \left[y_i (1 - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})) \mathbf{x}_i - (1 - y_i) \sigma(\mathbf{x}_i^\top \boldsymbol{\beta}) \mathbf{x}_i \right] = -\sum_{i=1}^{N} (y_i - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta})) \mathbf{x}_i$$

Gradient of NLL

- Therefore, the gradient of the NLL with respect to $\boldsymbol{\beta}$ is:

$$\nabla_{\boldsymbol{\beta}} \text{NLL}(\boldsymbol{\beta}) = -\sum_{i=1}^{N} (y_i - \sigma(\mathbf{x}_i^\top \boldsymbol{\beta}))\mathbf{x}_i$$

Learning OR

```
1 import numpy as np
2 import statsmodels.api as sm
```

- Let's start by considering the complete dataset for the OR function:

```
1 X_or = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
2 y_or = np.array([0, 1, 1, 1])
```

- We can check whether the function is linearly separable by trying a simple linear regression:

```
1 X_or_with_intercept = sm.add_constant(X_or)
2 or_lm_fit = sm.OLS(y_or, X_or_with_intercept).fit()
3 or_pred = or_lm_fit.predict(X_or_with_intercept)
4 or_pred
```

```
array([0.25, 0.75, 0.75, 1.25])
```

- While not perfect, the linear regression does a decent job at separating the binary outputs, so we should be able to learn it using a single-layer perceptron (SLP).

Exercise 2: Logistic Regression

- Now, having implemented a simple OR function with negative log-likelihood loss, we can also implement a classic logistic regression with stochastic gradient descent.
- To make sure that all works as expected let's apply it to a synthetic dataset that we fully control the data-generating process.

```
rng = np.random.default_rng(seed = 123)
X = rng.normal(size = (1000, 2))

bias = np.array([-1.5])
weights = np.array([0.8, 3.2])
pi = 1/(1 + np.exp(-(X @ weights + bias))) # pi = sigmoid(X @ weights + bias)

y = np.random.binomial(n = 1, p = pi, size = 1000)
```

- If we were to apply a canned logistic regression implementation from e.g. `sklearn` or `statsmodels`, we would get the following estimates for the weights and bias:

```
1 X_with_intercept = sm.add_constant(X)
2 logit_fit = sm.Logit(y, X_with_intercept).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.300276
         Iterations 8
```

```
1 logit_fit.summary()
```

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	1000
Model:	Logit	**Df Residuals:**	997
Method:	MLE	**Df Model:**	2
Date:	Mon, 06 Apr 2026	**Pseudo R-squ.:**	0.5296
Time:	15:12:09	**Log-Likelihood:**	-300.28
converged:	True	**LL-Null:**	-638.35
Covariance Type:	nonrobust	**LLR p-value:**	1.506e-147

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**

const	-1.6208	0.134	-12.071	0.000	-1.884	-1.358
x1	0.7629	0.112	6.805	0.000	0.543	0.983
x2	3.2941	0.219	15.020	0.000	2.864	3.724