Week8 Learning Reflection

Name: Xinyu Chang
Time: 2022.05.16-05.22
Class: CSE 416 Intro. Machine Learning
Lectures: 15 and 16

Summary

1. If never want to make false positive prediction
   always predict negative ($\alpha = \infty$)
   If never want to make false negative prediction
   always predict positive ($\alpha = -\infty$)
2. Bag of Words
   a. Pros: very simple to describe and computer
   b. Cons: Common words like “the” dominate counts of uncommon words, often it’s the uncommon words that uniquely define a doc.
3.

Concepts

(Lecture 15) K-Nearest Neighbors

Pre-class material

Notation

- $C_{TP} = \#\text{TP}, \quad C_{FP} = \#\text{FP}, \quad C_{TN} = \#\text{TN}, \quad C_{FN} = \#\text{FN}$
- $N = C_{TP} + C_{FP} + C_{TN} + C_{FN}$
- $N_P = C_{TP} + C_{FP}, \quad N_N = C_{FP} + C_{TN}$

Error Rate

$$\frac{C_{FP} + C_{FN}}{N}$$

Accuracy Rate

$$\frac{C_{TP} + C_{TN}}{N}$$

False Positive rate (FPR)

$$\frac{C_{FP}}{N_N}$$

False Negative Rate (FNR)

