Machine Learning

Week 1

Introduction

- A computer program learn from experience E with respect to some task T and some performance measure P, if its performance on T as measured by P improves with experience E
- Supervised learning
  - In the dataset we are told the right answer of each data
  - Regression - predict value
  - Classification
- Unsupervised learning
  - Given date with no labels
  - Clustering

Linear Regression with One Variable

Model Representation

- Training data ((x1, y1), (x2, y2), . . .)-> learning algorithm -> hypothesis
- A linear regression:

Cost Function

- thetas: parameters
- Different parameters give different hypothesis
- Come up with paramters that fit the data well - choose paramters so that h(x) is close to y for training example (x, y)
- One cost function:
- Goal:
- Intuition: sum of distance between hypothesis and the actual values

Gradient Descent

- Idea: start with some initial paramters; keep changing them to reduce cost
- Algorithm: repeat until convergence
- Derivates: tangent of the point in the function J
- Alpha: learning rate - how large the step it takes
  - Too small: the algorithm can be small
  - Too large: overshoot the minimum, may never converge

  - As we approach local minimum, gradient descent will automatically take smaller step; so no need to decrease alpha
- Simultaneously update parameters
- Apply gradient descent to cost function
- Batch gradient descent: use all training examples in each step

Week 2

Multiple Features (Multivariate Linear Regression)

- Hypothesis:

- x0 is equal to 1

Gradient Descent

- Cost function:
- Algorithm:
- Simultaneously update all parameters
- Have the same form as gradient descent with single variable

Feature Scaling

- Make sure features are on a similar scale
- It will take a long time for gradient descent to converge if they are not
- Get every feature into approximately [-1, 1]
- Can also use mean normalization
  - replace xi with (xi - mean) / [(max - min) | stdev]

Learning Rate

- Making sure that gradient descent work correctly
  - Plot cost function versus number of iterations
  - The value should decrease after each iteration
  - Use automatic convergence test: declare convergence after the decrease is smaller than a threshold
- If it is not working correctly
  - Use small learning rate
  - But a small learning rate will make convergence slow
- In summary
  - If learning rate is small, slow convergence

  - If learning rate is large, may not converge
  - Try a range of learning rate in practice

Features and Polynominal Regression

- Sometimes choosing different features yield a better model
- Polynominal regression
  - For example, suppose we want to fit a model:
- We can turn it to a linear regression by having:
- Features scaling becomes very important
- By choosing different features you can get different model

Normal Equation

- Method for solving theta analytically
- Intuition:
- Optimal Solution
- No need to do feature scaling
- Comparison with gradient descent
  - Gradient descent
    - Need to choose learning rate
    - Need many iterations
    - Works well even when n (number of features) is large
  - Normal equation
    - No need to choose learning rate
    - Don't need iterations
    - Need to compute (X'X)^-1 - O(n^3)
    - Slow if n is large
    - Doesn't work for some problems

Normal Equation and Noninvertibility

- What if X'X is non-invertible
  - May have redundant features
  - Too many features
    - Delete some features or use regularization
- pinv (pseudo inverse) in Octave will calculate correctly even X is not invertible

Week 3

Logistic Regression

- Classification problmes
  - Classify into "0" (negative class) or "1" (positive class)
  - Or multi class problem (classify into "0", "1", "2", ...)
- Can use linear regression
  - Threshold classifier: > 0.5, predict "1", otherwise predict "0"
  - Not a good idea
    - Often it is right beacause you are lucky (e.g., specific training data)
    - Output value can be > 1 or < 0 while we want output to be 0 or 1

Hypothesis Representation

- Want prediction to be between 0 and 1
- Use logistic (sigmoid) function
- Output can be interpreted as the probability that y = 1 given x, parameterized by theta

Desicion Boundary

- Predict "1" if output >= 0.5, else predict "0"
- If the underlying regression outputs >= 0, the output will be >= 0.5
- Fit a line that separated the region where the hypothesis predicts 1 and that predicts 0
  - The line is desicion booundary
- Nonlinear decision boundaries
  - Fit a polynominal desicion boundary

Cost Function

- Take the cost function from linear regression
- This will give us a non-convex function (lots of local optima), because hypothesis uses logistic function
- Cost function for logistic regression
- Capture the intuition that if y = h, cost is 0. If h = 0 and y = 1, the cost is infinity and vice versa.

