**Summary**

Machine learning is the science of getting computers to learn, without being explicitly programmed.

|  |  |  |
| --- | --- | --- |
| Supervised  learning |  |  |
| Linear  regression | Step 1. Hypothesis:  Step 2. Cost | A close up of a map  Description automatically generated |
|  | Step 3: Gradients | A close up of a map  Description automatically generated |
| Logistic  regression | Step 1. Hypothesis:  Step 2. Cost | A picture containing food  Description automatically generated |
|  | Step 3. Gradients: |  |
| Support  vector  machines  (SVM) | Cost | Related image |
| SVM with  Gaussian  Kernel | Cost | A picture containing text, map  Description automatically generated |

|  |  |  |
| --- | --- | --- |
| Unsupervised  learning |  |  |
| K-means | Centroids  denotes the index of cluster centroids closet to x(i) | K=3  A close up of a map  Description automatically generated |
|  | Means  denotes the average(mean) of points assigned to cluster k |
|  | cost |
| Principal  Component  Analysis  (PCA) | **Feature scaling (Mean normalization)**  **Mean:**  **Standard deviation:**  **Mean normalize:** | PC Scatterplot |
| Dimensionality  Reduction  Data compression | **Calculate** U, S, V.    U, S, V = numpy.linalg.svd(sigma)  Ureduce = U[:, 0:K].T  Z = Ureduce\*X = X\_norm \* U[:, 0:K]  X\_approximate = X\_recovered = Z \* U[:, 0:K].T |
|  | Pick the smallest value of k,  99% of variance retained |
| Anomaly  detection | **Gaussian (Normal) distribution**  **Mean:**  **Variance:**  **Probability:** | A close up of a map  Description automatically generated |
|  | 1. Original model   Density estimation:  Training set:  Each example is:  Probability: | A close up of a piece of paper  Description automatically generated |
|  | 1. Multivariate Gaussian   Mean:  **Variance:**  **Diagonal Sigma:**  **Probability:** | A close up of text on a white background  Description automatically generated |

# Supervised learning

Already know what our correct output should look like, having the idea that there is a relationship between the input and the output

## Linear regression

Regression, continuous, individual

1. Data & Hypothesis

Input data: X (m, n); dataset: ; features: ;

parameters: (n,1); actual output: y(m,1).

, ,

Hypothesis:

1. Cost function:

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's.

Minimize cost:

1. Gradients

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to:

Rate: , regulation:

Repeat until convergence: {

}

 will always be less than 1. Intuitively you can see it as reducing the value of  by some amount on every update.

A close up of a map

Description automatically generated

Figure 1 Example of batch gradient descent

## Logistic regression

**Classification, Discrete, collective**

1. Sigmoid function

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

The following image shows us what the sigmoid function looks like:

A picture containing kitchen, white, light, large

Description automatically generated

So if our input to g is *θTX*, then that means:

From these statements we can now say:

1. Cost function:

Minimize cost:

1. Gradients descent:

when computing the equation, we should continuously update the two following equations:

Repeat: {

}

## Support vector machine (SVM)

SVM gives a cleaner, and more powerful way of learning complex non-linear functions.

A picture containing drawing

Description automatically generated

Figure 2 SVM Kernel mapping

### No kernel (“linear kernel”)

Predict “y = 1” if

1. Hypothesis:
2. Cost function:

A picture containing object, clock

Description automatically generated

Large C: Lower bias, high variance.

Small C: Higher bias, low variance.

### SVM with kernels, also called gaussian kernel

#### Hypothesis

*Given x, compute features , parameters*

*Predict “y=1” if*

1. Training

*min*

|  |  |
| --- | --- |
| Large : Features vary more smoothly.  Higher bias, lower variance. | A picture containing clock  Description automatically generated |
| Small : Features vary less smoothly.  Lower bias, higher variance. | A close up of a logo  Description automatically generated |

1. Multiclass classification

Many SVM packages already have built-in multiclass classification functionality.

Otherwise, use one-vs.-all method. (Train SVMs, one to distinguish from the rest, for ), get Pick class with largest

A screenshot of a cell phone

Description automatically generated

## Neural Network

Non-linear Classification

|  |  |
| --- | --- |
| A picture containing building, window, drawing  Description automatically generated | no. of units (not counting bias unit) in layer |

|  |  |
| --- | --- |
| Binary classification  1 output unit | Multi-class classification (K classes)  E.g.  K output units |

* Training a neural network (e.g. L=4)
* Pick a network architecture (connectivity pattern between neurons)
* No. of input units: Dimension of features
* No. output units: Number of classes
* Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

1. Randomly initialize weights

Initialize parameters

Initialize each to a random value in (i.e. )

epsilon = 0.12

theta = np.random.rand(input\_layer, output\_layer) \* 2\*epsilon - epsilon

1. Implement forward propagation to get for any

|  |  |  |  |
| --- | --- | --- | --- |
| A picture containing building, window, drawing  Description automatically generated | | | |
| Input layer | Hidden layers | | Output layer |
|  |  |  |  |

1. Implement code to compute cost function
2. Implement backpropagation to compute partial derivatives

“error” of node in layer .

|  |  |  |  |
| --- | --- | --- | --- |
| A picture containing building, window, drawing  Description automatically generated | | | |
| Input layer | Hidden layers | | Output layer |
|  |  |  |  |

Perform forward propagation and backpropagation using example   
(Get activations and delta terms for ).

