ECE523: Engineering Applications of Machine Learning and Data Analytics
Due 02/09/2017 @ 11:59PM (D2L)

Name: David Ake

Signature: Dmk A

Date: 2/10/18

Instructions: There are seven problems. Partial credit is given for answers that are partially correct. No credit is given for answers that are wrong or illegible. All work must be supported and code must be submitted for credit.

Theory: _____

Practice: _____

Total: _____

Part A: Theory

① In class we derived and discussed linear regression in detail. Find the results of the minimization of the loss of the sum of the squared errors. However add in the penalty for an L2 penalty on the weights. More formally:

$$\arg\min_{w} \left\{ \sum_i (w^T x_i - y_i)^2 + \lambda \|w\|_2^2 \right\}$$

How does this change the solution to the original linear regression solution? What is the impact of adding in this penalty?

(a) Part 1: Find results of the minimization for the "loss of the sum of the squared errors"

Given: $\arg\min_{w} \left\{ \sum_i (w^T x_i - y_i)^2 + \lambda \|w\|_2^2 \right\}$

- L2 Regularization term: $\lambda \|w\|_2^2$
- Loss function of squared errors: $\sum_i (w^T x_i - y_i)^2$

Step 1: Minimize $\arg\min \left\{ \sum_i (w^T x_i - y_i)^2 + \lambda \|w\|_2^2 \right\}$ — Definitions

(i) w: model we're trying to learn
x: feature vectors
y: output

(ii) $w^T x_i$: prediction
y_i: actual output

(iii) $\lambda \|w\|_2^2$: L2 regularization term

∴ we're trying to prevent overfitting with a regularization term (independent of the data and solely a function of the parameters) which can help minimize the error

- $\arg\min_{w \in \mathbb{R}} \left\{ E(w) + \lambda E_R(w) \right\} = \arg\min \left\{ \sum (w^T x_i - y_i)^2 + \lambda \|w\|_2^2 \right\}$ → regularization term where $\lambda \geq 0$

Step 2: Take derivative of loss of 'sum of squared errors' and set it to zero

- $g(w) = \arg\min_{w \in \mathbb{R}} \left\{ E(w) + \lambda \|w\|^2 \right\}$ where $E(w)$ is the cost function $\sum_i (w^T x_i - y_i)^2$

$$\frac{dg(w)}{dw} = 0 \; ; \quad -x^T(y - xw) + 2\lambda w = 0$$

$$2\lambda w = x^T(y - xw)$$

$$2\lambda w = x^T y - x^T x w$$

$$w(2\lambda + x^T x) = x^T y$$

$$\boxed{w = \frac{x^T y}{2\lambda + x^T x}} \quad \text{where } \lambda \geq 0$$

(b) Part 2: How does this change the solution to the original linear regression solution?

∴ The original minimization of the squared loss sum looks as follows:

(i) $L(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2 = (y - Xw)^T (y - Xw)$ $\quad X \in \mathbb{R}^{n \times p}$

(ii) Take $\frac{dL(w)}{dw} = 0$; $\frac{d\,L(w)}{dw} = -X^T(y - Xw) = 0$

(iii) Solve for w; $w = (X^T X)^{-1} X^T y$

- This is very similar to the newly captured w value of $\frac{X^T y}{2\lambda + X^T X}$. The main difference is that we have 2λ terms in the denominator which will affect the overall value of w. Since $\lambda \geq 0$, we can assume that this will limit the capacity of the value w can take on (i.e. we're limiting w from getting to large).

∴ L2 Regularization constraining w from going to point p

(c) Part 3: What is the impact of adding this penalty?

∴ This is partly answered above in part 2; when we add in the penalty (i.e. L2 regularization term) we are constraining how large w can get. The λ term will ultimately dictate how much penalty will get enforced on w and if is given to be greater than or equal to 0.

(2) Density estimation: In k-nearest neighbors (KNN), the classification is achieved by majority vote in the vicinity of data. Suppose there are two classes of data each of $n/2$ points overlapped to some extent in a 2D space. Describe what happens to the training error (using all available data) when the neighbor size K varies from n to 1.

(a) Part 1: Describe what happens to the training error (using all available data) when the neighbor size K varies from n to 1

Step 1: Describe k-nearest neighbors procedure

- Procedure

(i) Select an initial volume around x that we want to estimate $p(x)$

(ii) Grow the region until K samples fall inside the region

(iii) $p(x) = \frac{(K/n)}{V}$

- (iii) $p(x) = \frac{(k/n)}{V}$ describes the probability of x (i.e. $p(x)$) is equal to the total number of points in the region (k) divided by the total number of samples in data (n) divided by the volume of the region (V).

