CS M146 - Week 5

Xinzhu Bei

xzbei@cs.ucla.edu

February 10, 2018

Midterm Exam

- In class, close book, close note
- 2/13 2:00-3:30 pm
- True/False, short questions, Long Questions

Topics for Midterm

- Math
    - Linear Algebra
    - Probability
    - Calculus, Matrix Calculus
- Concepts: Supervised or unsupervised learning, hypothesis space, etc. (Will cover it later)
- Classifiers
    - Decision trees
    - K Nearest neighbors
    - Linear Classification, Perceptron
    - Logistic Regression
- Regression
    - Linear Regression, Polynomial Regression

Decision Trees

- Nodes, Edges, Leaves
- Entropy:

$$H[X] = -\sum_x P(X = x) \log P(X = x)$$

- For binary classification $H[X] = -P_+ \log P_+ - P_- \log P_-$
- Conditional Entropy

$$H[Y|X] = \sum_x P(X = x) H[Y|X = x]$$

- Information Gain:

$$Gain(Y, X) = H[Y] - H[Y|X]$$

** Notation might be different from lecture.

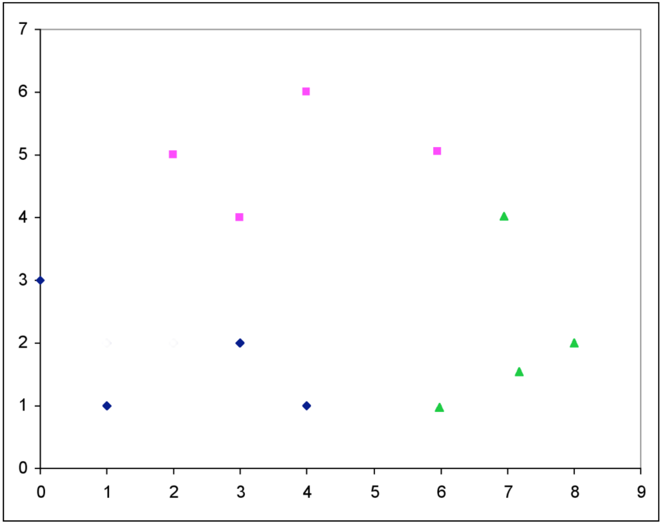
ID3 Algorithm: A greedy heuristic

See Slides CSM146-04

K Nearest Neighbor

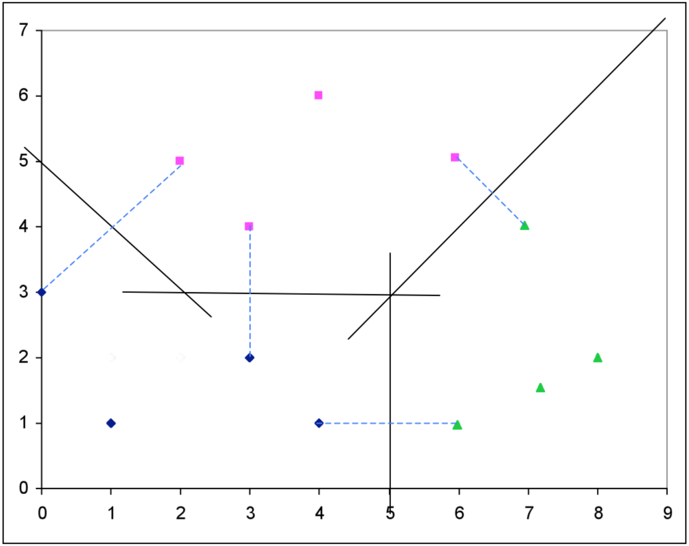
- Distance between instances:
  - Euclidean Distance $||\mathbf{x}_1 - \mathbf{x}_2||_2 = \sqrt{\sum_{i=1}^{n} (\mathbf{x}_{1,i} - \mathbf{x}_{2,i})^2}$
  - Manhattan Distance $||\mathbf{x}_1 - \mathbf{x}_2||_1 = \sum_{i=1}^{n} |\mathbf{x}_{1,i} - \mathbf{x}_{2,i})|$
  - Lp_norm $||\mathbf{x}_1 - \mathbf{x}_2||_p = (|\mathbf{x}_{1,i} - \mathbf{x}_{2,i}|^p)^{1/p}$
  - Hamming distance for Symbolic/categorical features: Number of bits that are different
- Choosing Hyper-parameter K (odd number): Validation, N-fold Cross-validation, Leave-one-out Cross-validation.
- Feature Normalization: e.g. $x_{nd} = (x_{nd} - \bar{x}_d)/s_d$
- Exercise: decision boundary of KNN Note: the decision boundary will change when K changes.

