CSE 6740 CDA Class Notes

Cole Welch

9/15/2025

1 Lecture 8: Naive Bayes, KNNs, & Logistic Regression

→ Topics covered in this lecture:

- Bayes & Naive Bayes
- K Nearest Neighbors
- Logistic regression

Classification

→ Given dataset, can we learn classifier (as mapping from features to labels)?

- In general → need to construct decision boundary
- Will cover how to do so w/ density estimation methods

Bayes' Rule

→ Recall: use

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x,y)}{\sum_{y'} P(x,y')}$$

where

$$\begin{aligned} P(x|y) &= \text{posterior (harder to measure)} \\ P(x|y) &= \text{likelihood} \\ P(y) &= \text{prior (easier to measure)} \\ P(x) &= \text{normalization constant} \end{aligned}$$

→ E.g., under Gaussian assumption:

$$P(x|y) = N(x; \mu_y, \Sigma_y)$$
$$P(y|x) = \frac{P(y)N(x; \mu_y, \Sigma_y)}{\sum_{y'} P(y')N(x; \mu_{y'}, \Sigma_{y'})}$$

- W/ known covariance, can get posterior
- Estimate likelihood

Bayes' Decision Rule
→ Repeat above ∀ classes (labels denoted by i):

$$q_i(y) = P(y = i|x) = \frac{P(x|y = i)P(y = i)}{P(x)}$$

→ (e.g., $y \in \{0, 1\}$)
→ Compute likelihood ratio (> 1 or < 1) to make decision:

$$g(x) = \frac{q_1(x)}{q_0(x)} = \frac{P(x|y = 1) \cdot P(y = 1)}{P(x|y = 0) \cdot P(y = 0)}$$

- $g(x) > 1 \Rightarrow y = 1$
- Otherwise $y = 0$

→ Repeat, find i w/ smallest largest $g(x)$
→ Decision boundary is curve satisfying $g(x) = 1$

Pros	Cons
• Simple	• DE difficult for higher dimensional spaces

1. Naive Bayes Classifier
→ Easier than pure Bayes in practice
→ Differences:

- Assume all features $X = (X_1, \ldots, X_d)$ fully independent (gets around $P(x|y)$ estimation difficulty for high-dimensional data)
- Decision boundary: $y = \arg\max_y P(y) \prod_{i=1}^{d} P(x_i|y)$

Ex: Stolen Car Prediction:

$$\begin{aligned} P(\text{stolen}|\text{red, suv, domestic}) &\propto P(\text{red, suv, domestic}|\text{stolen})P(\text{stolen}) \\ &= P(\text{red}|\text{stolen})P(\text{suv}|\text{stolen})P(\text{domestic}|\text{stolen})P(\text{stolen}) \\ &= \left(\frac{3}{5}\right)\left(\frac{1}{5}\right)\left(\frac{2}{5}\right)\left(\frac{5}{10}\right) \\ &= \frac{3}{125} \end{aligned}$$

2. K Nearest Neighbors

→ Geometric intuition - nearby data points have similar labels
→ Definition of Nearest Neighbors (NN):

- Assign X some label as closest training point $x_i \; \forall \; x$
- (Defines Voronoi partition of space)

→ NN Characteristics:

- Nonparametric
- Nonlinear Classifier
- Easy to Memorize

→ Generalize NNs: to K Nearest Neighbors Classifier (kNN) for $k \geq 1$
→ Definition:

- Now, assign each x a label by majority rate over k training points closest to x
- Mathematical Definition:
  - $I_k(x) :=$ index of $i = 1 \rightarrow k$th nearest training points to x
  - Letting $y_i = i$, can define classifier by

$$f_k(x) := \text{sign}\left(\sum_{i \in I_k(x)} y_i\right)$$

→ Choosing k

- Modeling choice—no definitive way to choose it
- k too small makes every point its own island
- k too large classifies everything the same; gives no info

- k chosen to have smallest training error a bad idea—$k = 1$ always 100% accurate, always the result
  - Can split into training & validation sets to solve this

→ Computational *KNN*

- Similar to KDE—no training / learning phase
- Resource usage:
  - Memory: $O(nd)$ where $n = \#$ data points, $d =$ dimension
  - Training compute: $O(n \log n)$
  - Testing compute: $O(\log n)$
- Need smart data structure usage to achieve this!
- (E.g., K-D tree or ball tree)

3. Logistic Regression

→ Precursor of neural networks
→ Directly tries to use decision boundary w/ only decision boundary $h(x) = \log \frac{q_1(x)}{q_0(x)}$

- Do NOT estimate Bayes' rule parts → $P(x|y)$ & $P(y)$

→ Model $h(x)$, $P(y = 1|x)$ as function of x

- No probabilistic meaning
- So can't be used to sample data

→ Characteristics

- *Discriminative classifier*
- Avoids difficult density estimation
- But actually achieves better empirical results than Naive Bayes

→ Def: Use logistic function; e.g.,

$$P(y = 1|x, \theta) = \frac{1}{1 + e^{-\theta^T x}} = f(u) = \frac{1}{1 + e^{-u}}$$

- Approaches 1 as $\theta^T x = u$ gets larger

→ Finding $\hat{\theta}_{MLE}$ w/ gradient of $l(\theta)$:

- Define

$$\begin{aligned} l(\theta) &= \log \prod_{i=1}^{n} p(y_i|x_i, \theta) \\ &= \sum_{i=1}^{n} (y_i - 1)\theta^T x_i - \log(1 + e^{-\theta^T x_i}) \\ \frac{\partial l}{\partial \theta} &= \sum_{i=1}^{n} (y_i - 1)x_i + \frac{x_i e^{-\theta^T x_i}}{1 + e^{-\theta^T x_i}} \end{aligned}$$

- No closed form solution to this
- Must use *gradient descent*:
  - Update iteratively by going along direction of negative gradient at current point
  - Update rule:

$$x_{k+1} = x_k - \gamma_k \nabla_x f(x_k)$$

    Where

$$\gamma_k = \text{step size (or } \textit{learning rate})$$

- **Algorithm**:
  1. Initialize parameter θ^0
  2. Do:

$$\theta^{t+1} = \theta^t + \eta \sum_{i=1}^{n} (y_i - 1)x_i + \frac{x_i e^{-\theta^T x_i}}{1 + e^{-\theta^T x_i}}$$

     while $||\theta^{t+1} - \theta^t|| > \epsilon$

- Likely to need multiple initializations to navigate optimization landscape!
- Will cover gradient descent in detail later