Table of Contents

[Supervised Learning 1](#_Toc29203360)

[2. Classification 1](#_Toc29203361)

[Linear Regression + Threshold 1](#_Toc29203362)

[Logistic Regression 1](#_Toc29203363)

[**Hypothesis Function** 1](#_Toc29203364)

[**Cost Function** 3](#_Toc29203365)

[**Gradient Descent & Advanced Optimization Algo** 4](#_Toc29203366)

[**Multiclass Classification: One-vs-all** 5](#_Toc29203367)

**Supervised Learning:** given the “right answer” for each example in the data

Regression: predict continuous outputs (e.g. housing prices)

Classification: predict discrete outputs (e.g. 0 benign, 1 malignant; 0 benign, 1 type I cancer, 2 type II cancer, 3 type III cancer)

**2. Classification**

## **Linear Regression + Threshold**

Linear regression may fit the data (pink), but not always (blue).

A close up of a map

Description automatically generated

## **Logistic Regression**

### **Hypothesis Function**

For binary classification, we want .

Take

Hypothesis function:

Interpret the output of hypothesis function:

= estimated probability that y is 1 given input x and parameters

=1

Sigmoid/ logistic function: output for



z

1

0

Sigmoid function outputs continuous values between 0 and 1:

We set a threshold value to classify outputs into two classes:

* predict y = 1 if threshold
* predict y = 0 if threshold

If we want to predict y = 1 only if very confident 🡪 set threshold = 0.7

If we want to avoid missing y = 1 🡪 set threshold = 0.3

(Classification is essentially regression with outputs classified by a threshold value).

Set threshold = 0.5, we predict y = 1 if 0.5

🡪

🡪

🡪 Predict y = 1 if

is called decision boundary

Decision boundary is the line that separates the area where y = 0 and y = 1.

Training data is used to fit parameters .

🡪 Parameters determine hypothesis function.

🡪 Hypothesis function defines decision boundary.

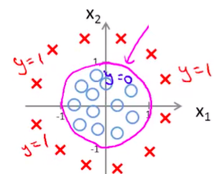
A close up of a piece of paper

Description automatically generated

e.g. Linear decision boundary

Predict y = 1 if

Decision boundary:



e.g. Non-linear decision boundary

Predict y = 1 if

Decision boundary:

### **Cost Function**

For linear regression:

Because of non-linear sigmoid function, squared cost is not a convex function. GD is not guaranteed to converge to the global minimum.

A picture containing object, clock

Description automatically generated

To make convex, we write cost function in this way:

|  |  |  |
| --- | --- | --- |
| If y = 1:   * = 1, cost = 0 * 🡪 0, cost 🡪 | If y = 0:   * = 0, cost = 0 * 🡪 1, cost 🡪 | “negative log function plot”的图片搜索结果 |

MLE

### **Gradient Descent & Advanced Optimization Algo**

To find that minimizes , we use the same gradient descent algorithm as in linear regression.

Repeat until convergence {

simultaneously update all for j = 0, ..., n

}

|  |
| --- |
| function [J, grad] = costFunction(theta, X, y)  % Compute cost and gradient of the cost w.r.t. to theta for logistic regression  m = length(y);  % using vectorization  % return cost  h = sigmoid(X \* theta)  J = -(1/m) \* (y' \* log(h) + (1-y)' \* log(1 - h));  % return gradient  grad = (1/m) \* X' \* (h - y);  end |

Advanced Optimization Algorithms: to optimize

* Gradient descent
* Conjugate gradient
* BFGS
* L-BFGS

Three advanced algorithms vs. Gradient descent:

+ No need to manually pick learning rate

+ No need to write loops yourself, only provide a function to compute cost and gradient

+ Faster convergence

-- More complex

|  |
| --- |
| % Use a built-in function (fminunc) to find the optimal parameters theta.    % Set options for fminunc  % Set 'MaxIter' = 400  % Set 'GradObj' to 'on': tells fminunc that our function returns both cost and gradient  % This allows fminunc to use the gradient when minimizing the function.  options = optimset('GradObj', 'on', 'MaxIter', 400);  initial\_theta = zeros(size(X, 2), 1);  % Run fminunc to obtain the optimal theta  % This function will return theta and the cost  % '@(t)(costFunction(t, X, y))' creates a function, with parameter t, which calls costFunction  [theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial\_theta, options); |

### **Multiclass Classification: One-vs-all**

Fit a logistic regression classifier for each class i to predict the probability that y = i

For a new input x, pick the class i that

A map with text

Description automatically generated

|  |
| --- |
| function [all\_theta] = oneVsAll(X, y, num\_labels, lambda)  % Trains num\_labels logistic regression classifiers  % y is a vector of labels from 1 to 10 (10 represents 0)  % When training classifier for class k, we want to return a m-diemnsional vector of labels y, where yi = 1 or 0 indicated if a training example belongs to class k or not. (use logical array)  m = size(X, 1);  n = size(X, 2);  % Add ones to the X data matrix  X = [ones(m, 1) X];  % Returns all the classifiers in a matrix "all\_theta",  % where i-th row corresponds to the classifier for label i  all\_theta = zeros(num\_labels, n + 1);  % loop over the different classes  for c = 1:num\_labels  initial\_theta = zeros(n + 1, 1);  % Use fmincg to optimize the cost function  % fmincg works similarly to fminunc, but is more efficient with large number of parameters.  options = optimset('GradObj', 'on', 'MaxIter', 50);  % Run fmincg to obtain the optimal theta  % y == c returns a vector of 1's and 0's indicating whether the ground truth is true/false for this class.  [theta] = fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)), initial\_theta, options);  % Copy column vector "theta" into matrix "all\_theta" as the i-th row  all\_theta(c,:) = theta';  end |

|  |
| --- |
| function p = predictOneVsAll(all\_theta, X)  % Return a vector of label for each example in the matrix X using a trained one-vs-all classifier.  % The labels are in the range 1...K, where K = size(all\_theta, 1).  % X contains the examples in rows. X is (m, n+1)  % all\_theta is a matrix (num\_labels, n + 1) where the i-th row is a trained logistic regression theta vector for the i-th class.  m = size(X, 1);  num\_labels = size(all\_theta, 1);  X = [ones(m, 1) X];  % return p: a m-dimensional vector of values from 1...K  % e.g. p = [1; 3; 1; 2] predicts classes 1, 3, 1, 2 for 4 examples  % Make predictions (m, num\_labels)  prediction = sigmoid(X \* all\_theta');  [value, indices] = max(prediction,[],2);  p = indices;  end |

|  |
| --- |
| [all\_theta] = oneVsAll(X, y, num\_labels, lambda);  pred = predictOneVsAll(all\_theta, X);  % Training set accuracy  mean(double(pred == y)) \* 100; |