Simplified Cost Function and Gradient Descent

- A simpler cost function
- Can be derived from statistics using the principle of maximum liklihood
- Convex function
- To fit the parameters, minimize the cost function to get a set of parameters
  - Use gradient descent, the algorithm looks idential to linear regression
  - Feature scaling also applies to logistic regression

Advanced Optimization

- Optimization algorithms
  - Conjugate gradient
  - BFGS
  - L-BFGS
- These algorithms
  - No need to pick up a learning rate
  - Often much faster to converge
  - More complex

Multiclass Classification: One-vs-all

- One-vs-all
  - Suppose we have three classes
  - Turn the training data into three separate binary classification problems
  - Train three classifiers
  - Pick the class i that maximizes h

The Problem of Overfitting

- Underfit (high bias)
  - Algorithm not fitting the data well
- Overfit (high variance)
  - If we have too many features, the learned hypothesis may fit the training set very well, but fail to generalize to new examples
- Addressing overfitting
  - Reduce number of features
    - Manually select features
    - Model selection to automatically select features

        * It also throws away useful information

    - Regularization
        * Keep all features, but reduce magnitude/values of parameter
        * Works well if we have lots of features and each of them contributes a bit

Cost Function

- If we have small values of parameters
    - We have a simpler hypothesis
    - Less prone to overfitting
- Take cost function and shrink all parameters
- By convention, not penalizing theta0
- Lambda is a regularization parameter
    - Control the tradeoff between fitting the training set well and keeping parameters small
    - Too large lambda will penalize parameters too much, thus causing underfitting

Regularized Linear Regression

- Gradient descent
    - Regualarization term shrinks theta a little for each iteration
    - For j = 0, the update rule doesn't change
    - For j = 1, 2, 3, ...
- Normal Equation
    - Non-invertibility
        * Suppose number of training data <= number of features. If lambda > 0, the matrix will be invertable

Regularized Logistic Regression

- Add the regularization term to the cost function
    - Same update rule for linear regression

Week 4: Neural Networks: Representation

Non-linear Hypothesis

- Many features

  - if include quadratic features, there are too many features and maybe overfitting
  - if not, not enough features to fit the data set
- Simple logistic regression with quadratic features added in is not a good way to learn complex hypothesis

Neurons and Brain

- Origin: mimic brain
- Popularity: 80s, early 90s, diminished later
- Resurgence: more powerful computers

Model Representation

- Neuron unit: logistic unit
  - Feed in some inputs, the neuron does some computation and outputs
  - A sigmoid(logistic) activation function
- Network: neuron unit wired together
  - Layer 1: input layer
  - Layer 2: hidden layer, can have many
  - Layer 3: output layer
- Forward propagation
- Neural networks learn its own features
  - Like logistic regression
  - Use the computed features a insteand of the original feature x
  - Hidden layer computed more complex features
- Multi-class Classification
  - Suppose we have four classes
  - Have four output units
  - Want [1 0 0 0] for class 1, [0 1 0 0] for class 2, and so on
  - For training set, represent as [1 0 0 0] and so on

Week 5 - Neural Networks: Learning

Cost Function

Backpropagation Algorithm

- Want to minimize cost function
- Need to compute cost function and gradient
- Intuition: calculate error of node j in layer l
- Backpropagation algorithm

```
For traning examples 1 to m
    Set a(1) = x(i)
    Perform forward propagation to compute a(l) for l = 2 to L
    Using y(i) to compute delta(L) = a(L) - y(i)
    Compute delta(L - 1) to delta(2)
```

- Formally delta is partial derivative of cost over z
- delta of *(i - 1)* is calculated by deltas from i weighted by the parameters

Gradient Checking

- Calculate a approximate value of the derivative and compare with gradient

```
for 1 = 1:n
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaPlus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2 * EPSILON);
end;
% Compare with gradient
```

- Turn off gradient checking once the code is verified to be correct

Random Initialization

- Initial value of 0 does not work for neural networks
  - All activiation ouputs are the same, the errors will also be same
  - Gradient will be the same
  - After each update, parameters corresponding to inputs going into each group of two hidden units are identical
- Initial to be value between -epsilon and epsilon

Putting it together

- Pick a network architecture
  - Number of input units: dimension of input x
  - Number of output units: number of classes
  - Reasonable default: 1 hidden layer, or if >1 hiddent layer, use same number of hidden units in each layer (usually the more the better)
- Training network
  - Random initialization
  - Implement forward propagation
  - Implement cost function
  - Implement back propagation to get derivatives

  - Use gradient checking, then disable it
  - Use gradient descent or other advanced algorithm to minimize cost function
    * Cost function is non-convext, so will probably stuck in local minimium

Week 7: Support Vector Machine

Optimization Goals

- Cost function for SVM
- C is 1 / lambda
- For cost 1, cost is 0 if z >= 1; for cost 0, cost is 0 if z <= -1
- Output 1 or 0, not probability