Decision boundary of KNN (K=1)



http://web.mit.edu/6.034/wwwbob/knn-bounds+soln.pdf

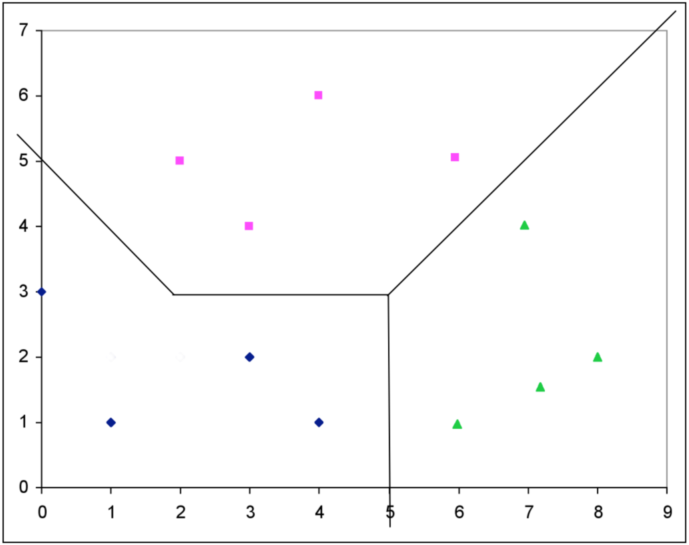
Decision boundary of KNN (K=1)



Construct lines between closest pairs of points in different classes.

Draw perpendicular bisectors.

Decision boundary of KNN (K=1)



End bisectors at intersections; extend beyond axes (to infinity).

Linear Classification, Perceptron

- **Linear Threshold Units**(LTUs): classify an example $\mathbf{x}$ using the following classification rule

$$\text{Output} = sgn(\mathbf{w}^T\mathbf{x} + b) = sgn(b + \sum w_i x_i)$$

- Decision boundary: hyper-plane
- Note: the normal vector is with the same direction of $\mathbf{w}$ and always point to the positive hyperspace!
- Trick to remove the bias term
- Convergence theorem, linearly separable
- **Margin** of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.
  The **margin of a data set** is the maximum margin possible for that dataset using any weight vector.

Linear Classification, Perceptron

- Perceptron Algorithm:

```
Given a training set $\mathcal{D} = \{(\mathbf{x}, y)\}$
Initialize $\mathbf{w} \in \mathbb{R}^n$
For $(\mathbf{x}, y)$ in $\mathcal{D}$:
    $\hat{y} = sgn(\mathbf{w}^T\mathbf{x})$ (if $y(\mathbf{w}^T\mathbf{x}) < 0$)
    If $\hat{y} \neq y$,$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$
Return $\mathbf{w}$
```

- Extension(optional):
  - **Voted Perceptron**: Remember every weight vector in your sequence of updates. At final prediction time, each weight vector gets to vote on the label.
  - **Averaged Perceptron**: Instead of using votes by all weight vectors, use the average weight vector
  - **Marginal Perceptron**: Pick a small positive and update $y_i\mathbf{w}^T x_i < \epsilon$

Linear Classification, Perceptron

```
X = [[1,1],[1,-1],[-1,1],[-1,-1]]
Y = [1,-1,-1,-1]
w = [0,0,1]
Round = 1, X = [1,1,1],y_gt = 1, np.dot(w,x) = 1, y_pred = 1
Round = 2, X = [1,-1,1],y_gt = -1, np.dot(w,x) = 1, y_pred = 1
        Weight updated w = [-1.000000,1.000000,0.000000]
Round = 3, X = [-1,1,1],y_gt = -1, np.dot(w,x) = 2, y_pred = 1
        Weight updated w = [0.000000,0.000000,-1.000000]
Round = 4, X = [-1,-1,1],y_gt = -1, np.dot(w,x) = -1, y_pred = -1
Round = 5, X = [1,1,1],y_gt = 1, np.dot(w,x) = -1, y_pred = -1
        Weight updated w = [1.000000,1.000000,0.000000]
Round = 6, X = [1,-1,1],y_gt = -1, np.dot(w,x) = 0, y_pred = 1
        Weight updated w = [0.000000,2.000000,-1.000000]
Round = 7, X = [-1,1,1],y_gt = -1, np.dot(w,x) = 1, y_pred = 1
        Weight updated w = [1.000000,1.000000,-2.000000]
Round = 8, X = [-1,-1,1],y_gt = -1, np.dot(w,x) = -4, y_pred = -1
Round = 9, X = [1,1,1],y_gt = 1, np.dot(w,x) = 0, y_pred = 1
Round = 10, X = [1,-1,1],y_gt = -1, np.dot(w,x) = -2, y_pred = -1
Round = 11, X = [-1,1,1],y_gt = -1, np.dot(w,x) = -2, y_pred = -1
Round = 12, X = [-1,-1,1],y_gt = -1, np.dot(w,x) = -4, y_pred = -1
```

