# Computational Intelligence SS20
# Homework 2
# Linear and Logistic Regression

Ceca Kraisnikovic
Horst Petschenig

| | |
|---|---|
| Tutor: | Flora Feldner, flora.feldner@student.tugraz.at |
| Points to achieve: | 20 pts |
| Bonus points: | 2* pts |
| Info hour: | 05.05.2020 Cisco WebEx Meeting, see TeachCenter |
| Deadline: | 12.05.2020 23:55 |
| Hand-in procedure: | Use the **cover sheet** to be found on the teach center. Submit your **python files and a colored report** on the teach center. |
| Course info: | TeachCenter |

## 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? . . . )
- Your submission should run with Python 3.5+

# 1 Linear Regression

## 1.1 Derivation of Regularized Linear Regression [5 points]

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,

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

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:

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

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 occurs when the number of features is large.
**Answer the following questions:**

1. Preliminary questions,
   (a) Why is the design matrix $\boldsymbol{X}$ containing $n+1$ columns and not just $n$?
   (b) Considering the function $J(\boldsymbol{\theta})$ as a function of all the variables $\theta_0, \theta_1 \ldots$, give one definition of the gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$? This gradient is actually a vector, what is its dimension?
   (c) What is the definition of the Jacobian matrix and what is the difference between the gradient and the Jacobian matrix?
   (d) The hypothesis $\boldsymbol{X\theta}$ generates predictions of the output vector $\boldsymbol{y}$, each coordinate of the vector corresponds to a different data point. We also use the notation $\frac{\partial \boldsymbol{X\theta}}{\partial \boldsymbol{\theta}}$ to define the Jacobian matrix of the vector $\boldsymbol{X\theta}$. What is the dimension of the Jacobian matrix $\frac{\partial \boldsymbol{X\theta}}{\partial \boldsymbol{\theta}}$ and what is it equal to?

2. Using the hints below or as done in the practical 4, 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 $(n+1) \times (n+1)$.

**Some hints for question 1.1.2:**

1. **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).

2. **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 $\theta$ exists and is unique.

## 1.2 Linear Regression with polynomial features [6 points]

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.

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 cost function in either the training or validation set. You can re-use the code provided above.

**The following should be included in your report:**

1. Using `plot_poly` in `main_poly.py`, plot your results for each of the following degrees: 1, 5, 10, 22.
2. Report which degree $\in \{1, \ldots, 30\}$ gives the lowest cost function on the training set. Plot the results for that degree. Report the cost on the testing set.
3. Report which degree $\in \{1, \ldots, 30\}$ gives the lowest cost function on the validation set. Plot the results for that degree. Report the cost on the testing set.
4. Plot training, validation and testing costs as a function of the polynomial degree using the provided function `plot_errors` in file `plot_poly.py`.
5. Discuss your findings in your own words using the concept of over-fitting. Why is it important to use a validation set?

## 1.3 (Bonus) Linear Regression with radial basis functions [2* 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 costs for each $l \in \{1, 2, \dots, 39, 40\}$. 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 costs on the vertical axis. For each of the three cost values, plot a separate line in a different colour. You can base your implementation on the equivalent function in the file `plot_poly.py`.

**The following should be included in your report:**

1. Using `plot_rbf` in `main_rbf.py`, plot your results for each of the following degrees: 1, 5, 10, 22.
2. Report which number of RBFs $\in \{1, \dots, 40\}$ gives the lowest cost function on the training set. Plot the results for that degree. Report the cost function on the testing set.
3. Report which number of RBFs $\in \{1, \dots, 40\}$ gives the lowest cost function on the validation set. Plot the results for that degree. Report the cost function on the test error.
4. Plot training, validation and testing costs as a function of the degree with your adaptation of `plot_errors`.
5. 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 [3 points]

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

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\Big(y^{(i)}\log(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) + (1-y^{(i)})\log(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))\Big) \tag{14}$$

**Answer the following question:**

1. Derive the gradient of the cost function, i.e. show that the partial derivative of the cost function with respect to $\theta_j$ equals

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\Big)\cdot x_j^{(i)} \quad . \tag{15}$$

**Hint** Note that the derivative of the sigmoid function $\sigma$ verifies $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)\cdot(1-\sigma(z))$.

### 2.2 Logistic Regression training with gradient descent [6 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. 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,

$$\phi_0 = 1, \tag{16}$$
$$\phi_1 = (x_1)^1,\ \phi_2 = (x_2)^1, \tag{17}$$
$$\phi_3 = (x_1)^2,\ \phi_4 = (x_1)(x_2),\ \phi_5 = (x_2)^2, \tag{18}$$
$$\phi_6 = (x_1)^3,\ \phi_7 = (x_1)^2(x_2)^1,\ \phi_8 = (x_1)^1(x_2)^2,\ \phi_9 = (x_2)^3 \tag{19}$$

**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 $\eta = 0.1$, $\eta = 2$ and $\eta = 20$. Report training and test errors for each iteration number and plot the decision boundaries. Comment on the results and explain what is happening when $\eta$ 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 $l \in \{1, 2, 7, 20\}$ 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.