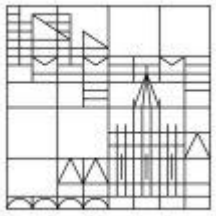
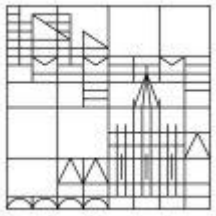


Machine Learning using Matlab

Logistic regression and regularization

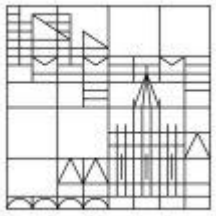
Joko Triloka, Ph.D

Outline



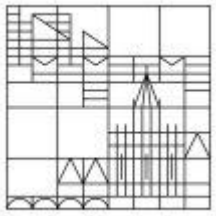
- Gradient descent for logistic regression
- Advanced optimization algorithms
- Polynomial model
- Options on addressing overfitting
- Regularized linear regression and logistic regression
- Multiclass classification (one-vs-all)

Logistic regression



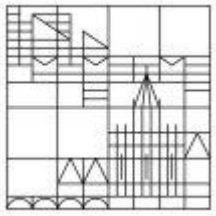
- Classification model
- Start from binary: $y \in \{0,1\}$
- E.g., tumor: malignant (1)/benign(0)

Logistic regression



- Hypothesis: $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
- Parameter: θ
- Cost Function: $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta(x^{(i)})\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta(x^{(i)})\right)\right)$
- Goal: $\min_\theta J(\theta)$

Gradient descent



- Given cost function $J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)})))$, our objective is $\min_\theta J(\theta)$
- Repeat{

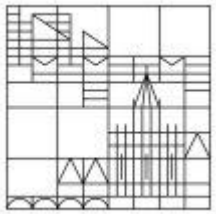
$$\theta_j:=\theta_j-\alpha\frac{\partial}{\partial\theta_j}J(\theta)\quad j=0,1,\ldots,n$$ (simultaneously update all θ_j)

$$\Rightarrow\quad \theta_j:=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$$

}

Identical to linear regression except the hypothesis is different!

Advanced optimization

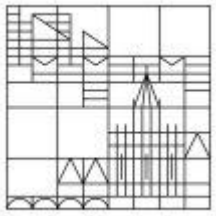


- Optimization algorithm: you have cost function, your objective is to minimize it, i.e., $\min_{\theta} J(\theta)$
- Solution: given parameter θ, we have code that can compute:
  - Cost function $J(\theta)$
  - Partial derivative $\frac{\partial}{\partial \theta_j} J(\theta) \quad j=0,1,\ldots,n$
- Gradient descent
  - Repeat{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad j=0,1,\ldots,n$$

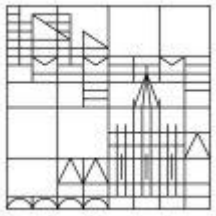
    }

Advanced optimization



- Other advanced algorithms:
  - Conjugate gradient
  - BFGS
  - L-BFGS
  - ...
- Advantages
  - No need to pick learning rate manually
  - Often faster than gradient descent
- Disadvantages:
  - More complex to implement
- Implementation is out of scope in the course, but you can still use them in Matlab!

Implementation in Matlab



1. Write your own cost function:

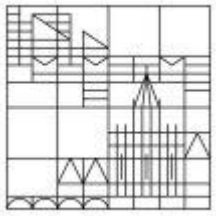
   ```
   function [jVal, gradient] = costFunction(theta)
         jVal = [...code to compute J(theta)...];
         gradient = [...code to compute derivative of J(theta)...];
   end
   ```

2. Use the function fminunc():

   ```
   options = optimset('GradObj', 'on', 'MaxIter', 100);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
   options);
   ```

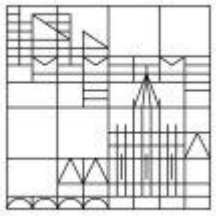
3. Example of linear regression

Nonlinear



- What is “nonlinear”?
  - The change of the output is not proportional to the change of the input.
- Nonlinear function:
  - Polynomial, Gaussian,...

