COMS 4771 Machine Learning (Spring 2018)
Problem Set #4

Jianfu Yang - jy2863 Wenda Xu - wx2195 Fan Yang - fy2232
- wx2195@columbia.edu

April 16, 2018

Problem 1

(a) Firstly, consider the case that $d < n$. The VC dimension of $\mathcal{F}_d$ should be $d+1$. We will construct a set to show that $d+2$ examples couldn't be splitted. The set could be shown as:

Dimension	1	2	$\cdots$	$d+1$	*Label*
x_1	1	0	$\cdots$	0	1
x_2	0	1	$\cdots$	0	1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_{d+1}	0	$\cdots$	$\cdots$	1	1
x_{d+2}	0	$\cdots$	$\cdots$	0	0

Each feature other than these $d+1$ features are all same for these $d+2$ examples. Hence the decision tree could only rely on these $d+1$ features. But when pick out arbitrary d features out of these $d+1$ features, there will be always two examples have exactly same d features with opposite labels (one is x_{d+2}, another will come from the first $d+1$ examples based on which d features are chosen). Hence, these $d+2$ examples couldn't be splitted by a d-depth tree. But if only $d+1$ examples are given, since a d-depth tree has at least $d+1$ leaves, these $d+1$ examples must be able to be splitted. Thus, the VC dimension of $\mathcal{F}_d$ is $d+1$.
Now consider the case that $d \geqslant n$. In this case, we don't have the $(d+1)$th feature. The number of all possible data points is 2^n, and they could be located at the leaves of the d-depth full binary tree. Hence, in this case, the VC dimension is 2^n.

(b) (i) Denote X as the number of correct matching points among the m examples. X obeys binominal ditribution. Hence,

$$P_{(x_i,y_i)_i^m \sim \mathcal{D}}[\forall i \ f(x_i) = y_i \mid f \ is \ \epsilon - bad] = P(X = m) < C_m^m (1-\epsilon)^m = (1-\epsilon)^m$$

Below we will denote $P[(i)]$ as $P_{(x_i,y_i)_i^m \sim \mathcal{D}}[\forall i \ f(x_i) = y_i \mid f \ is \ \epsilon - bad]$. We have $P[(i)] < (1-\epsilon)^m$.

(ii) For $\mathcal{F}$, despite f^*, all other classifiers are ϵ-bad for a fixed non-negative small enough ϵ. That is, if there are k (k should be no less than 1) good classifiers:

$$P[f \ is \ \epsilon - bad] = \frac{|\mathcal{F}| - k}{|\mathcal{F}|} \leqslant \frac{|\mathcal{F}| - 1}{|\mathcal{F}|}$$

Then, among these all classifiers,

$$\begin{aligned} P[\forall i \ f(x_i) = y_i] &= P[f = f^*] * 1 + P[f \ is \ \epsilon - bad] * P[\forall i \ f(x_i) = y_i \mid f \ is \ \epsilon - bad] \\ &= \frac{k}{|\mathcal{F}|} + \frac{|\mathcal{F}| - k}{|\mathcal{F}|} \times P[(i)] \quad (P[(i)] < 1) \\ &\geqslant \frac{1}{|\mathcal{F}|} + \frac{|\mathcal{F}| - 1}{|\mathcal{F}|} \times P[(i)] \end{aligned}$$

Now, we could calculate the probability of, f is ϵ-bad while $\forall i \ f(x_i) = y_i$, using Bayes theorem:

$$\begin{aligned} P[f \ is \ \epsilon - bad \mid \forall i \ f(x_i) = y_i] &= \frac{P[\forall i \ f(x_i) = y_i \mid f \ is \ \epsilon - bad] P[f \ is \ \epsilon - bad]}{P[\forall i \ f(x_i) = y_i]} \\ &\leqslant \frac{\frac{|\mathcal{F}|-1}{|\mathcal{F}|} \times P[(i)]}{\frac{1}{|\mathcal{F}|} + \frac{|\mathcal{F}|-1}{|\mathcal{F}|} \times P[(i)]} \\ &(This \ is \ positive \ correlation \ with \ P[(i)]) \\ &(P[(i)] < (1-\epsilon)^m) \\ &< \frac{\frac{|\mathcal{F}|-1}{|\mathcal{F}|} \times (1-\epsilon)^m}{\frac{1}{|\mathcal{F}|} + \frac{|\mathcal{F}|-1}{|\mathcal{F}|} \times (1-\epsilon)^m} \end{aligned}$$

This is what we need.