∴ Furthermore if $p(x)$ is the convergence point then:

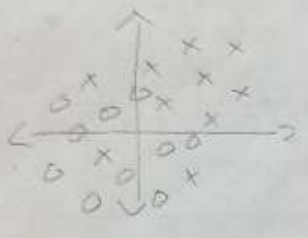
(iv) Assuming that the space will converge to $p(x)$ the region will shrink uniformly: $\lim_{n\to\infty} V_n = 0$

(v) Secondly, $\lim_{n\to\infty} k_n = \infty$

(vi) Lastly, $\lim_{n\to\infty} \frac{k_n}{n} = 0$

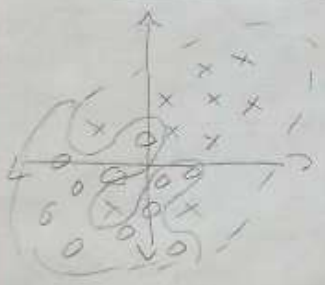
Hw 2: Describe what happens to the training data as we vary k from 1 to n

Example: Two classes, $C1 = O$ & $C2 = X$



- $C1 = 10$ training data points plotted ($n/2$)
- $C2 = 10$ training data points plotted ($n/2$)

- First case: $k = 1$



- When $k=1$ the training error is 0, but as we can see the testing error and overfitting is quite large

- Second case: $k = 10$

- When $k=10$ we start to get training error. In the example we misclassify 4 data points out of 20 total points (10 that belong to C1 and 10 that belong to C2)

- Last case: $k = 20$ (or n)

- When $k=n$ the KNN algorithm will select the all points under the class majority. For example if there are more C1 points than C2 points and $k=n$ then all data will be classified to C1 which will lead to more training data error. Since $k=n$ and C1 & C2 are equally distributed (i.e. $\frac{n}{2}$) knn will not be able to chose a majority class which will lead to a deadlock problem.

∴ Overall, if k=1 we will have zero training error but potentially large testing error. As k increases from 1 to n we begin to get training error (the training error count will increase (as well as the percentage in this example since each class has n/2 training data points). Even though the training error is increasing there will be a balance point (pareto optimality point) where overfitting & underfitting are normalized and the training error is acceptable. Once k reaches n then all data points will be classified to the majority class; in this example it's 50% class 1 & 50% class 2, so the algorithm will have to arbitrarily decide which classification to give all of the data points.

* when k is to low => overfitting of data and small training error
* when k is to large => larger training error and potentially underfitting

(3) Feature selection & Preprocessing: A friend asks you for some help with a feature selection project. Your friend goes out and collects O data for their project. Using O, your friend tries many subsets F ⊂ X by adapting F based on the error of a classifier. They return F that corresponds to the smallest classification error. After they have the feature set F they perform randomized trials to estimate the accuracy of the model using their feature selector. This procedure they carry out to validate the impact of the feature selection routine. This procedure is repeated:

- Make a new data set O with F features using the feature selection routine
- Repeat 50 times
  - Split O into randomized training and testing sets (80/20% split)
  - Train a classifier and record its error
- Report the error averaged over 50 trials

Critique and respond to how your friend performed their analysis

---

(a) Part 1: Critique and respond to how your friend performed their analysis

Step 1: Describe Feature selection Process

∴ Feature selection is ultimately the task of reducing the feature set to a smaller one that contains only relevant & potentially non-redundant features.

- In an example: $x \in \mathbb{R}^{n \times p}$ (collection of data)
  - $y \in \{\pm 1\}$ (two different states)
  - #p variables (instance for the state of the system)
    - -> Relevancy
    - -> Irrelevancy
    - -> Redundancy

- In this example we want to eliminate redundant and irrelevant data and so there are various different feature selection methods we can employ

(i) Wrapper: Search through a sub-space of features using training/validation accuracy of a particular classifier

→ SVM-RFE (Recursive Feature Elimination)
- Additionally this algorithm can train a feature set and add or remove it based on its relevancy factor
→ Downside: its computationally expensive

(ii) Embedded: Exploit the structure of a class of models to guide the feature selection process

→ Lasso: $\arg\min \|y - X^T w\|_2^2 + \lambda \|w\|_1$; $\|w\|_1 = \sum_{j=1}^{r} |w_j|$; $y = w^T x$
- Applying L2 penalty to our margin of the output and predicted result
- We can perform variable selection this way

(iii) Filter: Evaluate a classifier - independent source to evaluate feature relevance/redundancy

→ Filter assumption: (1) Pick good features
(2) Build the classifier

Step 2: Provide feedback based on step 1

- The critique: My friend does his due diligence by collecting many subsets of FCX based on the data captured D to assess the error of the classifier. He also adjusts F based on the error providing for a small feedback loop to minimize the error on the classifiers. He then returns the feature set with the smallest error. After this, the friend attempts to validate the accuracy of the model by performing the following steps:

(1) Make new data set S with F features
(2) Repeat 50 times
- Split D into random training and testing sets
- Train classifier and record its error
(3) Report error averaged over 50 trials.

Overall this process is thorough but time consuming. Instead of recursively collecting new subsets to reduce the error (which is similar to step 1-(i) Wrapper feature selection method) my friend could have employed an embedded approach to minimize the data collection function by also applying an L2 regularization term to constrain the margin of the results. This would save him time selecting the appropriate feature set to validate the accuracy of the model. Additionally, this would reduce the necessity to run the validation steps 50 times and report the error averaged over 50 trials because constraining the feature selection with the embedded method we reduce training error as well as normalize potential overfitting or underfitting of data on the model.