Features and polynomial regression



- Go back to linear regression with one variable:

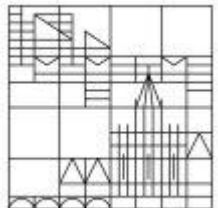
$$h_\theta(x)=\theta_0+\theta_1 x$$

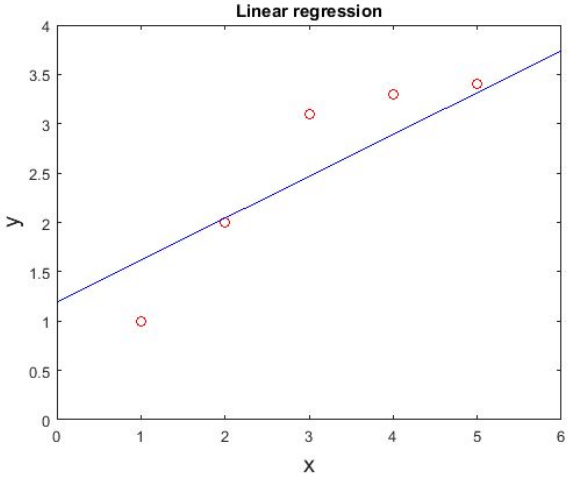
- To improve model, we can add more “artificial” features:

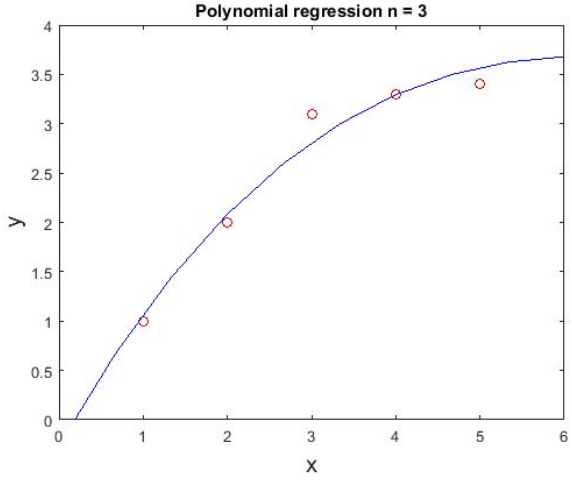
$$h_\theta(x)=\theta_0+\theta_1 x+\theta_2 x^2+\ldots+\theta_n x^n=\sum_{i=0}^{n}\theta_i x^i$$

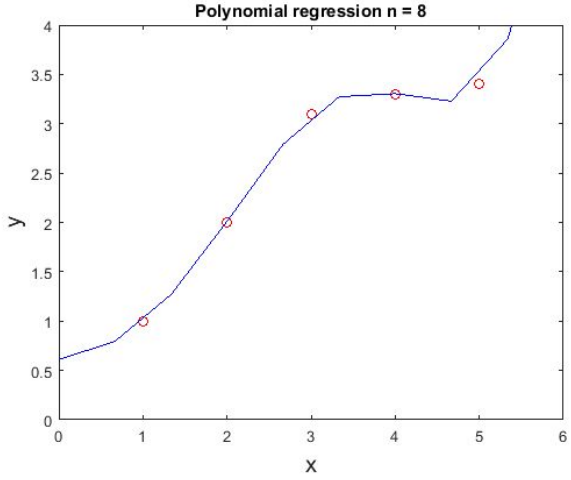
- The hypothesis becomes a polynomial function, therefore, we call it “polynomial regression” instead
- Question: the more parameters, the better?