(iii) From part(ii), what we want is,

$$\begin{aligned} P[f \ is \ \epsilon - bad \mid \forall i \ f(x_i) = y_i] < \frac{\frac{|\mathcal{F}|-1}{|\mathcal{F}|} \times (1-\epsilon)^m}{\frac{1}{|\mathcal{F}|} + \frac{|\mathcal{F}|-1}{|\mathcal{F}|} \times (1-\epsilon)^m} &< \delta \\ (1-\epsilon)^m < \frac{\delta}{(1-\delta)(|\mathcal{F}|-1)} \\ m \ln(1-\epsilon) < \ln \frac{\delta}{(1-\delta)(|\mathcal{F}|-1)} \qquad &(Both \ sides \ are \ negative) \\ m > \frac{1}{\ln \frac{1}{1-\epsilon}} \ln \frac{(1-\delta)(|\mathcal{F}|-1)}{\delta} \end{aligned}$$

Comparing with the Occam's Razor bound, $\frac{1}{\ln \frac{1}{1-\epsilon}}$ replaces $\frac{1}{\epsilon^2}$. For $0 < \epsilon < 1$, we always have $\frac{1}{\ln \frac{1}{1-\epsilon}} < \frac{1}{\epsilon}$, thus when ϵ is small, $\frac{1}{\ln \frac{1}{1-\epsilon}} \ll \frac{1}{\epsilon^2}$. And when δ is small and $|\mathcal{F}|$ is large, $\frac{(1-\delta)(|\mathcal{F}|-1)}{\delta} \sim \frac{|\mathcal{F}|}{\delta}$. Hence, the new bound is sharper than Occam's Razor bound.

(c) We draw $n' = kn$ samples from the distribution and partition them into k batches. Then use algorithm $\mathcal{A}$ to get k models on each batch. These k models form a hypothesis space $\mathcal{F}_n^{\mathcal{A}}$. For each $f_n^{\mathcal{A}}$ we have,

$$\mathbb{P}[err(f_n^{\mathcal{A}}) - err(f^*) \geqslant \epsilon] = 0.45$$

Denote

$$f_n^{\mathcal{A}^*} = \arg\min_{f_n^{\mathcal{A}} \in \mathcal{F}_n^{\mathcal{A}}} err(f_n^{\mathcal{A}})$$

If any $f_n^{\mathcal{A}}$ satisfies $err(f_n^{\mathcal{A}}) - err(f^*) \leqslant \epsilon$, we must have $err(f_n^{\mathcal{A}^*}) - err(f^*) \leqslant \epsilon$. Hence, we have

$$\mathbb{P}[err(f_n^{\mathcal{A}^*}) - err(f^*) \geqslant \epsilon] = 0.45^k$$
$$\mathbb{P}[err(f_n^{\mathcal{A}^*}) - err(f^*) \leqslant \epsilon] = 1 - 0.45^k$$

If we could find an algorithm to choose $f_{n'}^{\mathcal{B}} = f_n^{\mathcal{A}^*}$, then for all $\delta > 0$, we could find $k = \Theta(\ln \frac{1}{\delta})$ to make

$$\mathbb{P}[err(f_{n'}^{\mathcal{B}}) - err(f^*) \leqslant \epsilon] = 1 - 0.45^k \geqslant 1 - \delta$$

which means

$$n' = nk = O(\frac{1}{\epsilon^2} \ln \frac{1}{\delta})$$

Here, we choose to use ERM to choose $f_{n'}^{\mathcal{B}}$ from $\mathcal{F}_n^{\mathcal{A}}$. That is,

$$f_{n'}^{\mathcal{B}} = \arg\min_{f_n^{\mathcal{A}} \in \mathcal{F}_n^{\mathcal{A}}} err_{n'}(f_n^{\mathcal{A}})$$

