Notes for Implementing Machine Learning Models

Yue Wang

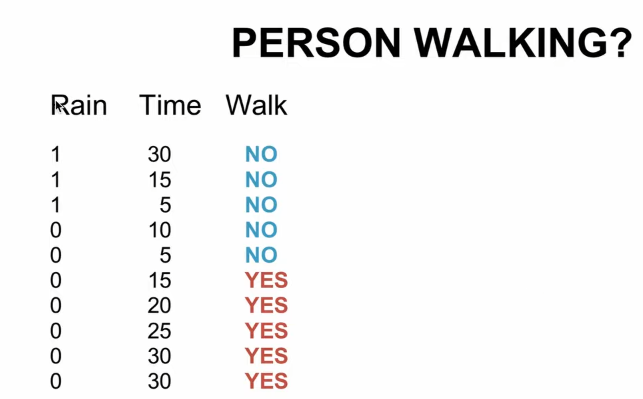
**Notes**:

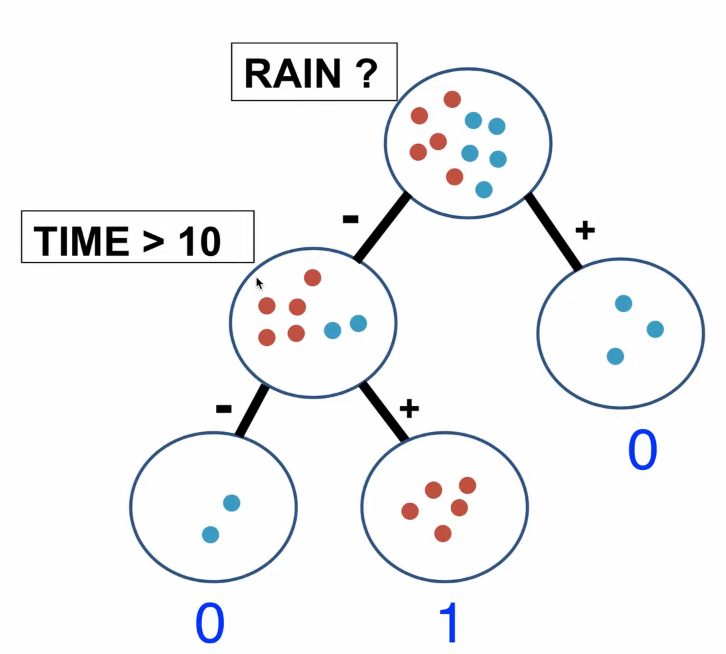
Use bold lower-case characters represent column vectors, for example:

The row vectors are represented as the transpose of a column vector, for example ***x***T=[*x*1, *x*2…,*xn*].

A matrix is represented by a bold upper case character.

# Decision trees





# Linear Regression

The number of examples is *m*, and the number of features is *n*. Each example is a column vector, i.e., the shape of the *i*th example is (*n*, 1),

or represented as ***x*** for short,

Weights ***w*** has the shape (*n*, 1)

Loss function:

The average loss of the m examples (mean sum of error):

Gradient descent:

where,

Vectorization

Use ***X*** to represent all learning examples:

The shape of ***X*** is (*m*, *n*), *m* is number of examples, and *n* is the number of features. The vectorized form of ***y*** is:

And gradient of ***b*** is similar.

Pay attention how the vectorization of gradient descent is done in Python codes. (Note that the equation/codes of linear regression and logistic regression are the same, perceptron has the similar form)

Text

Description automatically generated

# Logistic regression

The notation is very similar to the linear regression.

One modification is made in this implementation. We add 1 to each example so that we do not have to add the *b*.

Loss function:

The cost function *J* is the average loss of the m examples (mean sum of error):

Gradient descent:

For a single sample:

For gradient descent of a batch (averaged gradient):

Vectorization

Use ***X*** to represent all learning examples:

The shape of ***X*** is (*m*, *n*), *m* is number of examples, and *n* is the number of features. The vectorized form of ***y*** is:

Gradient descent:

where,

Text

Description automatically generated

(Note that the equation/codes of linear regression and logistic regression are the same, perceptron has the similar form)

# Perceptron

Perceptron can only be used for linearly separatable data sets. Label of y is either 0 or 1.

Activation function *g*(*z*) is a step function:

The equation for updating weight *w*, is the same to logistic regression.

Use the updating rule above for all the samples that are predicted wrong. When all samples are correctly predicted, stop.

(Note that the equation/codes of linear regression and logistic regression are the same, perceptron has the similar form)

# Principal component analysis (PCA) and singular value decomposition (SVD)

Use ***X*** to represent all learning examples:

The shape of ***X*** is (*m*, *n*), *m* is number of examples, and *n* is the number of features. The vectorized form

PCA steps:

Step 1: compute mean of *X*.

Step 2: subtract mean from *X*. Note the examples with mean subtracted as *B*.

Step 3: compute covariance matrix of rows of B. the covariance is noted as *C*. Since *B* is the matrix with mean subtracted, we can use the following equation.

The covariance can also be computed using np.cov(*B*T). This function assumes that each column is an example, so we transpose *B*.

Step 4: compute eigen values and eigen vectors of *C*.

Method 1 of computing the eigen values/vectors of C: compute eigen vectors directly, e.g., np.linalg.eig(*C*)

Method 2 of computing the eigen values/vectors of C: use SVD.

dimensions: *C* is *n*×*n*, *U* is *n*×*n*, *V*T is *m*×*n*. Note that *σ*1≥ *σ*2≥…>0, and ***u****i* is the eigen vector corresponds to *σi*.

Step 5: keep the first *r* columns of *U*, note as *U*reduce, . And the compressed data, *Z* is:

To reconstruct the compressed Z back to the original dimension:

# Adaboost

Create an ensemble classifier .

Differentiate weights *αt* and *wi*: *αt* is the voting weights for the weak learner *ht*, *wi* is the weights in training that related to the ith example.

The number of examples is *n*. Iterate *T* times, i.e., ensemble *T* weak learners.

|  |  |  |  |
| --- | --- | --- | --- |
| **Adaboost Algorithm** | | | |
|  | *wi* = 0 (*i* = 0, 1, 2, …, *n*) | | |
|  | **for** *t*=1:*T* **do** | | |
|  |  | Fit a weak learner *ht* (e.g., stump) to the training set using weights ***w***. | |
|  |  | **if** *ε* >0.5 **then** | |
|  |  |  | Flip the weak learner’s, i.e., *ht*, prediction (e.g., if *ht* predict 1 when ***x*** > threshold, now *ht* will predict -1 when *x* > threshold) |
|  |  | **end** |  |
|  |  | compute weighted error:    save *ht* and *αt* in a list  , (*i* = 0, 1, 2, …, *n*)  renormalize | |
|  | **end** |  |  |

In line 3, a weak learner *ht* (e.g., stump) is fit for the training set using weights ***w***. There are two ways to do this:

1. Fit a weak learner *ht* (e.g., stump) for *X*sample sampled from *X* (with replacement) with probability being selected equals to *wi*. The shape of *X*sample and *X* are the same. This corresponds to parameter method = “resample” in function fit of class Adaboost.
2. Try all features and threshold and compute the cost weighted by ***w***. This corresponds to parameter method = “exhaustive” in function fit of class Adaboost.
3. Use weighted Gini index or entropy. (This is not implemented).