Example: linear/polynomial regression







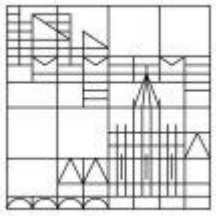


Underfitting, high bias

Just right

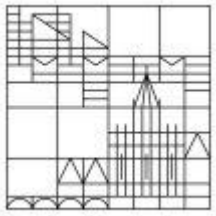
Overfitting, high variance

Underfitting and Overfitting



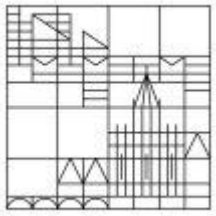
- Underfitting (high bias)
  - A phenomenon that the ML model maps poorly to the trend of the data
  - Occurs when your hypothesis is too simple or use too few features
- Overfitting (high variance)
  - A phenomenon that the ML Model fit the training set very well, but fail to generalize to test set. In other words, an overfitting model performs well in training set, but quit bad in test set.
  - Occurs when your hypothesis is excessively complex, e.g, too many parameters or too many features

Address Underfitting and Overfitting



- Underfitting
  - Add more features
  - Use a more complex hypothesis
- Overfitting
  - Reduce number of features
    - Manually select which features to keep (still questionable)
    - Model selection algorithm (later in this course)
  - Regularization
    - Keep all the features, but reduce values of parameters θ_j
    - Regularization works well when we have lots of slightly useful features

Regularization



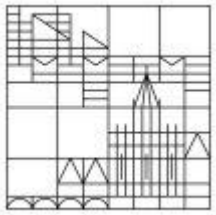
- Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$
  - “Simpler” hypothesis, thus less prone to overfitting
- Regularization for linear regression

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

Note j start from 1 not 0!

- Here lambda is the regularization parameter, control the tradeoff between two different goals: fitting the training set well and keeping the parameter small
  - What would happen if lambda is too small?
  - What would happen if lambda is too large?
  - How to tune it? (Later in this course)

Regularized linear regression



Gradient descent:

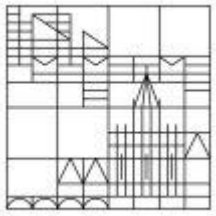
- Repeat until convergence{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad j=0,1,\ldots,n$$

}

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

Regularized linear regression



Gradient descent:

- Repeat until convergence{

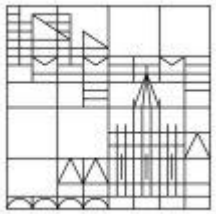
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad j = 1, 2, \ldots, n$$

}

Additional term here

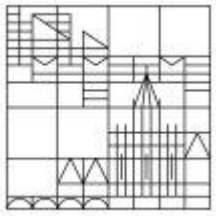
Regularized linear regression



- Normal equation

$$\theta=(X^TX+\lambda\cdot L)^{-1}X^Ty$$

Regularized logistic regression



- No regularization:

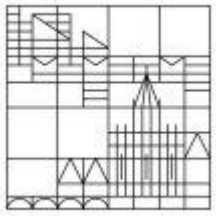
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta(x^{(i)})\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta(x^{(i)})\right)\right)$$

- With regularization:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta(x^{(i)})\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta(x^{(i)})\right)\right) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Regularization term

Regularized logistic regression



Gradient descent:

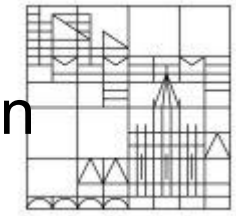
- Repeat until convergence{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad j = 1, 2, \ldots, n$$

}

Advanced optimization for regularized logistic regression



- function [jVal, gradient] = costFunction(theta, x, y)

  jVal = [...code to compute J(theta)...];

  $$J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)})))+\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

  gradient = [...code to compute derivative of J(theta)...];

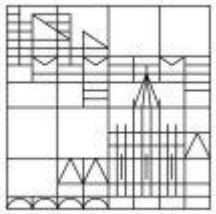
  $$\frac{\partial J(\theta)}{\partial \theta_0}=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)}$$

  $$\frac{\partial J(\theta)}{\partial \theta_j}=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}+\frac{\lambda}{m}\theta_j \quad j=1,2,\ldots,n$$

  end

- Feed costFunction and data into fminunc()