From class material, we have

$$\mathbb{P}[err(f_{n'}^{\mathcal{B}}) - err(f_n^{\mathcal{A}^*}) \leqslant \epsilon] = 1 - \delta'$$

The bound on n', δ' and ϵ should be

$$\delta' \geqslant 2|\mathcal{F}_n^{\mathcal{A}}|e^{-2\epsilon^2 n'} = 2ke^{-2\epsilon^2 nk} \tag{i}$$

The right part of the inequality is decreasing for $k \geqslant 1$. That is, the larger k is, the smaller δ' we could take.
Back to $err(f_{n'}^{\mathcal{B}}) - err(f^*)$, we have

$$err(f_{n'}^{\mathcal{B}}) - err(f^*) = [err(f_{n'}^{\mathcal{B}}) - err(f_n^{\mathcal{A}^*})] + [err(f_n^{\mathcal{A}^*}) - err(f^*)]$$
$$\begin{aligned}\mathbb{P}[err(f_{n'}^{\mathcal{B}}) - err(f^*) \geqslant \epsilon] &\leqslant \mathbb{P}[err(f_{n'}^{\mathcal{B}}) - err(f_n^{\mathcal{A}^*}) \geqslant \epsilon] + \mathbb{P}[err(f_n^{\mathcal{A}}) - err(f^*) \geqslant \epsilon] \\ &= \delta' + 0.45^k\end{aligned}$$

Now what we need is

$$\begin{aligned}\mathbb{P}[err(f_{n'}^{\mathcal{B}}) - err(f^*) \leqslant \epsilon] &\geqslant 1 - \delta \\ 1 - \delta' - 0.45^k &\geqslant 1 - \delta \\ \delta' &\leqslant \delta - 0.45^k\end{aligned} \tag{ii}$$

Combine (i) and (ii), the bound of k should satisfy

$$\delta - 0.45^k \geqslant \delta' \geqslant 2ke^{-2\epsilon^2 nk}$$

Since $n = O(\frac{1}{\epsilon^2})$, it could be rewritten as

$$\delta - 0.45^k \geqslant 2ke^{-2Ck} \tag{iii}$$

which only contains k, δ and constant. Now, the question becomes whether the bound of k derived from (iii) is $poly(\frac{1}{\delta})$. We could take $k = O(\frac{1}{\delta})$ to exam:

$$f(\delta) = \delta - 0.45^{1/\delta} - \frac{2}{\delta}e^{-2C/\delta}$$

This is larger than 0 for $0 < \delta < 1$ with proper C (for example, take $C = 2$).
The final bound on n' would be $O(\frac{1}{\delta\epsilon^2})$, which is $poly(\frac{1}{\epsilon}, \frac{1}{\delta})$.

Problem 2

(a) The minimum value of (1) would be 0. That is, each point forms a cluster only containing itself. In this case, $k = n$, and $\boldsymbol{c}_i = \boldsymbol{x}_i$ for all i.

(b) (i) When $k = 1$, there will be only one cluster, then (1) could be rewrite as:

$$f = \sum_{i=1}^{n} ||\boldsymbol{x}_i - \boldsymbol{c}||^2$$

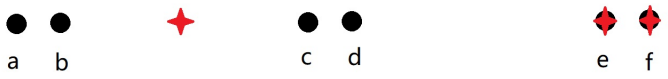
This function is convex, hence we could get its minimum value by taking derivation:

$$\frac{df}{d\boldsymbol{c}} = \sum_{i=1}^{n} 2(\boldsymbol{c} - \boldsymbol{x_i}) = 0$$

$$\boldsymbol{c} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x_i}$$

That is, $\boldsymbol{c}$ is the average of $\boldsymbol{x}$. The minimum value of the objective functoin will be the variance of $\boldsymbol{x}$.