Large Margin Intuition

- If y = 1, we want z >= 1
- If y = 0, we want z <= -1
- Suppose C is very large, then we want the first term in cost function to be small
  - Choose theta such that when y = 1, z >= 1, z <= -1 when y = 0
  - Solve this, we will get a decision boudary that separates datasets at the maximum margin
  - This is sensitive to outliers
  - If we make C small, it is less sensitive to outliers

Kernels

- We have the formula theta0 + theta1 * f1 + theta2 * f2 + ...
- Predict 1 if the above formula >= 0
- We choose some landmarks and have fi = similarity(x, li)
  - Similarity function is the kernel
  - For example, we can use a gaussian kernel: exp(-||xi - li||^2 / (2 * sigma^2))
    * sigma controls the "width"
  - If xi is close to li, fi is close to 1
  - If xi is far away from li, fi is close to 0
- Choosing the landmarks
  - For every training example, choose a landmark at the exact same location
- For every training example
  - Have f1 = sim(xi, l1), f2 = sim(xi, l2), f3 = sim(xi, l3), ...

  - Put these f into a vector, f0 = 1
  - Predict 1 if theta' * f >= 0
- Use some computational tricks
  - Kernel is not used for other methods because the trick doesn't work well
- Large C, high variance
- Small C, low variance
- Large sigma, features fi vary more smoothly. Higher bias, lower variance
- Small sigma, features fi vary less smoothly. Lower bias, higher variance

Using SVM

- Use software library
- Have to choose
  - Parameters
  - Choice of kernel
    - No kernel: theta0 + theta1 * x1 + ... + thetan * xn >= 0 If n is large and m is small, you can use no kernel
    - Gaussian kernel
      - Need to choose sigma
      - If n is small and m is large
      - Do perform feature scaling
    - Not all similarity functions are valid kernel
      - Must satisfy Mercer's Theorem
- Multi-class SVM
  - Use built-in functions in packages
  - Use one-vs-all
- Logistic regression vs SVM
  - If n is large, use logistic regression or SVM with no kernel
  - If n is small, m is intermediate, use SVM with Gaussian kernel
  - If n is small, m is large, create/add more features, use logistic regression or SVM with no kernel
  - Neural network tends to work well in all these settings, but is slow to train

Week 8: Clustering

K-means Algorithm

Algorithm

- Input: K(number of clusters), training set {x1, x2, ...}
- Randomly initialize K cluster centroids
- Repeat

  - For every training example, assign it to the closest centroid
  - For every cluster, compute a new centroid based on the mean of points assigned to the cluster
  - Eliminate cluster with no points to it #### Optimization objective

- Find the set of parameters that minimize this cost function
- The first part of the algorithm minimize it with respect to c
- The second part of the algorithm minimize it with respect to centroid #### Random Initialization
- Should have K < m
- Randomly pick K training examples
- Set centroids to be these examples
- Depending on different initialization, K-mens can be at local optima
  - Try multiple initialization
  - Run K-means
  - Compute cost functions
  - Pick one with lowest cost
  - Works well for K between 2 to 10
  - If K is large, generally the first initialization will give a good result #### Choosing the number of clusters
- Elbow method
  - Vary K
  - Plot K vs cost J
  - Find a elbow that before the cost reduces rapidly but slowly after
  - But many times there is no clear elbow
- Evaluate K-means based on a metric for how well it performs for later purpose

Week 8: Dimensionality Reduction

Motivation

- Data compression
  - Reduce 2D to 1D, 3D to 2D, etc.
  - Some features may be redundant or highly related
  - Save memory and space
  - Make learning algorithm faster
- Visualization ### Principal Component Analysis
- Problem Formulation
  - Try to find k vectors onto which to project data so that the projection error is minimized
- Algorithm
  - Preprocessing
    - Compute mean of each feature
    - Replace each training example with xj - meanj

        * If different features are on different scale, scale features to have comparable range of values
    - Suppose we want reduce from n-dimensions to k-dimensions
        * Compute a covariance matrix
        * Compute eigenvectors of covariance matrix
        * U is the vectors we want: each column is the vector ui
        * Use the first k column as u1, ... uk
        * x = Ureduce' * x
        * No x0 = 1 convention

```
[U, S, V] = svd(Sigma);
Ureduce = U(:, 1:k);
z = Ureduce' * x;
```