Multiclass Classification



- In the previous work we assume the labels in logistic regression were binary:

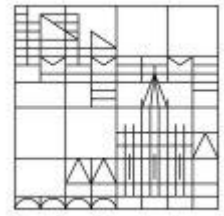
$$y \in \{0, 1\}$$

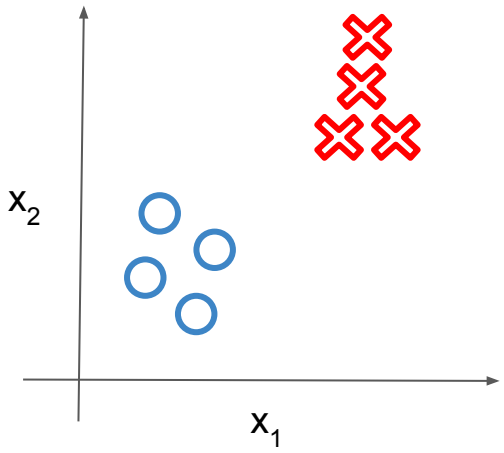
- In multiclass classification, we expand our definition so that:

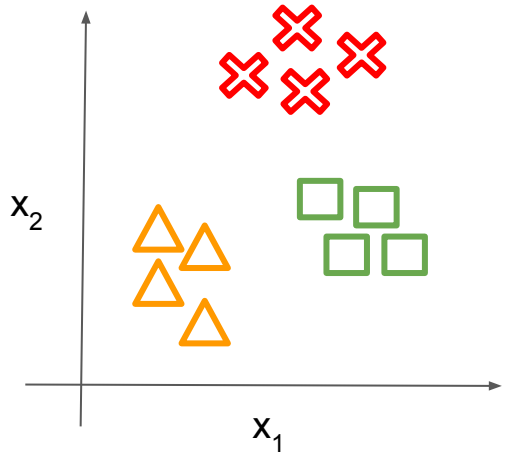
$$y \in \{1, 2, \ldots, K\}$$

- Example:
  - face recognition: attendance system
  - object categorization: human, car, face, …
  - Weather: sunny, cloudy, rain, snow
- It doesn’t matter what the index starts 0 or 1!

