Data Analysis and Machine Learning: Logistic Regression

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

Sep 20, 2018

Logistic Regression

So far we have focused on learning from datasets for which there is a **continuous** output. In linear regression we have been concerned with learning the coefficients of a polynomial to predict the response of a continuous variable y_i on unseen data based on its independent variables $\mathbf{x}_i$.

Classification problems, however, are concerned with outcomes taking the form of discrete variables (i.e. categories). For example, we may want to detect if there's a cat or a dog in an image. Or given a specific system, we'd like to identify its state, say whether it is an ordered or disordered system (typical situation in solid state physics). (e.g. ordered/disordered).

Logistic regression deals with binary, dichotomous outcomes (e.g. True or False, Success or Failure, etc.). It is worth noting that logistic regression is also commonly used in modern supervised Deep Learning models, as we will see later.

Basics

We consider the case where the dependent variables $y_i \in \mathbb{Z}$ are discrete and only take values from $m = 0, \ldots, M-1$ (i.e. M classes).

The goal is to predict the output classes from the design matrix $X \in \mathbb{R}^{n \times p}$ made of n samples, each of which bears p features. The primary goal is to identify the classes to which new unseen samples belong.

Linear classifier

Let us start by considering a slightly simpler classifier: a linear classifier that categorizes examples using a weighted linear-combination of the features and an additive offset

$$s_i = \boldsymbol{x}_i^T \boldsymbol{w} + b_0 \equiv \mathbf{x}_i^T \mathbf{w}, \tag{1}$$

where we use the short-hand notation $\mathbf{x}_i = (1, \boldsymbol{x}_i)$ and $\mathbf{w}_i = (b_0, \boldsymbol{w}_i)$.

Some selected properties

This function takes values on the entire real axis. In the case of logistic regression, however, the labels y_i are discrete variables. One simple way to get a discrete output is to have sign functions that map the output of a linear regressor to $\{0, 1\}$, $f(s_i) = sign(s_i) = 1$ if $s_i \geq 0$ and 0 if otherwise. Indeed, this is commonly known as the "perceptron" in the machine learning literature. This model is extremely simple, and it is favorable in many cases (e.g. noisy data) to have a "soft" classifier that outputs the probability of a given category. For example, given $\mathbf{x}_i$, the classifier outputs the probability of being in category m. One such function is the logistic (or sigmoid) function:

$$f(s) = \frac{1}{1 + \mathrm{e}^{-s}}. \tag{2}$$

Note that $1 - f(s) = f(-s)$, which will be useful shortly.

The cross-entropy as a cost function for logistic regression

The perceptron is an example of a "hard classification": each datapoint is deterministically assigned to a category (i.e $y_i = 0$ or $y_i = 1$). In many cases, it is favorable to have a "soft" classifier that outputs the probability of a given category rather than a single value. For example, given $\mathbf{x}_i$, the classifier outputs the probability of being in category m. Logistic regression is the most canonical example of a soft classifier. In logistic regression, the probability that a data point $\boldsymbol{x}_i$ belongs to a category $y_i = \{0, 1\}$ is is given by

$$\begin{aligned} P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}) &= \frac{1}{1 + \mathrm{e}^{-\mathbf{x}_i^T \mathbf{w}}}, \\ P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{\theta}) &= 1 - P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}), \end{aligned} \tag{3}$$

where $\boldsymbol{\theta} = \mathbf{w}$ are the weights we wish to learn from the data.

Notice that in terms of the logistic function, we can write

$$P(y_i = 1) = f(\mathbf{x}_i^T \mathbf{w}) = 1 - P(y_i = 0).$$

Maximum likelihood

We now define the cost function for logistic regression using Maximum Likelihood Estimation (MLE). Recall, that in MLE we choose parameters to maximize the probability of seeing the observed data. Consider a dataset $\mathcal{D} = \{(y_i, \boldsymbol{x}_i)\}$ with binary labels $y_i \in \{0, 1\}$ where the data points are drawn independently. The likelihood of the seeing the data under our model is just:

$$P(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^{n} \left[f(\mathbf{x}_i^T \mathbf{w})\right]^{y_i} \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right]^{1-y_i} \tag{4}$$

from which we can readily compute the log-likelihood:

$$l(\mathbf{w}) = \sum_{i=1}^{n} y_i \log f(\mathbf{x}_i^T \mathbf{w}) + (1 - y_i) \log \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right]. \tag{5}$$

The maximum likelihood estimator is defined as the set of parameters that maximize the log-likelihood where we maximize with respect to θ

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} y_i \log f(\mathbf{x}_i^T \mathbf{w}) + (1 - y_i) \log \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right].$$

Since the cost (error) function is just the negative log-likelihood, for logistic regression we have that

$$\begin{aligned} \mathcal{C}(\mathbf{w}) &= -l(\mathbf{w}) \\ &= \sum_{i=1}^{n} -y_i \log f(\mathbf{x}_i^T \mathbf{w}) - (1 - y_i) \log \left[1 - f(\mathbf{x}_i^T \mathbf{w})\right]. \end{aligned} \tag{6}$$

This equation is known in statistics as the *cross entropy*. Finally, we note that just as in linear regression, in practice we usually supplement the cross-entropy with additional regularization terms, usually L_1 and L_2 regularization as we did for Ridge and Lasso regression.

Minimizing the cross entropy

The cross entropy is a convex function of the weights $\mathbf{w}$ and, therefore, any local minimizer is a global minimizer. Minimizing this cost function leads to the following equation

$$\mathbf{0} = \nabla \mathcal{C}(\mathbf{w}) = \sum_{i=1}^{n} \left[f(\mathbf{x}_i^T \mathbf{w}) - y_i\right] \mathbf{x}_i, \tag{7}$$

where we made use of the logistic function identity $\partial_z f(z) = f(z)[1 - f(z)]$. This equation defines a transcendental equation for $\mathbf{w}$, the solution of which, unlike linear regression, cannot be written in a closed form. Here we need gradient descent methods!

A scikit-learn example

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
['data', 'target_names', 'feature_names', 'target', 'DESCR']
X = iris["data"][:, 3:] # petal width
y = (iris["target"] == 2).astype(np.int) # 1 if Iris-Virginica, else 0

from sklearn.linear_model import LogisticRegression
```

```
log_reg = LogisticRegression()
log_reg.fit(X, y)

X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris-Virginica")
plt.show()
```