- Choosing number of principal components
    - Average squared projection error: 1/m * sum(||x(i) - xapprox(i)||^2)
    - Total variation in the data: 1/m * sum(||x(i)||^2)
    - Their ratio(average error / variation) is <= 0.01
    - 99% variance is retained
    - Algorithm
        * Try k = 1
        * If ratio is <= 0.01, if so use this k, else try next k
        * Alternatively, use the S matrix from svd
            · For given k, ratio can be computed 1 - (sum(Sii) from 1 to k) / (sum(Sii) from 1 to n)
            · Just need to run svd once and try different k until the ratio is small enough
- Reconstruction from Compressed Representation
    - xapprox = Ureduce * z, roughly equal to x
- Use PCA to speedup some learning algorithms
    - In supervised learning, the dimension of feature may be very large
        * Apply PCA to the training set to get a new training set
        * Map test data to the same dimension and make a prediction
- Bad use of PCA: to prevent overfitting
    - Regularization will work as well if you want to retain 99% of variance
    - Will throw away useful information
- Generally first try to run the whole thing with raw data, and use PCA only if this doesn't work well

Week 9: Anomaly Detection

Motivation

- Have a training dataset and xtest
- Have a model p(x)
- If p(xtest) < epsilon, flag as anomaly, else ok

- Example uses
  - Fraud detection
  - Manufacturing
  - Monitoring computer cluster and data center ### Algorithm
- Suppose we have training set with size m
- Each example has n features
- Choose features xi that you think might be indicators for anomaly
- Compute mean and variance
- p(x) = p(x1; mu1, sigma1)p(x2; mu2, sigma2)...p(xn; mu(n). sigma(n))
  - Assume each feature has a Gaussian distribution
- Anomaly if p(x) < epsilon ### Developing and Evaluating Anomaly Detection System
- Suppose we have some labeled data of anomalous and non-anomalous examples
- We have training examples, usually normal
- We have cross validation set and test set that can include anomalous examples
- On a cross-validation or test example, predict y
- Possible metrics: true positive false positive, false negative, true negative, precision/recall, F-1 score
- Can also use cross validation set to set the value of epsilon ### Vs Supervised Learning
- You should use anomaly detection
  - If you have very small number of positive examples and large number of negative examples,
  - Many different types of anomalies; hard for algorithm to learn
  - Future anomaly may not like any seen so far
- You should use supervised learning
  - If you have large number of positive and negative examples
  - Enough positive examples to learn
  - Future positive examples may be similar to the ones already seen ### Choosing Features
- For non-gaussian features
  - Take a log of data
  - Transform data into gaussian-like data
- Error analysis for anomaly detection
  - Want p(x) large for normal examples but small for anomaly
  - Most common problem: p(x) is comparable for both normal and anomalous examples
    - Look at the failing example, create some new features to capture that
- Choose features that may take on unusually large or small values in the case of anomaly ### Multivariate Gaussian Distribution
- Model all variables altogether, not separately
- Parameters: mean and covariance matrix
- Can be model corelation between data

- Parameter fitting:
- Algorithm:
  - Fit model
  - Given a test example, compute p(x), flag anomaly if p(x) is small
- Relationship to original model:
  - Axies aligned with X and Y
  - Covariance matrix must have 0 on off diagnal
- Use cases
  - Original model
    - Manually create features that capture anomaly
    - Computationaly cheaper, scale better to larger n
    - Ok if small training examples
  - Multivariate Gaussian
    - Automatically capture correlations between different features
    - Computationaly expensive
    - Must have m > n, or covariance matrix is not invertible
    - If you have redudant features, covariance matrix is not invertible

Week 9: Recommender System

Content Based Recommendation

- For each user perform a linear regression to learn a set of parameters
- Have features for each content
- Use the parameters and features to predict
- Learn parameters for all users; use gradient descent
- Content features are not always available

Collaborative Filtering

- Given parameters, to learn content feature vector
- Find the content feature vector such that the prediction is not far from the true value
- Can guess parameters and learn content features
- Then use content features to learn parameters
- Repeat
- Algorithm
  - Minimize x(1), x(2), ... and theta simultaneously
  - Initialize x and theta to small random values
  - Minimize the cost function
  - For a user with parameters theta and learned content features, make a prediction

Vectorization

- Low rank matrix factorization
  - Have a matrix X where each row is a content features vector
  - Have a matrix Theta where each row is a parameter vector for each user
  - X * Theta' gives the result
- Find related movies
  - For each product, we learn a feature vector
  - Find movies such that the distance between their feature vectors is small ### Implementation Detail: Mean Normalization
- For a user who has not rated any movie, the parameters learned will be all 0
- Subtract each rating for a movie by the average rating for that
- Use the new ratings to learn parameters and features
- For user, make a prediction using the learned parameters and features, then add the mean