$$\frac{C_{FN}}{N_P}$$

True Positive Rate or Recall

$$\frac{T_P}{N_P}$$

Precision

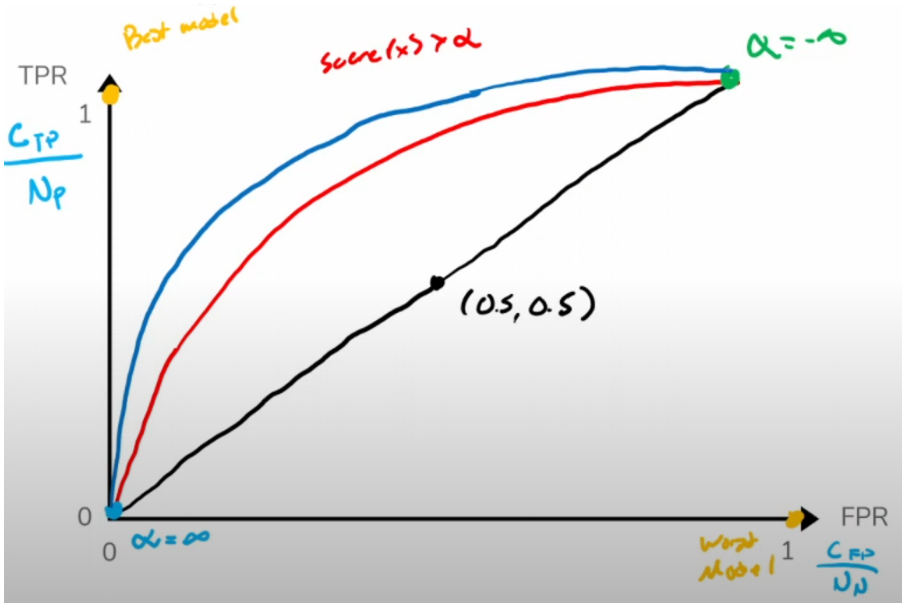
$$\frac{T_P}{C_{TP} + C_{FP}}$$

F1-Score

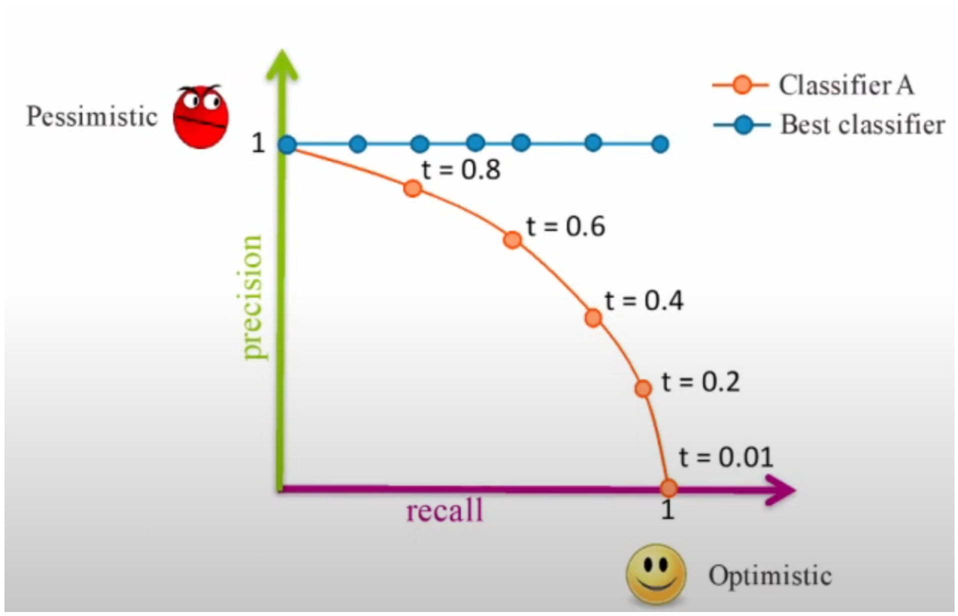
$$2\,\frac{Precision \cdot Recall}{Precison + Recall}$$

See more!

- **ROC Curve**
  - Examine the relationship between True Positive and False Positive.

- Want high TPR
- 

- **Precision**
  - Of the ones, I predicted as positive, how many of them were actually positive?
  - $\frac{C_{TP}}{C_{TP}+C_{FP}}$
- **Recall - True Positive Rate**
  - Of all the things that are truly positive, how many of them did I correctly predict as positive?
  - $\frac{C_{TP}}{N_p} = \frac{C_{TP}}{C_{TP}+C_{FN}}$
- **Precision-Recall Curve**
  - An optimistic model will predict almost everything as positive
    - High recall, low precision
  - A pessimistic model will predict almost everything as negative
    - High precision, low recall.
  - For logistic regression, we can control for how optimistic the model is by changing the threshold for positive classification
    - Before: $\hat{y}_i = +1 \; if \; \hat{P}(y = +1|x_i) > 0.5 \; else \; \hat{y}_i = -1$
    - Now: $\hat{y}_i = +1 \; if \; \hat{P}(y = +1|x_i) > t \; else \; \hat{y}_i = -1$

- 

- **Precision at *k***
  - If you show the top k most likely positive examples, how many of them are true positive.

Class session

- **Document retrieval - clustering and similarity - Unsupervised Learning**
  - Big idea: Define an embedding and a similarity metric for the books, and find the "nearest neightbor" to some query book
- **Curse of Dimensionality in k-means clustering and k-nearest neighbors**
- **Nearest Neighbors**
  - **1-Nearest Neighbor (1-NN), O(n)**
    - input
      - x_q Query example
      - $x_1, \dots, x_n$ Corpus of documents
    - output
      - The document in corpus that is most similar to x_q
      - $x^{NN} = argmin_{x_i \in [x_1, \dots, x_n]} distance(x_q, x_i)$
    - pseudo code

Input: x_q

$x^{NN} = \emptyset$

$nn\_dist = \infty$

$for\ x_i \in [x_1, \dots, x_n]:$

$\quad dist = distance(x_q, x_i)$

$\quad if\ dist < nn\_dist:$

$\quad\quad x^{NN} = x_i$

$\quad\quad nn\_dist = dist$

Output: x^{NN}

- **k-Nearest Neighbors (k-NN)**
  - output - return k books simialr to x_q

Input: x_q

current k NNs

$X^{k-NN} = [x_1, \dots, x_k]$

$nn\_dists = [dist(x_1, x_q), dist(x_2, x_q), \dots, dist(x_k, x_q)]$

$for\ x_i \in [x_{k+1}, \dots, x_n]:$

$\quad dist = distance(x_q, x_i)$

$\quad if\ dist < \max(nn\_dists):$

$\quad\quad remove\ largest\ dist\ from\ X^{k-NN}\ and\ nn\_dists$

$\quad\quad add\ x_i\ to\ X^{k-NN} and\ distance(x_q, x_i)\ to\ nn\_dists$

Output: X^{k-NN}

- **k-NN in retrieval, regression, and classification**
  - Retrieval
    - Return X^{K-NN}
  - Regression
    - $\hat{y}_i = \frac{1}{k}\sum_{j=1}^{k} x^{NN_j}$
  - Classification

