CIS 520 - Machine Learning Notes

Eric Oh - University of Pennsylvania

September 15, 2017

1 Local Learning

The class of local learning methods seems less like learning and more like pure memorization - and in some ways it is. Main idea: given a new example x, find the most similar training example(s) and predict a similar output. We generally assume some measure of similarity or distance between examples and learn a *local* method in a neighborhood of the new example. Consider the one dimensional regression problem:

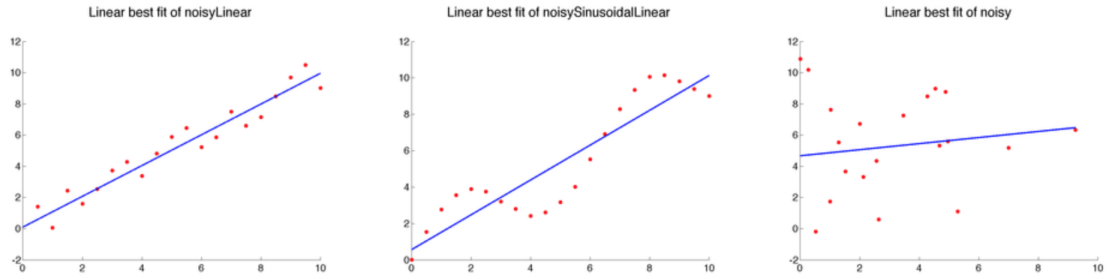


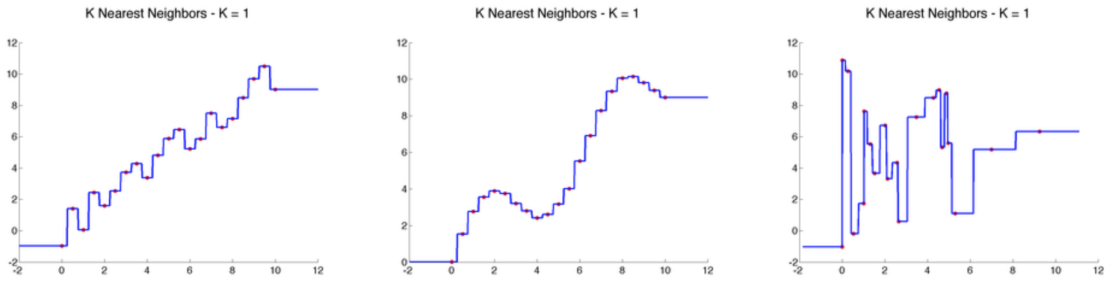
Figure 1: Clearly, linear model doesn't fit data well always

1.1 Nearest Neighbor

We could try adding higher-order polynomial terms or try a local method like nearest neighbors:

```
1 - Nearest Neighbor Algorithm
  1. Given training data D={x_i,y_i}, distance function d(.,.), and input x
  2. Find j = arg min_i d(x,x_i) and return y_j
```

Some examples of the above algorithm with $d(x, x_i) = |x - x_i|$ as x varies:



Generally, we use some arbitrary distance function to define nearest neighbor. Some common choices are the L_1, L_2, or the L_∞ norms.

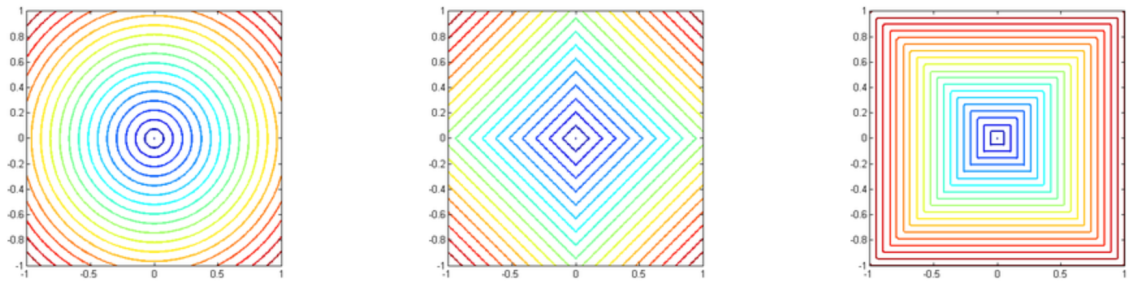
Definition 1.1. ***Norm*** *- for all $a \in \mathbb{R}$ and all $u, v \in V$,*

- $L_p(av) = |a|L_p(v)$
- $L_p(u + v) \leq L_p(u) + L_p(v)$ - triangle inequality or subadditivity
- *If $L_p(v) = 0$ then v is the zero vector*

Example 1.1. *L_p norm :* $\left(\sum_j |x_j|^p\right)^{\frac{1}{p}}$

Definition 1.2. L_0 ***pseudo-norm*** : $|x|_0$ = *number of* $x_j \neq 0$

The contours of the most common distance measures are below.



