2EL1730: MACHINE LEARNING

CENTRALESUPÉLEC

Lab 1: Logistic Regression

Instructor: Nora Ouzir and Maria Vakalopoulou
TAs: Theodore Aouad, Hakim Benkirane, Aaron Mamann, Lily Monnier

November 21, 2022

1 Description

In this lab, we will study supervised learning algorithms. Initially, we will discuss *Ridge Regression*, and then we will see the basic characteristics of *Logistic Regression*, and then we will examine how the algorithm can be applied on real classification problems.

2 Ridge Regression

Let $\{y_i, X_i\}_{i=1}^m$ denotes a set of m observations, where each X_i is an n-dimensional vector. In *Ridge Regression*, a regularization term is added in the linear regression model in order to penalize the model complexity, leading to the following optimization problem:

$$\arg\min_{\theta} \|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \lambda\|\theta\|_2^2,$$

where $\lambda > 0$ is a regularization parameter.

(a) Find the closed form solution of the ridge regression problem.

(b) Calculate the Hessian matrix $\nabla_\theta^2 J(\theta)$, where $J(\theta)$ corresponds to the objective function.

3 Logistic Regression

Logistic regression is a discriminative classification model that can be used to predict the probability of occurrence of an event. It is a supervised learning algorithm that can be applied to binary or multinomial classification problems. Similar to the case of linear regression, each feature has a coefficient θ (i.e., a weight), that captures the contribution of the feature to the variable y (recall that in linear regression with only one feature x, we want to predict the value y of a new instance according to $y = \theta_1 x + \theta_0$; the goal is to learn parameters θ_0 and θ_1 from the training data). In the case of two-class logistic regression, variable y does not take continues values (as in linear regression), but actually corresponds to the class values, namely 0 or 1.

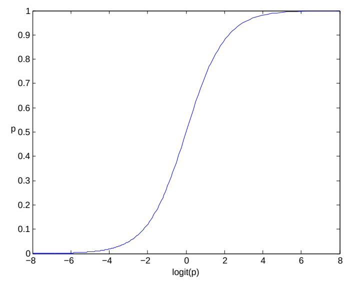

Figure 1: Logistic function.

That way, the probability that a new instance belongs to class 0 is given by: $p(Y = 0|x) = \theta_1 x + \theta_0$. In other words, the goal of logistic regression is to predict the probability that a new instance belongs to class 0 or 1. This is the main reason why the probability $p(x)$ cannot be considered as the variable y in a linear regression problem (the value should be in the $[0, 1]$ range).

To overcome this problem, we apply the *logistic transformation* of probability $p(x)$. In other words, we replace probability p with $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$[1]. While probability p can take values from 0 to 1, logit(p) ranges from $-\infty$ to $+\infty$. Figure 1 shows an example of the logistic function. Observe that the function is symmetric around 0.5, where the function takes value 0. In our case, the logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values between zero and one and hence is interpretable as a probability. That way, the logistic regression model can be expressed as:

$$\log \frac{p(x)}{1 - p(x)} = \theta_0 + \theta_1 x. \tag{1}$$

Solving for p we have:

$$p(x) = \frac{e^{\theta_0 + \theta_1 x}}{1 + e^{\theta_0 + \theta_1 x}} = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}. \tag{2}$$

In the case of a binary classification problem (i.e., two classes), we should predict class $Y = 1$ with probability $p \geq 0.5$ and class $Y = 0$ with probability $p < 0.5$. This means guessing class 1 whenever $\theta_1 x + \theta_0$ is non-negative, and 0 otherwise. So logistic regression gives us a *linear classifier*. The decision boundary separating the two predicted classes is the solution of $\theta_1 x + \theta_0 = 0$, which is a point if x is one dimensional, a line if it is two dimensional, a plane in the case of three features, etc. Logistic regression not only describes where the boundary between the classes is, but also indicates (via Eq. (2)) that the class probabilities depend on the distance from the boundary in a particular way, and that they go towards the extremes (0 and 1) more rapidly when $\|\theta\|$ is larger.

In the case where we have more than one features, the logistic regression model can be expressed as follows:

$$\log \frac{p(x)}{1 - p(x)} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n. \tag{3}$$

Solving for p we have:

$$p(x) = \frac{e^{\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n}}{1 + e^{\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n}} = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n)}}. \tag{4}$$

Logistic regression learns the weights θ_i so as to maximize the likelihood of the data. The likelihood function can be expressed in terms of the Bernoulli distribution:

$$p(Y|\theta) = \prod_{i=1}^{m} p(x_i)^{y_i} (1 - p(x_i))^{1 - y_i}, \tag{5}$$

where $p(x_i)$ the predicted probability that x_i belongs to the first class (as shown in Eq. 4), y_i the class label of instance i and m the number of instances. Based on this, we can define an error (or cost) function by taking the negative logarithm of the likelihood and applying the product rule for logarithms:

[1]Wikipedias lemma for *Logit*: `http://en.wikipedia.org/wiki/Logit`.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log\left(\frac{1}{1+e^{-(\theta_0+\theta_1 x_1^i+\theta_2 x_2^i+\ldots+\theta_n x_n^i)}}\right) + (1-y_i)\log\left(1-\frac{1}{1+e^{-(\theta_0+\theta_1 x_1^i+\theta_2 x_2^i+\ldots+\theta_n x_n^i)}}\right)\right]. \tag{6}$$

Thus, logistic regression aims to find parameters θ in order to minimize the above error (or cost) function. This can be achieved taking the gradient of the cost function with respect to parameters $\theta_0, \theta_1, \ldots, \theta_n$ and applying the gradient descent method:

$$\nabla J(\theta) = \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{1+e^{-(\theta_0+\theta_1 x_1^i+\theta_2 x_2^i+\ldots+\theta_n x_n^i)}} - y_i\right)x_j^i. \tag{7}$$

3.1 Pipeline of the Task

The goal of this part is to implement the logistic regression method and use it to find the decision boundary in a classification problem. The problem is to decide the probability that a student will be admitted to the Master's program of a university based on the grades of two courses.

