Computational Intelligence
Homework 1
Linear and Logistic Regression

Anand Subramoney
Guillaume Bellec

Contents

General remarks

Your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed)
- The quality of your plots (Is everything clearly visible in the print-out? Are axes labeled? . . .)

Python hints:

- Your submission should run with Python 3.3+
- The installation guide for Python can be found on the course website: `https://www.spsc.tugraz.at/courses/computational-intelligence/python-and-scipy-stack-installation`

1 Linear Regression

1.1 Derivation of Regularized Linear Regression [3 points + 1* point]

Given the design matrix $\boldsymbol{X}$ (m rows, $n+1$ columns), output vector $\boldsymbol{y}$ (m rows) and parameter vector $\boldsymbol{\theta}$ ($n+1$ rows), the mean squared error cost function for linear regression can be written compactly as,

$$J(\boldsymbol{\theta}) = \frac{1}{m}\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 \ . \tag{1}$$

The notation $\|\cdot\|$ refers to the Euclidean norm. The optimal parameters which minimize this cost function are given by,

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{2}$$

Consider the following slightly extended, "regularized" cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|^2 + \frac{\lambda}{m}\|\boldsymbol{\theta}\|^2 \tag{3}$$

The additional term penalizes parameters of large magnitudes. The constant $\lambda \geq 0$ controls the influence of this penalization (low $\lambda \rightarrow$ weak regularization, high $\lambda \rightarrow$ strong regularization). Such regularized cost functions are often used for linear regression (and other learning models) instead of (1) because regularization drastically reduces the effects of over-fitting that occur when the number of features is large.
Answer the following questions:

- Preliminary questions,
  - why is the design matrix $\boldsymbol{X}$ containing $n+1$ columns and not just n ? (see lecture on linear regression)
  - considering the function $J(\boldsymbol{\theta})$ as a function of all the variables $\theta_0, \theta_1 \ldots$, what it the definition of the gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$? This gradient is actually a vector, what is its dimension ?
  - What is the definition of the Jacobian matrix and what is the difference between the gradient and the Jacobian matrix ?
  - the hypothesis $\boldsymbol{X}\boldsymbol{\theta}$ generates predictions of the output vector $\boldsymbol{y}$, each coordinate of the vector corresponds to a different data point. What is the dimension of the vector $\boldsymbol{X}\boldsymbol{\theta}$? We also use the notation $\frac{\partial \boldsymbol{X}\boldsymbol{\theta}}{\partial \boldsymbol{\theta}}$ to define the Jacobian matrix. What is the dimension of the Jacobian matrix $\frac{\partial \boldsymbol{X}\boldsymbol{\theta}}{\partial \boldsymbol{\theta}}$? In this particular case, what is $\frac{\partial \boldsymbol{X}\boldsymbol{\theta}}{\partial \boldsymbol{\theta}}$ equal to ?
- Using the following hints, show that the optimal parameters which minimize the regularized linear regression cost (3) are given by,

$$\boldsymbol{\theta}^* = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \ , \tag{4}$$

  where $\boldsymbol{I}$ denotes the identity matrix of dimension $(n+1) \times (n+1)$.
- (Optional) In practice the matrix $\boldsymbol{X}^T\boldsymbol{X}$ might not be invertible. Using the spectral theorem (eigenvalue decomposition of symmetric matrices), what can we say about the eigenvalues of $\boldsymbol{X}^T\boldsymbol{X}$? Explain why the matrix $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is invertible for any small positive λ.

Some hints:

- **Hint 1** There are three ways to compute the gradient of a function $f(\theta)$. (1) compute each component individually $\frac{\partial f}{\partial \theta_j}$, (2) use the matrix notation and definition of the gradient as the unique vector v that verifies $f(\theta + d\theta) - f(\theta) = v^T d\theta + o(d\theta)$ or (3) get used to manipulating the derivative of matrix operators directly with care; the wikipedia page on matrix calculus contains a useful summary of rules regarding matrix/vector derivatives (numerator layout).