L_2*norm* L_1*norm* L_{inf}*norm*

Note that for the L_p norm, p must be at least 1. L_p norms with $p < 1$ result in **concave** contours - NOT GOOD. So how do norms delate explicitly to distances?

$$d_p(x, y) = |x - y|_p$$

Note that norms are **SCALE-INVARIANT**. Thus, if x is a vector of non-negative terms, then

$$\sum_i x_i \qquad \text{and} \qquad \sum_i i x_i$$

are both norms.

1-Nearest Neighbor is one of the simplest examples of a **non-parametric** method (ie. model structure determined by the training data). Non-parametric models are much more flexible and expressive than parametric models, and thus **overfitting** can be a big concern. One nice property of NP models is that since the complexity increases as the data increases, with enough data, NP models can model nearly anything and achieve close to zero bias for any distribution (ie. consistent..sort of). A huge problem with this, however, is that for any finite sample size there will likely be huge variance (ie. overfitting).

One way to reduce the variance is local averaging : instead of just one neighbor, find K and average their predictions.

```
K-Nearest Neighbors Algorithm
  1. Given training data D={x_i,y_i}, distance function d(.,.), and input x
  2. Find {j_1,...,j_K} closest examples wrt d(x,.)
      a. (Regression) if y \in \mathbb{R}, return average of y_{j_k}
      b. (Classification) if y is 1 or -1, return majority
```

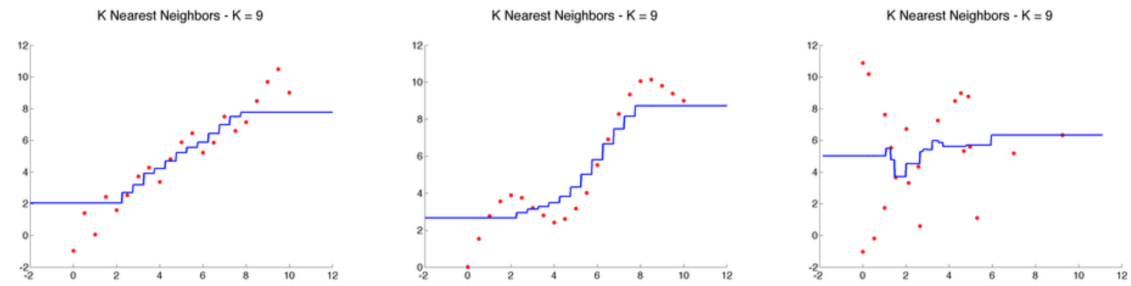


Figure 2: 9 nearest neighbors - smoother prediction but edges still sketchy

1.2 Kernel Regression

Two main shortcomings of K-NN method:

1. All neighbors receive equal weight
2. Number of neighbors chosen globally

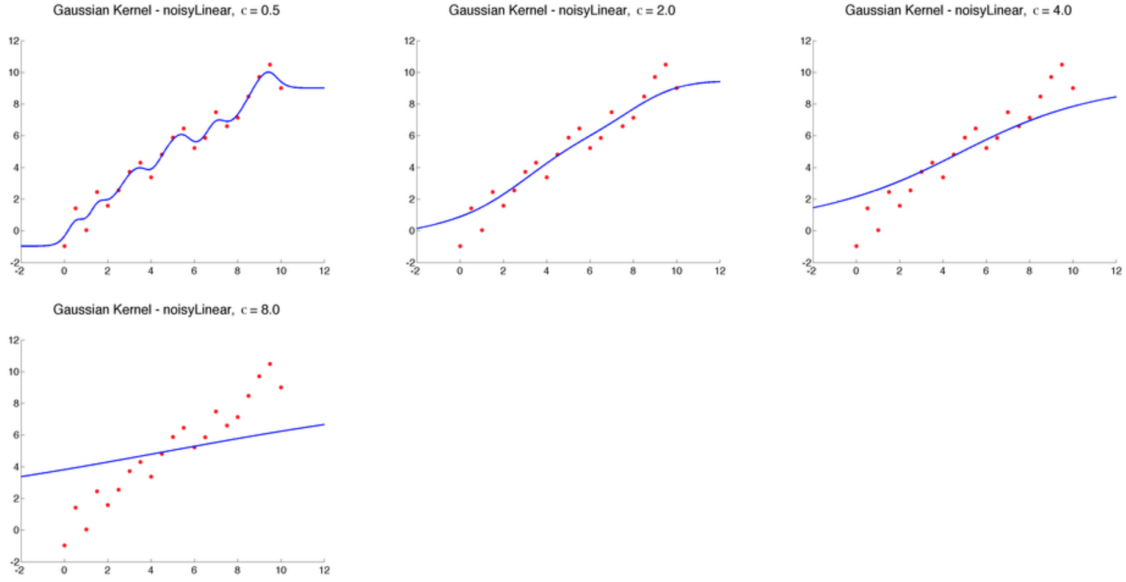
Kernel regression address these issues by *using all neighbors* but with different weights (ie. closer neighbors receiving higher weights and vice versa). Weighting function is a **kernel** and it measures similarity (as opposed to distance) between examples. Can convert from a distance d(.,.) to kernel K(.,.) easily - most common way is via Gaussian kernel (ignoring normalizing constants):