- $\hat{y}_i = majorityClass(X^{K-NN})$

- **Embeddings**
  - **Bag-of-words**
    - Each document will become a W dimension vector where W is the number of words in the entire corpus of documents
    - The value of x_i[j] will be the number of times word j appears in document i
    - This ignores order of words in the document, just the counts.
  - **TF-IDF**
    - Goal: Emphasize important words
    - TF: Term frequency = word counts
    - IDF: Inverse doc frequency = $\log\frac{\#\,docs}{1+\,\#docs\ using\ word}$
    - Words that appear in every document will have a small IDF making the TF-IDF small. – its not useful at all.
- **Distance Metrics**
  - **Euclidian Distance**
    - $distance(x_i, x_q) \;=\; ||x_i - x_q||_2 \;=\; \sqrt{\sum_{j=1}^{D}(x_i[j] \,-\, x_q[j])^2}$
  - Manhattan Distance
    - $distance(x_i, x_q) \;=\; ||x_i - x_q||_1 \;=\; \sum_{j=1}^{D}|x_i[j] \,-\, x_q[j]|$
  - **Weighted Euclidian Distance**
    - $distance(x_i, x_q) \;=\; \sqrt{\sum_{j=1}^{D} a_j^{\,2}(x_i[j] \,-\, x_q[j])^2}$
    - For feature j : $a_j = \frac{1}{max_i(x_i[j]) - min_i(x_i[j])}$
- **Similarity metrics**
  - **Product similarity**
    - $x_i^{\,T}x_q = \sum_{j=1}^{D} x_i[j]x_q[j]$
    - This means a bigger number is better

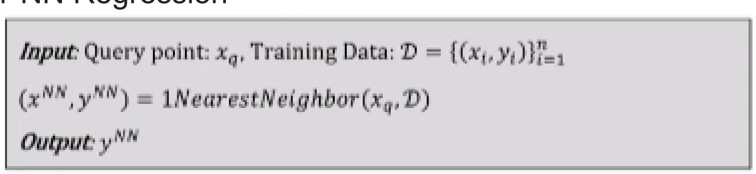
- **Cosine Similarity**
  - **Cosine Distance = 1 - Cosine Similarity**
  - $similarity = \frac{x_i^T x_q}{||x_i||_2 ||x_q||_2} = cos(\theta)$
  - Not a true distance metric
  - Efficient for sparse vector.

- **Normalization**
  - the length of each paperwork is different, then we can normalize the docs
  - Normalizing can make dissimilar objects appear more similar.
  - Common Compromise:
    - Just capture maximum word counts.

(Lecture 16)

Pre-class material - Local Methods, Locality Sensitive Hashing

- **1-NN Regression vs k-NN Regression**
  - 1-NN Regression



Input: Query point: x_q, Training Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

$(x^{NN}, y^{NN}) = 1NearestNeighbor(x_q, \mathcal{D})$

Output: y^{NN}

  -
  - “locally constant”
  - Can visualize it with a Voronoi Tessellation
    - Show all of the points that are closest to a particular training point.