* Set (from all, ).
* Set
* Perform forward propagation to compute for
* Using y, compute
* Compute

|  |  |
| --- | --- |
|  |  |
|  |

1. Use gradient checking to compare computed using backpropagation vs. using numerical estimate of gradient of .
2. Implement backpropagation to compute DELTA VECTOR (unrolled ).
3. Implement numerical gradient check to compute **gradient approximation**.
4. Make sure they give similar values.

diff = slin.norm(numgrad-grad)/slin.norm(numgrad+grad)

print('If your backpropagation implementation is correct, then \n\

the relative difference will be small (less than 1e-9). \n\

\nRelative Difference: ', diff)

1. Turn off gradient checking. Using backprop code for learning.

* **Important:** Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of cost function your code will be very slow).

1. Use gradient descent or advanced optimization method with backpropagation to try to minimize as a function of parameters .

result = minimize(cost\_func, initial\_nn\_params, method='CG', jac=grad\_func,

options={'disp': True, 'maxiter': 50.0})

nn\_params = result.x

Jcost = result.fun

# Unsupervised learning

Fund hidden pattern in unlabeled data with little or no idea what our results should look like.

Don't necessarily know the effect of the variables.

## K-means

1. Choose the value of K

Sometimes, you’re running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

A close up of text on a white background

Description automatically generated

Figure 3 T-shirt size

1. Initialize centroids

Random initialize K clustering centroids

For i = 1 to 100{

Randomly pick examples from given points as K () different centroids

initial\_centroids = random.sample(X.tolist(), K)

cost function

*Return centroids of the smallest J.*

**}**

1. Iteration

Set Iterate times, max\_iters

For iters = 1 to max\_iters {

denotes the index of cluster centroids closet to .

denotes the average(mean) of points assigned to cluster k.

}

A close up of a map

Description automatically generated

Figure 4 K clusters

## Principal component analysis (PCA)

**Dimensionality Reduction**

**Motivation:**

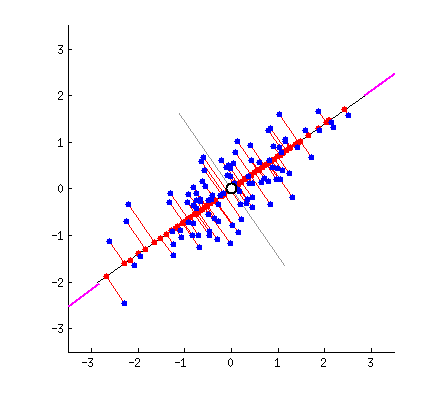


Figure 5 2 dimensions to 1 dimension

1. **Data processing**

Training set:

Processing: feature scaling (mean normalization) to ensure every feature has zero mean.

1. Mean value:
2. Standard deviation
3. Replace each
4. PCA algorithm.

Reduce data from n-dimensions to k-dimensions

Principal Component Analysis (PCA) algorithm:

U, S, V = numpy.linalg.svd(sigma)

Dimension matrix:

Ureduce = U[:, 0:K].T

Reduced K-dimensions data:

Z = Ureduce \* X = X\_norm \* U[:, 0:K]

Recover data to n-dimentions:

X\_approximate = X\_recovered = Z \* U[:, 0:K].T

1. **Choosing K** (number of principal components)**:**

Pick the smallest value of K.

**Check if**

**99% of variance retained.**

**A screenshot of a cell phone

Description automatically generated**

## Anomaly detection

### Original model

1. Choose feature:

Training set:

Density estimation: **1, 2, …, n**

Choose features that might be indicative of anomalous examples.

1. Fit parameters:
2. Given new example , compute

Flag an anomaly if

### Multivariate Gaussian

Don’t model etc. separately. Model all in one go.

1. Choose feature

Training set:

Density estimation: **1, 2, …, n**

1. Fit parameters

Parameters: (covariance matrix)

1. Given new example , compute

Flag an anomaly if

* The importance of real-number evaluation

1. When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labeled data, of anomalous and non-anomalous examples. ( if normal, if anomalous).

Training set 60%: (assume normal examples/not anomalous)

Cross validation set 20%:

Test set 20%:

1. Algorithm evaluation

Possible evaluation metrics

Precision

Recall

F1-Score

**Comparison between these two models**

**A screenshot of a cell phone

Description automatically generated**

### Collaborative filtering

Recommender system

1. Problem formulation

if user has rated movie (0 otherwise)

rating by user on movie (if defined)

= parameter vector for user

= feature vector for movie   
For user , movie, predicted rating:

= no. of movies rated by user

1. Initialization

Initialize and to small random values.

1. Cost function
2. Minimize and

Min

1. using gradient descent (or an advanced optimization algorithm). E.g. for every :

Gradients

1. Prediction.

For a user with parameters and a movie with(learned) features , predict a star rating of

is the mean of rated users for movie .

# Tips

## Model Selection and Train/Validation/Test Sets

One way to break down our dataset into the three sets is:

1. Training set: 60%
2. Cross validation set: 20%
3. Test set: 20%

We can now calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross-validation set.
3. Estimate the generalization error using the test set with Jtest(Θ(d)), (d = theta from polynomial with lower error);

This way, the degree of the polynomial d has not been trained using the test set.

## Gaussian (Normal) distribution

A close up of a map

Description automatically generated

Figure 6 68% of the data is within 1 standard deviation, 95% is within 2 standard deviation, 99.7% is within 3 standard deviations

## F1-score

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Actual class | |
|  |  | 1/+ | 0/- |
| Predicted class | 1/+ | True positive (TP) | False positive (FP) |
| 0/- | false negative (FN) | True negative (TN) |

**Precision**

**Recall**

**F1-score**

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

Plot matrix

plt.imshow(Y, aspect='equal', origin='upper', extent=(0, Y.shape[1], 0, Y.shape[0]/2.0))