(ii) Consider the following dataset consists of six black points:



The optimal three clusters should be: $\{a, b\}$, $\{c, d\}$ and $\{e, f\}$. But if the initial three centers locate as the red stars in the picture, it will reach a suboptimal state: $\{a, b, c, d\}$, $\{e\}$ and $\{f\}$. After this, no change will occur and the algorithm stops.

(iii) The python code is shown below:

```
def Loss(A,i,j):
    m=0
    for k in range(i-1,j):
        m+=A[k]
    m/=(j+1-i)
    sum=0
    for k in range(i-1,j):
        sum+=(A[k]-m)**2
    return sum

def optimal(X,k):
    A=sorted(X)
    n=len(A)
    inf=65535
    L=[[0 for i in range(n+1)] for i in range(k+1)]
    J=[[[] for i in range(n+1)] for i in range(k+1)]
    for i in range(1,n+1):
```

```
        L[1][i]=Loss(A,1,i)
        J[1][i].append(1)
    for m in range(2,k+1):
        for i in range(m,n+1):
            min=inf
            for j in range(m,i+1):
                if L[m-1][j-1]+Loss(A,j,i)<min:
                    min=L[m-1][j-1]+Loss(A,j,i)
                    j_optimal=J[m-1][j-1]+[j]
            L[m][i]=min
            J[m][i]=j_optimal[:]
    clusters=[[] for i in range(k)]
    for i in range(k-1):
        for j in range(J[k][n][i],J[k][n][i+1]):
            clusters[i].append(A[j-1])
    for j in range(J[k][n][k-1],n+1):
        clusters[k-1].append(A[j-1])
    return clusters
```

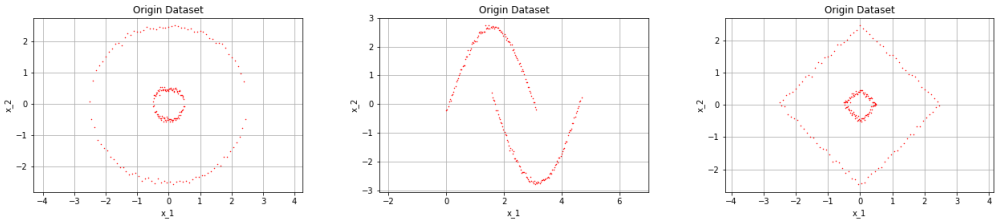
The input is X $(x_1 \,..\, x_n)$, firstly sort it to form a non-descending sequence. Denote this sequence as A.
Then, finding the clusters could be solved as a dynamic programming problem. Denote $L[m,i]$ as the minimum loss (i.e., the total square distance) of the first i elements with m clusters. And denote $J[m,i]$ as the corresponding list of indexes of first element of each cluster. Hence, what we need is $L[k,n]$ and $J[k,n]$.
Firstly, we initialize $L[1,i]$ for $1 \leqslant i \leqslant n$ by simply getting the mean of the first i elements and then calculating the sum of the square distances.
Note that $L[m,i]$ could be calculated from $L[m-1,j-1]$ for $m \leqslant j \leqslant i$ (the number of clusters must be no larger than the number of elements):

$$L[m,i] = \min_{j:\ m \leqslant j \leqslant i} (L[m-1,j-1] + Loss(A,j,i))$$

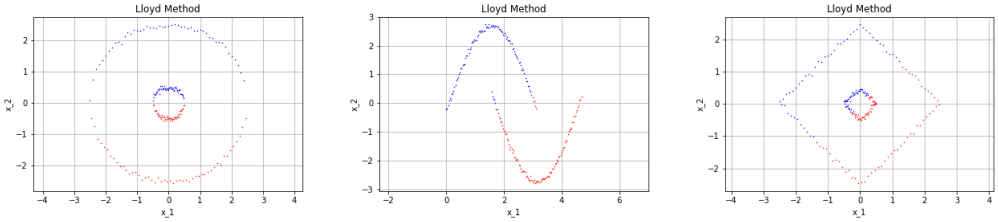
Here, $Loss(A,j,i)$ is the loss of $x_j \,..\, x_i$, i.e., we assign $x_j \,..\, x_i$ as the mth cluster. Since $L[m-1,j-1]$ is already optimal for the first $j-1$ elements in $m-1$ clusters, we go through all possible values of j and then could find the minimal loss of the first i elements in m clusters. Also, we record the index of the first element of the m clusters in $J[m,i]$.
After going through all i and m, we could get the optimal solution by $L[k,n]$ and $J[k,n]$. Finally, use $J[k,n]$ to construct the clusters.