- **Hint 2** This is the derivation of the solution of the Linear Regression without regularization when using matrix operators:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{m} \frac{\partial (\boldsymbol{X\theta} - \boldsymbol{y})^T (\boldsymbol{X\theta} - \boldsymbol{y})}{\partial \boldsymbol{\theta}} \tag{5}$$

$$= \frac{2}{m} (\boldsymbol{X\theta} - \boldsymbol{y})^T \cdot \frac{\partial (\boldsymbol{X\theta} - \boldsymbol{y})}{\partial \boldsymbol{\theta}} \tag{6}$$

$$= \frac{2}{m} (\boldsymbol{X\theta} - \boldsymbol{y})^T \cdot \boldsymbol{X} \tag{7}$$

When the cost function is minimized the gradient of the cost function is zero:

$$\boldsymbol{X}^T (\boldsymbol{X\theta} - \boldsymbol{y}) = \boldsymbol{0} \tag{8}$$

$$\boldsymbol{X}^T \boldsymbol{X\theta} - \boldsymbol{X}^T \boldsymbol{y} = \boldsymbol{0} \tag{9}$$

$$\boldsymbol{X}^T \boldsymbol{X\theta} = \boldsymbol{X}^T \boldsymbol{y} \tag{10}$$

$$\theta = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y} \tag{11}$$

As long as $\boldsymbol{X}^T \boldsymbol{X}$ is invertible the solution θ exists and is unique.

For questions 1.2 and 1.3 download the associated zip file from the course website and unzip it. The file `data_linreg.json` contains training, validation and test sets of a regression problem with a single input dimension.

1.2 Linear Regression with polynomial features [4 points]

Your task is to fit polynomials of different degrees to the training data, and to identify the degree which gives the best predictions on the validation set (performing simple model selection). The features that should be used for linear regression are (x corresponds to the scalar input, and n is the polynomial degree)

$$\phi_0 = 1, \phi_1 = x, \phi_2 = x^2, \ldots, \phi_n = x^n, \tag{12}$$

Fill in the TODO boxes in the following places:

- The functions `design_matrix`, `train` and `compute_errors` in the file `poly.py`
- In `main_poly_model_selection.py`. Your code should find the degree of the polynomial that minimizes the error in either the training or validation set. You can re-use the code provided above.

The following should be included in your report:

- Plot your results for each of the following polynomial degrees: 1, 2, 5, 20. Use the provided function `plot_poly` in file `plot_poly.py` to create the plots.
- Report which degree $\in \{1, \ldots, 30\}$ gives the lowest training error. Plot the results for that degree. Report the test error.
- Report which degree $\in \{1, \ldots, 30\}$ gives the lowest validation error. Plot the results for that degree. Report the test error.
- Plot training, validation and testing errors as a function of the polynomial degree using the provided function `plot_errors` in file `plot_poly.py`.
- Discuss your findings in your own words using the concept of over-fitting. Why is it important to use a validation set?

1.3 Linear Regression with radial basis functions [4 points]

Using the same data set, your task is to fit hypotheses with different numbers of radial basis functions to the training data, and to perform model selection. Let l be the number of RBFs. For a given l, the RBF centers should be chosen to uniformly span the whole input range $[-1, 1]$. The width of the basis functions should be set to $\sigma = 2/l$, i.e. with a higher l, the RBFs should be narrower (implementing a finer resolution). The features used for linear regression should be (x corresponds to the scalar input and c_j is the center of RBF j)

$$\phi_0 = 1,\ \phi_1 = e^{\frac{-(x-c_1)^2}{2\sigma^2}},\ \ldots\ ,\ \phi_l = e^{\frac{-(x-c_l)^2}{2\sigma^2}} \tag{13}$$

Fill in the TODO boxes in the following places::

- The functions `get_centers_and_sigma`, `design_matrix`, `train` and `compute_errors` in the file `rbf.py`.
- In `main_rbf_model_selection.py`, to get training/validation/test errors for each $l \in \{1, 2, \ldots, 39, 40\}$. You can re-use the code provided in `main_poly_model_selection.py`.
- In the function `plot_errors` in the file `plot_rbf.py`. This makes a plot with the number of RBFs l on the horizontal axis and the training/validation/test error on the vertical axis. For each of the three errors, plot a separate line in a different color. You can base your implementation on the equivalent function in the file `plot_poly.py`.

The following should be included in your report:

- Using `main_rbf.py`, plot your results for each of the following degrees: 1, 2, 5, 20.
- Report which number of RBFs $\in \{1, \ldots, 40\}$ gives the lowest training error. Plot the results for that degree. Report the test error.
- Report which number of RBFs $\in \{1, \ldots, 40\}$ gives the lowest validation error. Plot the results for that degree. Report the test error.
- Plot training, validation and testing errors as a function of the degree with your adaptation of `plot_errors`.
- Briefly describe and discuss your findings in your own words. Is the polynomial or the RBF model better?