Logistic Regression

- Linear Regression is a classifier, classifying by predict the probability $P(y = 1|x)$. We model the probability $P(y = 1|x)$ by a logistic curve:

$$\begin{aligned} P(y = 1|\mathbf{x}; \mathbf{w}) &= \sigma(\mathbf{w}^T\mathbf{x}) \\ P(y = -1|\mathbf{x}; \mathbf{w}) &= 1 - \sigma(\mathbf{w}^T\mathbf{x}) \\ \text{or equivalently } P(y|\mathbf{x}; \mathbf{w}) &= \sigma(y\mathbf{w}^T\mathbf{x}) \end{aligned} \tag{1}$$

  To classify, compute $P(y = 1|\mathbf{x}; \mathbf{w})$. If this is greater than half, predict 1 else predict -1.

- **Decision Boundary**: $\mathbf{w}^T\mathbf{x} = 0$
- Sigmoid function is a special case of Logistic function.
- The shape of Sigmoid function
- $\frac{d\sigma}{dz} = \sigma(z)(1 - \sigma(z))$, $1 - \sigma(z) = \sigma(-z)$

Maximum Likelihood Estimator

- Let $X_1, \cdots, X_N$ be IID with PDF $p(x|\theta)$ (also written as $p(x;\theta)$), the likelihood function is

$$L(\theta) = p(X_1, \cdots, X_N; \theta) = \prod_{i=1}^{N} p(X_i; \theta)$$

  The log-likelihood is $l(\theta) = \log L(\theta)$
- Some informal interpretation:
  - Our learning goal is to find an $h \in H$, such that $h_\theta(x) \approx P(y = 1|x)$
  - The likelihood is just the joint density of the data, expect that we treat it as a function of θ.
  - You only get to see what the nature wants you to see. Things you see are facts. These facts have an underlying process that generated it. These process are hidden, unknown, needs to be discovered. Then the question is: Given the observed fact, what is the likelihood that process P1 generated it? What is the likelihood that process P2 generated it? And so on... One of these likelihoods is going to be max of all. MLE is a function that extracts that max likelihood.

Learning Logistic Regression by MLE

Maximum Likelihood Estimator:

$$\arg\min_{\mathbf{w}} \sum_{i}^{m} \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)) \tag{2}$$

which is equivalent to

$$\arg\max_{\mathbf{w}} \sum_{i}^{m} y_i \log \sigma(\mathbf{w}^T \mathbf{x}) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}))$$

Linear Regression

- Least Mean Square (LMS) Regression

$$\min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

- By gradient descent:

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

- By Closed-form Solution:

$$\min_{w}(X^T\mathbf{w} - Y)^T(X^T\mathbf{w} - Y)$$

$$\mathbf{w}^* = (XX^T)^{-1}XY$$

- Extension: weighted LMS

General Framework for (Supervised) Learning

Problem Setting

- Set of possible instances X
- Set of possible labels Y
- Unknown target function $f : X \to Y$
- Set of function hypothesis $H = \{h | h : X \to Y\}$

Input: Training instances drawn from data generating distribution p

$$\{x_i, y_i\}_{i=1}^{n} = \{(x_1, y_1), \cdots, (x_n, y_n)\}$$

Output: Hypothesis h in H that best approximates f

Concepts

- Supervised Learning v.s. Unsupervised learning
- Classification v.s. Regression
- Parameter v.s. Hyper-parameter
- Validation, Cross validation, Leave-one-out validation
- Instance space, concept space, hypothesis space, label space
- Gradient Descent v.s. Stochastic Gradient Descent
- Convexity: definition, second derivative ≥ 0, multi-variate convexity, Hessian (semi-definite).
- Advantages and Disadvantages: Decision Trees, KNN, Linear Classifier (Perceptron), Logistic Regression, Linear Regression, etc.

Gradient Descent v.s. Stochastic Gradient Descent

- Batch Learning, Online Learning
- **(Batch) Gradient Descent**

Algorithm 1 Gradient Descent (J)

```
1: $t \leftarrow 0$
2: Initialize $\theta^{(0)}$
3: repeat
4:     $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \nabla J(\boldsymbol{\theta}^{(t)})$
5:     $t \leftarrow t + 1$
6: until convergence
7: Return final value of $\boldsymbol{\theta}$
```

- **Stochastic Gradient Descent**: Approximate the true gradient by a gradient at a single example at a time

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

```
1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For epoch $1 \dots T$:
3.     For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
4.         Update $w \leftarrow w - \eta \nabla f(w)$
5. Return $\boldsymbol{w}$
```

The End