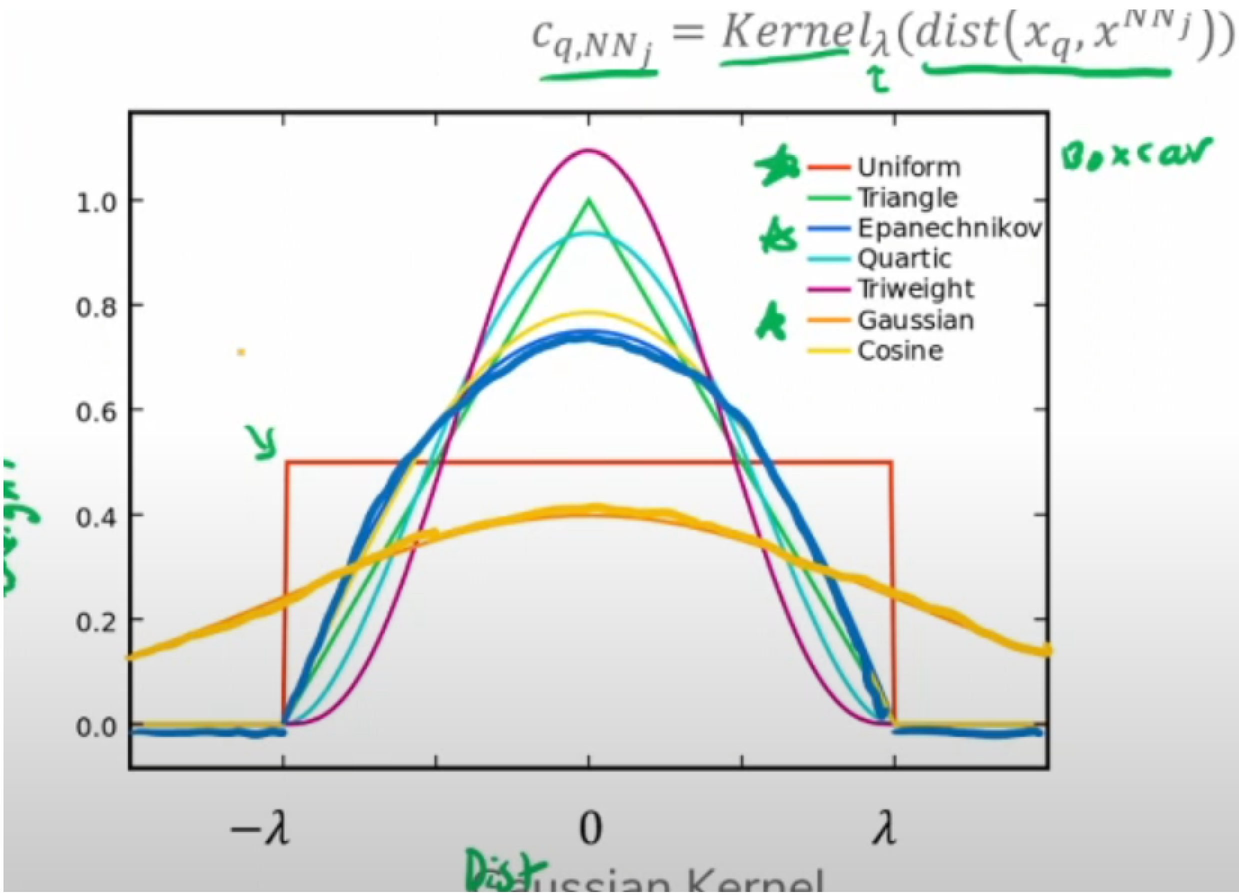
- Weaknesses
    - Inaccurate if data is sparse
    - Can wildly overfit. It relies too much on a single data point.
- k-NN Regression
    - Using a larger k, we make the function a bit less crazy
        - still discontinuous
        - boundaries are stil sort of a problem
        - somehow prevent overfit.
    - Issues
        - Have to choose right value of k,
            - if k is too large, model is too simple
        - Discontinuities matter in many applications
            - The error might be good, but would you believe a price jump for a 2640 sqft house to a 2641 sqft house?
    - use weighted k-NN to smooth the processes

- **Voronoi Tesselation**
- **Weighted k-NN**
    - Big idea: instead of trating each neighbor equally, put more weight on closer neighbors.
    - Predict:
        - $\hat{y}_q = \frac{\sum_{j=1}^{k} c_{q,NNj} y^{NNj}}{\sum_{j=1}^{k} c_{q,NNj}}$
    - Weight each nearest neighbor by some value $c_{q,NNj}$
    - How to choose $c_{q,NNj}$
        - Want it to be small if $dist(x_q, x^{NN_j})$ is large
        - want it to be large if dist is small
- **Kernel - turn distance into weight that satisfies the properties we listed before.**

$$c_{q,NN_j} = Kernel_\lambda(dist(x_q, x^{NN_j})), \lambda = \text{ bandwidth}$$