2 Logistic Regression

2.1 Derivation of Gradient [2 points]

The logistic regression hypothesis function is given by,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{\theta}) = \sigma(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n) \tag{14}$$

with the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$, parameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^T$, and input vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_n)^T$. By convention the constant feature x_0 is fixed to $x_0 = 1$. With $\boldsymbol{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \ldots, x_n^{(i)})^T$, $x_0^{(i)} = 1$ and $\log(\cdot)$ referring to the natural logarithm, the logistic regression cost function can be written as,

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)}\log(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) + (1-y^{(i)})\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\Big) \tag{15}$$

Derive the gradient of the cost function, i.e. show that the partial derivative of the cost function with respect to θ_j equals

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\Big)\cdot x_j^{(i)} \quad . \tag{16}$$

Hint: $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)\cdot(1-\sigma(z))$

2.2 Logistic Regression training with gradient descent and `scipy.optimize` [6 points + 2* points]

Download the associated zip file from the course website and unzip it. The file `data_logreg.json` contains training and test set of a binary classification problem with two input dimensions. Your task is to fit a logistic regression classifier with polynomial features of varying degrees to the data, and to compare different optimization techniques for this task. The optimization techniques that should be compared are gradient descent with a constant learning rate, gradient descent with an adaptive learning rate (optional), and the advanced optimization method implemented by the Scipy function `minimize`. The expansion of the input is already implemented. The features that should be used for logistic regression are all monomials of the two inputs x_1 and x_2 up to some degree l (monomials are polynomials with only one term), i.e. all products $(x_1)^a(x_2)^b$ with $a, b \in \mathbb{Z}$ and $a + b \leq l$. For example, for degree $l = 3$ the features should be,

$$\phi_0 = 1, \tag{17}$$

$$\phi_1 = (x_1)^1,\ \phi_2 = (x_2)^1, \tag{18}$$

$$\phi_3 = (x_1)^2,\ \phi_4 = (x_1)(x_2),\ \phi_5 = (x_2)^2, \tag{19}$$

$$\phi_6 = (x_1)^3,\ \phi_7 = (x_1)^2(x_2)^1,\ \phi_8 = (x_1)^1(x_2)^2,\ \phi_9 = (x_2)^3 \tag{20}$$

2.2.1 Gradient descent (GD) [5 points]

Fill the following blocks in the code you are provided:

- In `logreg.py` implement the cost funtion of logistic regression as the python function `cost`
- In `logreg.py` implement the gradient of the cost with respect to theta as the python function `grad`
- In `gradient_descent.py` implement a generic gradient descent solver

Your report should include the following:

1. The function `check_gradient` in `toolbox.py` is here to test if your gradient is well computed. Explain what it is doing.
2. For degree $l = 1$ run GD for 20 and 2000 iterations (learning rate $\eta = 1$, all three parameters initialized at zero). Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain why the number of iterations should be neither too low nor too high.
3. For degree $l = 2$ run GD for 200 iterations and learning rates of $\eta = .15$, $\eta = 1.5$ and $\eta = 15.$. Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain what is happening when η is too large or too small.
4. Identify reasonably good pairs of values for the number of iterations and learning rate for each degree in $l \in \{1, 2, 5, 15\}$ that provides a good solution in reasonable time. Report these pairs, the performance obtained on training and testing set and plot each solution. Conclude by describing which degree seems the most approriate to fit the given data.
5. Describe a possible stopping criterium using the gradient of the cost function ?

2.2.2 Adaptative gradient descent (GDad) [2* points optionals]

Fill the following blocks in the code you are provided:

- In `gradient_descent.py` implement the adaptive learning rate as follows: After every update check whether the cost increases or decreases.
  - If the cost increased, **reject** the update (go back to the previous parameter setting) and multiply the learning rate by 0.7.
  - If the cost decreased, accept the update and multiply the learning rate by 1.03.

  The iteration count should be increased after every iteration even if the update was rejected.

Your report should include the following:

1. Run GDad for varying degrees $l \in \{1, 2, 5, 15\}$ (with zero initialization of parameters, 1000 iterations, and initial learning rate $\eta = 1$). Report training and test errors, final learning rates and plot the hypothesis obtained in each case with function `plot_logreg` in file `logreg_toolbox.py`.
2. For each degree compare with the non-adaptive gradient descent variant. In particular, is the evolution of the learning rate numerically coherent with your previous guess of the optimal learning rate ? Discuss why GDad can be useful.

2.2.3 Scipy optimizer [1 point]

In `main_logreg.py` the VARIANT 3 is commented out. Un-comment the code and compare the results to the previous implementations.

1. Report training and test errors, and plot the hypothesis obtained for the four degrees $l \in \{1, 2, 5, 15\}$ with function `plot_logreg`.
2. Compare the results with those above (GD/GDad) and briefly discuss and interpret your observations. In particular does it change your opinion on which degree is best to fit the data?