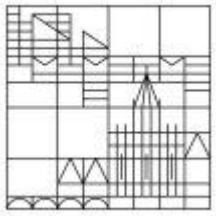
Binary vs. Multiclass





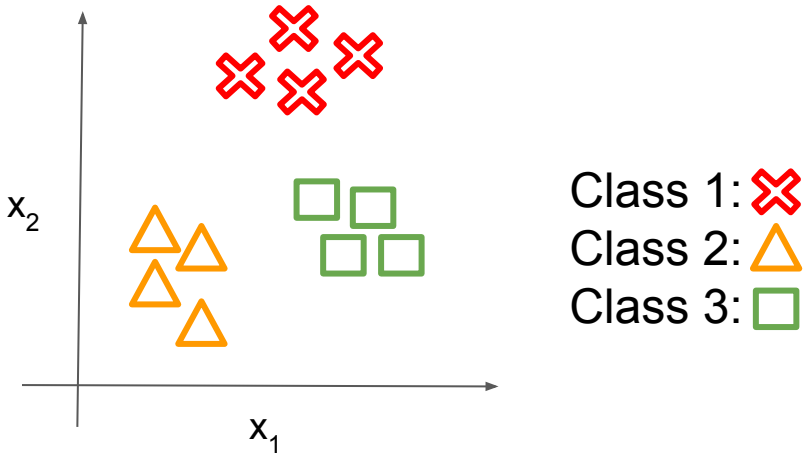


Multiclass classification

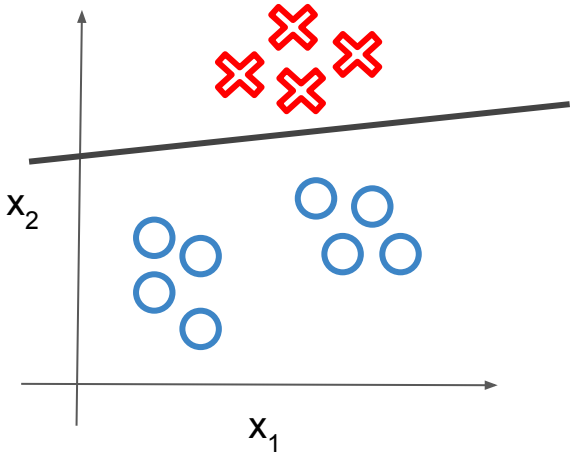


- Two solutions:
  - One-vs-all (one-vs-rest): we can divide multiclass classification problem to K binary classifier
  - Softmax regression (multinomial logistic regression): train a multiclass classifier

One-vs-all (one-vs-rest)

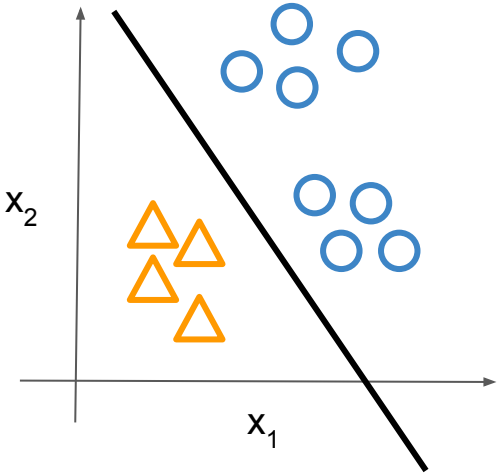


One-vs-all (one-vs-rest)



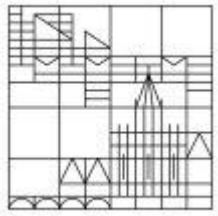
$$h_\theta^{(1)}(x)=P(y=1|x;\theta)$$

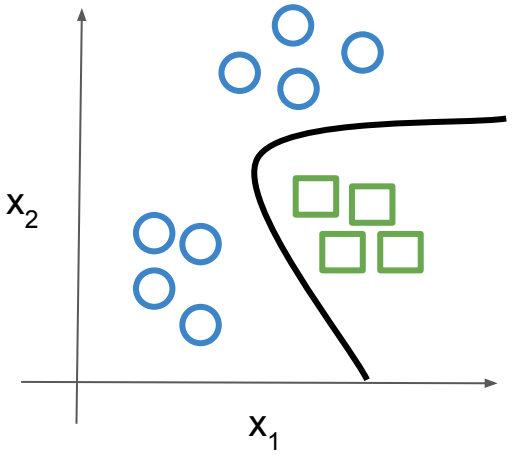
One-vs-all (one-vs-rest)



$$h_\theta^{(2)}(x)=P(y=2|x;\theta)$$

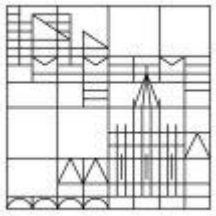
One-vs-all (one-vs-rest)





$$h_\theta^{(3)}(x)=P(y=3|x;\theta)$$

One-vs-all (one-vs-rest)



Summary:

- Train a logistic regression classifier for each class to predict the probability
- To make a prediction on new data, pick the class that has the maximum output
- Q: what if a new data doesn't belong to any class?

$$y \in \{1,2,\ldots,K\}$$

$$h_\theta^{(1)}(x) = P(y=1|x;\theta)$$

$$h_\theta^{(2)}(x) = P(y=2|x;\theta)$$

$$h_\theta^{(3)}(x) = P(y=3|x;\theta)$$

$$\ldots$$

$$h_\theta^{(K)}(x) = P(y=K|x;\theta)$$

$$prediction = \max_i \left(h_\theta^{(i)}(x)\right)$$