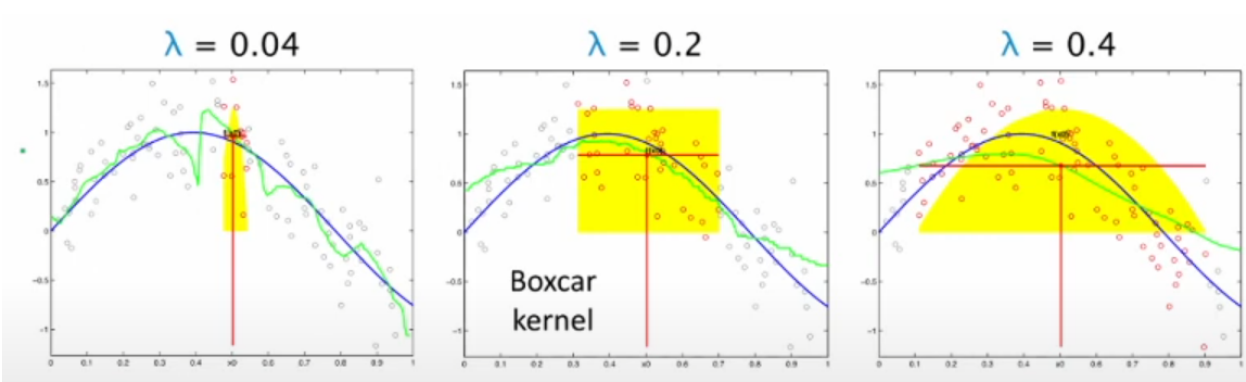
- **Uniform / Boxcar Kernel**
    - assigns equal weights
- **Epanechnikov Kernel**
    - outside of the bandwidth assigned zero
- **Gaussian Kernel**
    - normal distribution, outside of the bandwidth is not assigned zero

$$Kernel_\lambda(dist(x_i, x_q)) = exp(-\frac{dist(x_i,x_q)^2}{\lambda})$$



- **Kernel Regression**
    - instead of just using a kernel to weight the k nearest neighbors, can use the kernel to weight all training points.

$$\hat{y}_q = \frac{\sum_{i=1}^{n} c_{q,i} y_i}{\sum_{i=1}^{n} c_{q,i}} = \frac{\sum_{i=1}^{n} Kernel_\lambda(dist(x_i, x_q)) y_i}{\sum_{i=1}^{n} Kernel_\lambda(dist(x_i, x_q))}$$



Class session

- **Comparing k-NN, weighted k-NN, kernel regression - three local methods.**

  Non-parametric functions.

  - All of these models are taking averages.
    - K-Nearest: unweighted average. The samples used are K-nearest.
    - Weighted: weighted average (Use a kernel, tells you based on the distance between two points, what weight should use). The samples used are also K-nearest.
    - Kernel Regression: weighted average (Kernel). The samples used are all of the data points, many kernels give zero weight to points whose distance > lambda.
- **The efficiency of nearest neighbors**
  - Nearest neighbor methods require no training time (just store the data)
  - If there is a lot of data, might take O(n) if there are n data points.
  - There is not an obvious way of speeding this up.
  - Big Idea: Sacrifice accuracy for speed. We will look for an approximate nearest neighbor to return results faster.
- **Approximate nearest neighbors**
  - Use locality sensitive hashing to answer this approximate nearest neighbor problem
    - Design an algorithm that yields a close neighbor with high probability
    - These algorithms usualy come with a guarantee of what probability they will succeed.
- **Locality Sensitive Hashing (LSH)**

  an algorithm that answers the approximate nearest neighbor problem

  Big idea: break the data into smaller bins on how close they are to each other. When you want to find a nearest neighbor, choose an appropriate bin and do an exact nearest neighbor search for the points in that bin.

  More bins -> fewer points per bin -> faster searchj

  More bin -> more likely to make errors if we aren't careful

  - **Bin index**
    - Put the data in bins based on the sign of the score (2 bins total)
    - call negative score points bin 0, and the other bin 1 (bin index)
    - When asked to find neighbor for query point, only search through points in the same bin.
    - This reduces the search time to n/2 if we choose the line right.
  - **How to choose bins**
  - **Searching nearby bins**

Section material

Uncertainties

1. What are some examples used Euclidean and some examples used Cosine Distance? Any typical examples?