The idea is that we already have available the grades for students of previous years, as well as the decision (accepted or not). This will be the training dataset for the logistic regression classifier. Figure 2 depicts the training data. The axes correspond to the two courses. The blue dots show the grades of the students that were admitted to the Master's program, while the red $\times$ to the students that were rejected. Next, we will apply logistic regression to design a model that predicts the acceptance probability for a student.

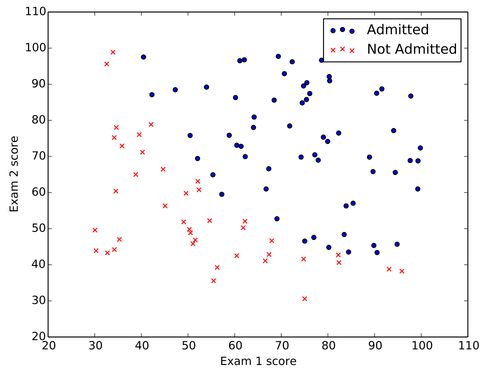

Figure 2: Training data.

The pipeline of the task is in the `logistic/main.py` script file. Initially, we load the data contained in the `data1.txt` file. The first two columns correspond to the two features of the dataset (grades on two courses), while the third one to the class label (admitted or not). We can also plot the data, as shown in Fig. 2.

```
# Load the dataset
# The first two columns contains the exam scores and the third column
# contains the label.
data = loadtxt('data1.txt', delimiter=',')

X = data[:, 0:2]
y = data[:, 2]
```

Then, we proceed with the main task. We initialize parameter θ with zero and call the `minimize()` optimization function of Python[2], that will minimize the value of the `computeCost()` function defined in Eq. (6), with respect to parameter θ. In the `minimize()` function, we also use the `computeGrad()` function, that computes the gradient of the cost function defined in Eq. (7). Also note that the `args` parameter contain the training data $\mathbf{X}$ and the corresponding class labels. The function returns the values of parameter θ.

```
# Initialize fitting parameters
initial_theta = zeros((3,1))
```

[2]`scipy.optimize.minimize` function: `http://goo.gl/HoM6Ib`

```
# Run minimize() to obtain the optimal theta
Result = op.minimize(fun = computeCost, x0 = initial_theta , args = (X, y), method = 'TNC',
jac = computeGrad);
theta = Result.x;
```

We can also plot the training data and the decision boundary produced by the logistic regression method.

```
# Plot the decision boundary
plot_x = array([min(X[:, 1]), max(X[:, 1])])
plot_y = (- 1.0 / theta[2]) * (theta[1] * plot_x + theta[0])
plt.plot(plot_x , plot_y)
plt.scatter(X[pos, 1], X[pos, 2], marker='o', c='b')
plt.scatter(X[neg, 1], X[neg, 2], marker='x', c='r')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend(['Decision Boundary', 'Not admitted', 'Admitted'])
plt.show()
```

Lastly, we perform the classification of the data, i.e., the prediction of the class labels. As we have described earlier, this can be done using the logistic function that returns a value in the range $[0, 1]$: if the predicted value of an instance is greater than 0.5 assign it to the first class (otherwise to the second one). This is implemented by the `predict()` function in the `predict.py` file.

```
# Compute accuracy on the training set
p = predict(array(theta), X)
counter = 0
for i in range(y.size):
if p[i] == y[i]:
counter += 1
print 'Train Accuracy: %f' % (counter / float(y.size) * 100.0)
```

3.2 Tasks to be Performed

The goal of the lab is to implement the basic components of the logistic regression classification algorithm:

- Fill in the code of the logistic function in the `sigmoid.py` file. Here we will use the following *logistic sigmoid* function[3]:

$$S(z) = \frac{1}{1 + e^{-z}}.$$

- Fill in the code of the `computeCost()` function in the `computeCost.py` file, based on the formula of Eq. (6). Note that, the terms within the logarithms of Eq. (6) correspond to the sigmoid (logistic) function of the dot product between the input data X and parameter θ.

- Fill in the code of the `computeGrad()` function in the `computeGrad.py` file, based on the formula of Eq. (7).

- Finally, fill in the code in the `predict.py` file to implement the `predict()` function. This can be done by applying the logistic (sigmoid) function to the test data (dot product between test data

[3]Wikipedia's lemma for `http://en.wikipedia.org/wiki/Sigmoid_function`.

X and parameters θ). Note that, the dataset used in the lab is not split into training and test sets; thus, you are encouraged to apply model evaluation techniques that we have seen in the class, and in particular the one of *cross-validation*[4]. *(For simplicity – but methodologically incorrect approach which can cause overfitting – you can also measure the accuracy on the training set).*

References

[1] Christopher M. Bishop. "Pattern Recognition and Machine Learning". Springer-Verlag New York, Inc., 2006.

[2] Tom M. Mitchell. "Machine learning". Burr Ridge, IL: McGraw Hill 45, 1997.

[4]Cross-validation in scikit-learn: `http://scikit-learn.org/stable/modules/cross_validation.html`