$$K(x, x_i) = \exp\left(\frac{-d^2(x, x_i)}{h}\right)$$

Kernel Regression/Classification Algorithm

1. Given training data $D = x_i, y_i$, Kernel function K(.,.), and input x
   - (Regression) if $y \in \mathbb{R}$, return weighted average: $\sum_i y_i \frac{K(x,x_i)}{\sum_j K(x,x_j)}$
   - (Classification) if $y \in \pm 1$, return weighted majority: $\text{sgn}\left(\sum_i y_i K(x, x_i)\right)$

One of the keys kernel regression is h, the **bandwidth**, which determines how quickly the influence of neighbors falls off with distance. The following shows the effect of increasing levels of h.



As $h \to \infty$, all the neighbors weight the same and the prediction is the global average.
As $h \to 0$, prediction tends to 1-NN.
Clearly choosing the "right" h is very important - in practice, usually use **cross-validation** to pick.

2 Decision Trees

- Gives nice interpretable output but each split uses exponentially less data (observations)
- How do we decide which features to split on? How to define importance? Need definition of information

2.1 Entropy

Suppose X can have one of m values, $V_1, \ldots, V_m$, with $P(X = V_i) = p_i$. Entropy is the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from X's distribution.

Definition 2.1. *Entropy :* $H(X) = -\sum_{j=1}^{m} p_j \log_2 p_j$

- High entropy : X is from uniform (boring) distribution. Values sampled from it would be all over the place.
- Low entropy : X is from varied (valleys and peaks) distribution. Values sampled from it would be more predictable (from the peaks).

Entropy is the expected value of the information content (surprise) of the message $\log_2 p_j$. Range of entropy is from 0 to ∞.

Example 2.1. *1. If an event is certain, entropy is 0*

2. *If two events are equally likely, entropy is 1* $(-(-0.5 * \log(0.5) + -0.5 * \log(0.5))$

Definition 2.2. *Specific Conditional Entropy :* $H(Y|X = v)$*, the entropy of* Y *among only those records in which* X *has value* v*.*

Suppose I'm trying to predict output Y and I have input X

X = College Major

Y = Likes "Gladiator"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
History	No
Math	Yes

Let's assume this reflects the true probabilities

E.G. From this data we estimate

- *P(LikeG = Yes) = 0.5*
- *P(Major = Math & LikeG = No) = 0.25*
- *P(Major = Math) = 0.5*
- *P(LikeG = Yes | Major = History) = 0*

Note:

- *H(X) = 1.5*
- *H(Y) = 1*

Definition 2.3. *Conditional Entropy :* $H(Y|X) = \sum_j P(X = v_j)H(Y|X = v_j)$*, the average specific conditional entropy of* Y

- *H(Y|X=Math) = 1*
- *H(Y|X=History) = 0*
- *H(Y|X=CS) = 0*

Definition 2.4. *Information Gain :* $IG(Y|X) = H(Y) - H(Y|X)$*, how many bits on average would it save me in transmitting* Y *if both ends of the line knew* X*?*

Example 2.2. *Suppose,*
$H(Y) = 1, H(Y|X) = 0.5$*. Then* $IG(Y|X) = 1 - 0.5 = 0.5$*, and knowing* X *gives gain of 0.5 information (unit is bits).*

Note: range of IG is from 0 to ∞. If Y and X are independet, then $IG = 0$.
In general, features with more options (more categories) have more possible information gain than binary features.

