Problemset 2: K Nearest Neighbor and Linear Regression

Jiarong Ye

September 14, 2018

Q1

1.(25 pts.) Suppose 7 nearest neighbor search for a query sample returns {7, 6, 8, 4, 7, 11, 9} as the values of the function at the 7 nearest data samples in the training set.

(a) Assuming prediction using 7 nearest neighbor interpolation, what is the predicted value of the function for the query sample?

(b) Assuming prediction using 7 nearest neighbor regression, what is the predicted value of the function for the query sample?

- (a) Since the nearest neighbor interpolation selects the value of the nearest point and does not consider other values of neighboring points at all, thus $\arg\min distance(x', x_i)$ is achieved at i when $f(x_i) = 7$, thus the predicted value of the function for the sample x' should be 7

- (b) the result depends on the loss function applied, for instance:

  * L1 loss

$$\min \frac{1}{n} \sum_{i=1}^{n} \mid f(x') - f(x_i) \mid$$

  in this case

$$\begin{aligned}\arg\min \{ \mid f(x') - 7 \mid + \mid f(x') - 6 \mid + \mid f(x') - 8 \mid + \mid (x') - 4 \mid \\ + \mid f(x') - 7 \mid + \mid f(x') - 11 \mid + \mid f(x') - 9 \mid \}\end{aligned}$$

  is achieved when $f(x')$ is the median of the array, thus the predicted value of the function for the sample x' should be 7

  * L2 loss

$$\min L, \textit{ where } L = \frac{1}{n} \sum_{i=1}^{n} \big(f(x') - f(x_i)\big)^2$$

  in this case

$$\begin{aligned}\arg\min \{ \big(f(x') - 7\big)^2 + \big(f(x') - 6\big)^2 + \big(f(x') - 8\big)^2 + \big(f(x') - 4\big)^2 + \\ \big(f(x') - 7\big)^2 + \big(f(x') - 11\big)^2 + \big(f(x') - 9\big)^2 \}\end{aligned}$$

is achieved when

$$\frac{\partial L}{\partial f(x')} = 2[\big(f(x') - 7\big) + \big(f(x') - 6\big) + \big(f(x') - 8\big) + \big(f(x') - 4\big) + \big(f(x') - 7\big) + \big(f(x') - 11\big) + \big(f(x') - 9\big]]$$

$$= 14f(x') - 2 \cdot 52 = 0$$

$$f(x') = 7.429$$

(**also the mean of the array**)

thus the predicted value of the function for the sample x' should be 7.429

Q2

2.(25 pts.) Alex is building a K nearest neighbor classifier for predicting the language to which a word belongs. The words use the Latin alphabet (which is used by English, Spanish, German and several other European languages). For better or for worse, Alex chooses to represent the words using three features: length of the word, number of vowels in the word, and whether or not the word ends in the letter 'n" (0 for No, 1 for Yes). Suppose she has training data for two languages: regeln: German; pido: spanish.

(a) What is the dimensionality of the feature space?

(b) How are the two training data samples encoded in terms of the 3 features listed?

(c) Using 1-nearest neighbor classifier using the Manhattan distance metric, what is the prediction for the word "neben"?

- (a) The dimensionality of the feature space is 3 because there are three engineered features in feature set:

  * length of the word
  * number of vowels in the word
  * whether or not the word ends in the letter 'n'

- (b)

sample	length of word	number of vowels in the word	whether or not the word ends in the letter 'n'
regeln	6	2	1
pido	4	2	0

- (c)

the feature encoding for 'neben' is:

sample	length of word	number of vowels in the word	whether or not the word ends in the letter 'n'
neben	5	2	1

Using the Manhattan distance metric:

$$distance = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$$

Therefore:

- 'neben' to 'regeln':

$$distance_1 = |5 - 6| + |2 - 2| + |1 - 1| = 1$$

- 'neben' to 'pido':

$$distance_2 = |5 - 4| + |2 - 2| + |1 - 0| = 2$$

So:

$$\min(distance_1, distance_2) = distance_1 = 1$$

Thus 'neben' is more likely to be German.

Q3

3.(25 pts.) Assume that you have a trained K nearest neighbor classifier.

(a) What is the runtime complexity of this classifying p data samples given a K nearest neighbor classifier trained on m training samples, each represented using d features?.

(b) Can you do better, using a carefully designed data structure with a K nearest neighbor search algorithm that is designed to take advantage of the data structure? Explain.

(a) Explain by steps:

- for each of the p data samples:
    * step 1: distance with each of the training data points need to be calculated for later comparison, each calculation takes $O(d)$, thus for all m training data points, the runtime complexity is $O(dm)$
    * step 2: then for the top K required nearest neighbors 1 to k, each loop through all m distance factor calculated, selecting the index with the smallest distance, the runtime complexity is $O(km)$

Summary:

for each of the p data samples, a classification task takes runtime complexity of $O(dm) + O(km)$;

for total p samples, the runtime complexity is $O((d + k)mp)$

(b) As we can see, the part where we probably could optimize the runtime complexity would be **step 2** where in part (a), we traverse the entire m calculated distance factor to search the shortest distance, costing $O(km)$ complexity

we could use a more carefully designed data structure, like the **KD-Tree** for this part in the KNN search algorithm by organize the m training data points into different segmentations.

