INF554 - MACHINE LEARNING I
ÉCOLE POLYTECHNIQUE
Lab 7: Regularization

November 12, 2018

1 Description

The goal of this lab is to study

- Feature engineering
- Regularized Logistic Regression
- Cross Validation

2 Regularized Logistic Regression

You will implement regularized logistic regression to predict whether microchips from a fabrication plant pass quality assurance (QA). During QA, each microchip goes through various tests to ensure it is functioning correctly. Assume that you are the product manager of the factory and you have the test results for some microchips on two different tests. From these two tests, you would like to determine whether the microchips should be accepted or not. To help you make the decision, you have a dataset of test results on past microchips, from which you can build a logistic regression model.

Figure 1 shows that our dataset cannot be separated into positive and negative examples by a straight-line. This means that a naive application of logistic regression will not perform well on this dataset, since it can only model a linear decision boundary.

One way to overcome this limitation is to construct features from polynomials of the original features, which transforms the input space into one that *is* linearly separable. Below, we provide a simple example where the features are mapping into all polynomial terms of x_1 and x_2 up to the sixth degree:

$$\mathbf{\Phi}\big(\boldsymbol{x} = (x_1, x_2)^\top, \mathsf{degree} = 6\big) = \big[1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, \ldots, x_1x_2^5, x_2^6\big] \tag{1}$$

As a result of this mapping, our feature vector has been transformed into a 28-dimensional vector. A logistic regression classifier trained on this higher-dimension feature vector will have a more complex (non-linear) decision boundary with respect to the original feature space.

However, despite the fact that the feature mapping allows us to build a more expressive classifier, it also more susceptible to overfitting. To avoid overfitting, we are going to implement the regularized logistic regression and we will see how regularization can help in this direction.

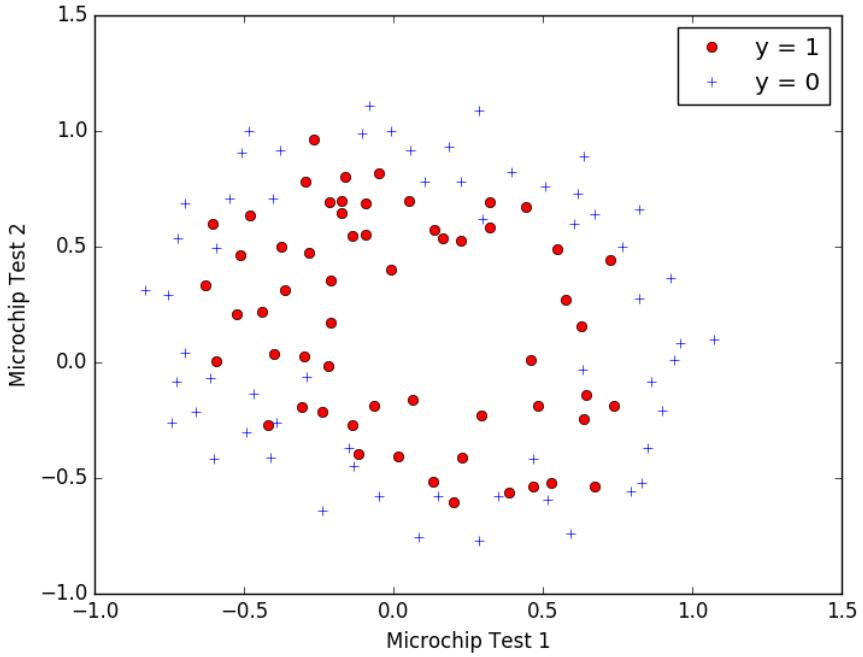

Figure 1: Microchip dataset

Recall that the logistic regression hypothesis is defined as follows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = g(\boldsymbol{\theta}^\top \boldsymbol{x}), \tag{2}$$

where g is the sigmoid function given as

$$g(z) = \frac{1}{1 + \exp(-z)} \tag{3}$$

For large positive values, the sigmoid should be close to 1, while for large negative values, the sigmoid should be close to 0.

Logistic regression learns the unknown parameters $\boldsymbol{\theta}$ so as to maximize the likelihood of the data. The likelihood function can be expressed in terms of the Bernoulli distribution:

$$p(Y|\boldsymbol{\theta}, X) = \prod_{i=1}^{m} p(\boldsymbol{x}^{(i)})^{y^{(i)}} (1 - p(\boldsymbol{x}^{(i)}))^{1-y^{(i)}}, \tag{4}$$

where $p(x_i)$ is the predicted probability that $\boldsymbol{x}^{(i)}$ belongs to the first class (as shown in Eq. 2), $y^{(i)}$ the class label of instance i, $X \in \mathbb{R}^{m \times n}$, n is the number of features and m is the number of instances. We define the error (cost) function as the negative log-likelihood:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \left(p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) + \left(1 - y^{(i)} \right) \log \left(1 - p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2. \tag{5}$$

Note that you should not regularize the intercept parameter θ_0. Thus, regularized logistic regression aims to find the unknown parameters $\boldsymbol{\theta}$ in order to minimize the above error (or cost) function (Eq. 5). This can be achieved taking the gradient of the cost function with respect to parameters $\boldsymbol{\theta}$ and applying

the gradient descent method thereafter. The gradient of the cost function is a vector where the j^{th} element is defined as follows:

$$\frac{\partial J(\boldsymbol{\theta})}{\theta_0} = \frac{1}{m}\sum_{i=1}^{m}\left(p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\boldsymbol{x}_j^{(i)} \quad \text{for} \quad j = 0$$
$$\frac{\partial J(\boldsymbol{\theta})}{\theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\boldsymbol{x}_j^{(i)} + \frac{\lambda}{m}\theta_j \quad \text{for} \quad j \geq 1 \tag{6}$$

2.1 Tasks to be Performed

1. Generate the feature matrix in `mapFeatures.py`
2. Extend your logistic regression implementation from Lab 2 to incorporate the L_2 penalty term.
3. Try out different regularization parameters λ and polynomial degrees for the microchip dataset to understand how regularization prevents over-fitting.
4. Build regularized logistic regression also on the second dataset: `bus_train.csv`.
5. Use 10-fold cross validation to choose the best λ parameter, and obtain results on the test data: `bus_test.csv`
6. Now implement your own stochastic gradient descent optimization (over iterations)
7. Plot the error rate vs iterations
8. Split a 20% validation set off from the training set and compare the coefficients/weights obtained with those under *early stopping*, wherein, once you stop: Record the error rate and then retrain the model on the full training set until obtaining that error rate.
9. Compare also against using no regularization at all.
10. Plot the ROC curves resulting from L_2-regularization strategy. Hint: an example with scikit-learn).
11. Bonus task: Use Keras to implement Logistic Regression, and use Dropout on the visible layer. Hint: `model.add(Dropout(λ, input_shape=(2,)))`. Experiment with the regularization parameter in the same way as earlier (i.e., using cross validation).