Key to Decision Trees: How do we prevent overfitting? How we know when to stop splitting?! Number of leaves grows exponentially

3 MLE

Self-explanatory.

3.1 Learning Guarantees

The following bound will be useful (based on Hoeffding inequality):

Theorem 3.1. $P(|\hat{\theta}_{MLE} - \theta| \geq \epsilon) \leq 2e^{-2n\epsilon^2}$

In other words, MLE deviates from true parameter exponentially with n.

4 MAP (maximum a posteriori)

Bayes rule gives

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Definition 4.1. $\hat{\theta}_{MAP} = arg\ max_\theta P(\theta|D) = arg\ max_\theta \log(P(D|\theta)) + \log(P(\theta))$

Note: the $\log(P(\theta))$ acts as a regularizer for the log likelihood for MAP estimation.
Note: as $n \to \infty$, the prior's effect vanishes and we recover the MLE.
Note: MAP is the simplest Bayesian approach to parameter estimation - sets estimate equal to mode of posterior distribution $P(\theta|D)$. There is more information in the posterior such as the mean and variance of the distribution.

5 Regression

6 Classification

If we had binary data (0's and 1's), we could fit linear regression and just predict everything > 0.5 to be 1 and everything < 0.5 to 0. We **CAN** fit linear regression to binary data, but this is bad!! This gives probabilities below 0 and above 1!!

Make a transformation! Most common one is **logit** function:

$$f(s) = \frac{1}{1+e^{-s}}$$

6.1 Logistic Regression

Thus we obtain something like (for Y=1 and Y=-1)

$$P(Y=1|x,w) = \frac{1}{1+\exp(-w^T x)} = \frac{1}{1+\exp(-yw^T x)}$$
$$P(Y=-1|x,w) = 1 - P(Y=1|x,w) = \frac{\exp(-w^T x)}{1+\exp(-w^T x)} = \frac{1}{1+\exp(-yw^T x)}$$

Since log-likelihood for logistic regression is convex, can solve using **gradient ascent/descent**!

Can also do the same with multinomial outcomes!

$$P(Y=k|x,w) = \frac{\exp(w_k^T x)}{1+\sum_{k'=1}^{K-1}\exp(w_{k'}^T x)} \qquad k = 1,\ldots,K$$

6.2 Naive Bayes

Naive Bayes classifier is an example of a **generative** model : we model $P(x,y) = P(y)P(x|y)$ where

- $P(y)$: class prior
- $P(x|y)$: class model

We estimate them separately as $\hat{P(y)}$ and $\hat{P(x|y)}$. Note: the class prior is different than the parameter prior classic Bayesian analysis. We then use our estimates to output a classifier using Bayes rules:

$$h(x) = \text{argmax}_y \hat{P(y|x)} = \text{argmax}_y \frac{\hat{P(y)}\hat{P(x|y)}}{\sum_y \hat{P(y)}\hat{P(x|y)}} \propto \text{argmax}_y \hat{P(y)}\hat{P(x|y)}$$

In English: Classify a new instance x based on a tuple of attribute values $x = (x_1,\ldots,x_n)$ into one of the classes $y_j \in Y$.

Estimating $P(y)$: Can be estimated from the frequency of classes in the training examples

Estimating $P(x|y)$: The *Naive Bayes assumption* is that $P(x|y) = \prod_i P(\hat{x_i}|y)$ (Conditional independence)

If we assume both Y and X are discrete, can do (MLE estimation)

$$P(\hat{y}) = \frac{N(y = y_j)}{N} \qquad P(\hat{x_i}|y) = \frac{N(x_i = x_j, y = y_j)}{N(y = y_j)}$$

Note: If Y and/or X are continuous, we can pick some model for the distribution (ie. Gaussian).

Problem with MLE though is that if we see no training examples for some class, we have 0 probabilities! To deal with this, we can smooth the probabilities to avoid overfitting.

$$P(\hat{x_i}|y) = \frac{N(x_i = x_j, y = y_j) + mP_{i,k}}{N(y = y_j) + m}$$

This is sort of like **empirical bayes** where our prior includes information on probabilities in the entire sample?

Note for implementation: DO NOT MULTIPLY PROBABILITIES. THIS LEADS TO UNDERFLOW. INSTEAD ADD LOG PROBABILITIES.