Basically how KD-Tree works:

- If there is just one point, form a leaf with that point.
- Otherwise, divide the points in half by a line perpendicular to one of the axes.
- Recursively construct k-d trees for the two sets of points.

Runtime complexity of searching for top K nearest neighbors:

Since $depth\ of\ KDTree = \log m$

for each of the p data samples:

- each distance calculation with m training data points takes $O(d)$, thus for all m training data points, the runtime complexity is $O(dm)$
- for the top K required nearest neighbors 1 to k, each takes $O(\log m)$ to traverse the tree and extract the shortest distance, then the runtime complexity is $O(k \log m)$

Summary:

- If construct a KD-Tree to organize the training data first then do the knn search
  * the runtime complexity for knn searching is $O(k \log m)$
  * the whole runtime complexity of the adjusted KNN algorithm for p samples is $O((dm + k \log m)p)$

Q4

4.(25pts.)Consider approximation of $f(X)$ where $X \in \mathbb{R}^d$ given a training set $\{(X_1, y_1), ..., (X_n, y_n)\}$ using linear regression. That is, the goal is to find a linear approximation $y(X) = W \cdot X + w_0$ where W and w_0 are learnable parameters. Suppose we further know that the function is sparse, i.e., only a few of the d features define the behavior of f. How would you modify the objective function to be minimized to ensure that the resulting linear is sparse? (Hint: Add a term to the mean squared error that when minimized attempts to drive the weights to 0).

Since the function is sparse, with a few among all d features that actually non-zero, thus to make sure the resulting linear is sparse as well, we could modify the loss function by adding a regularization term, a smoothing technique to prevent overfitting by bounding the weights. And in this case, to drive the weights to 0, it would be better if the added regularization term is **L1 regularization**.

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \sum_{j=1} w_j \cdot x_{ij} - w_0)^2$$

Objective function:

$$\arg\min_{w} \{L + \lambda||w||_1\}$$

Explain:

L1 regularization is better for sparse fucntion, because it is more likely to create 0-weights:

- mathematical proof:

for a model consisting of the weights $(w_1, w_2, ..., w_m)$

1. With L1 regularization, the loss function $L_1(w) = f(w) + \lambda||w||_1$
2. With L2 regularization, the loss function $L_2(w) = f(w) + \lambda||w||_2^2$

$$\frac{\partial L_1}{\partial w} = f(w)' + \lambda \cdot \frac{w}{|w|}, \; where \; \frac{w}{|w|} = \pm 1$$

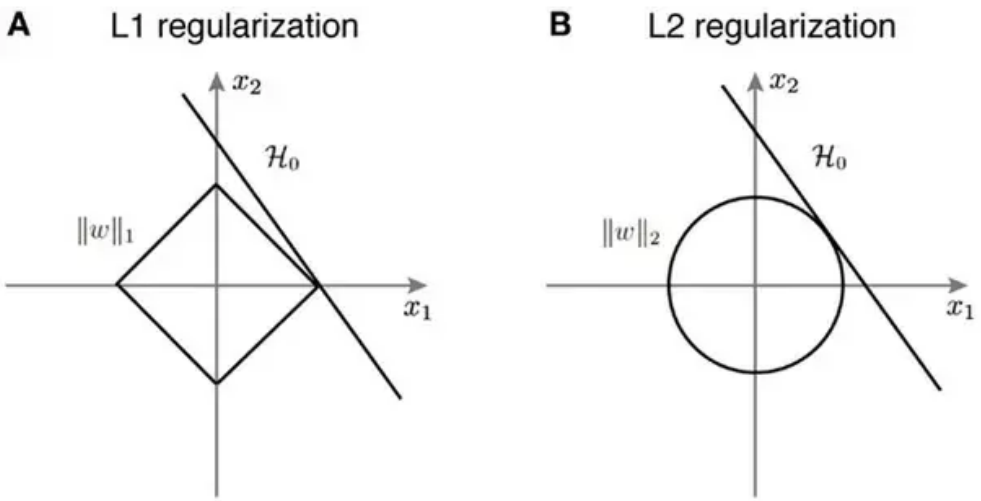
Optimization theory says that the optimum of a function is either the point with 0-derivative or one of the irregularities (corners, kinks, etc.). So, it's possible that the optimal point of L is 0 even if 0 isn't the 0-derivative of $f(w)$. In fact, it would be 0 if λ is large enough (stronger regularization effect).

$$\frac{\partial L_2}{\partial w} = f(w)' + 2\lambda w$$

$$w = w - \eta \cdot \frac{\partial L}{\partial w}, \; where \; \frac{\partial L}{\partial w} is \; dependent \; to \; w$$

So in L2 regularization, gradient $\frac{L_2}{w}$ is linearly decreasing towards 0 as the weight goes towards 0. Therefore, it will take smaller and smaller steps as a weight approaches 0. The 0-derivative point of L_2 can get very small when λ increases, but still won't be 0 unless f'(0)=0

- intuition:



As the objective is to minimize the error function, we then focus on the intersection with the L1, L2 regions. In this case, that corresponds to a sparse solution. It's apparent that the diamond region represented by L1 norm is more likely to create an intersection that cross the axes, thus L1 norm has the property of producing many coefficients with zero values or very small values with few large coefficients.