For running time, the sorting will be $O(n \log n)$; the initialization of K and J will be $O(nk)$; the initialization of $L[1]$ will be $O(n^2)$ ($Loss(A,j,i)$ is $O(n)$). Then for DP part, m-loop will take $O(k)$ times, i-loop will take $O(n)$ times for each m-loop, and j-loop will take $O(n)$ times for each i-loop. That is, j-loop will take $O(kn^2)$ times. For each j-loop, $Loss(A,j,i)$ will be $O(n)$ and the copy of $J[m-1,j-1]$ will cost $O(n)$. Hence, the total running time of DP part will be $O(kn^3)$. The finally cluster construction will be $O(n)$. Hence, the total running time will be $O(kn^3)$.

(c) (i) The origin datasets are:



Applying Lloyd methods, the resulting clusters are:



The ideal resulting clusters will be shown in the last part.

(ii) The jth center should locate at the average position of the points contained in it. That is,

$$\boldsymbol{c_j} = \frac{1}{\sum_{i=1}^{n} z_{ij}} \sum_{i=1}^{n} z_{ij}\phi(\boldsymbol{x_i})$$

Hence,

$$\alpha_{ij} = \frac{z_{ij}}{\sum_{k=1}^{n} z_{kj}}$$

Proven.

(iii)

$$\begin{aligned} ||\phi(\boldsymbol{x_i}) - \phi(\boldsymbol{x_j})||^2 &= \phi(\boldsymbol{x_i})^2 - 2\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}) + \phi(\boldsymbol{x_j})^2 \\ &= K(\boldsymbol{x_i}, \boldsymbol{x_i}) - 2K(\boldsymbol{x_i}, \boldsymbol{x_j}) + K(\boldsymbol{x_j}, \boldsymbol{x_j}) \end{aligned}$$

Proven.

(iv)

$$\begin{aligned} ||\phi(\boldsymbol{x_i}) - \boldsymbol{c_j}||^2 &= \phi(\boldsymbol{x_i})^2 - \frac{2\sum_{k=1}^{n} z_{kj}\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_k})}{\sum_{k=1}^{n} z_{kj}} + \frac{\sum_{i,k=1}^{n} z_{ij}z_{kj}\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_k})}{(\sum_{k=1}^{n} z_{kj})^2} \\ &= K(\boldsymbol{x_i}, \boldsymbol{x_i}) - \frac{2\sum_{k=1}^{n} z_{kj}K(\boldsymbol{x_i}, \boldsymbol{x_k})}{\sum_{k=1}^{n} z_{kj}} + \frac{\sum_{i,k=1}^{n} z_{ij}z_{kj}K(\boldsymbol{x_i}, \boldsymbol{x_k})}{(\sum_{k=1}^{n} z_{kj})^2} \quad (I) \end{aligned}$$

Proven.

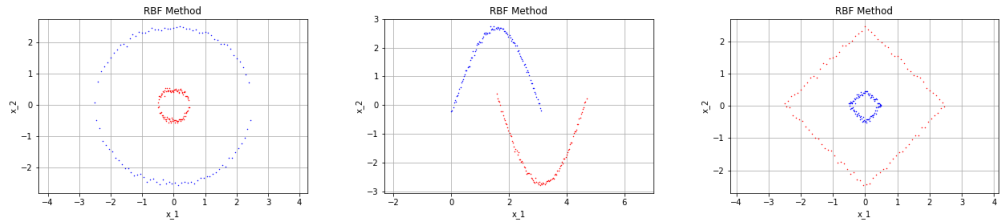
(v) The pseudo code is shown below:

KERNEL-KMEANS($x_{1..n}$, k)

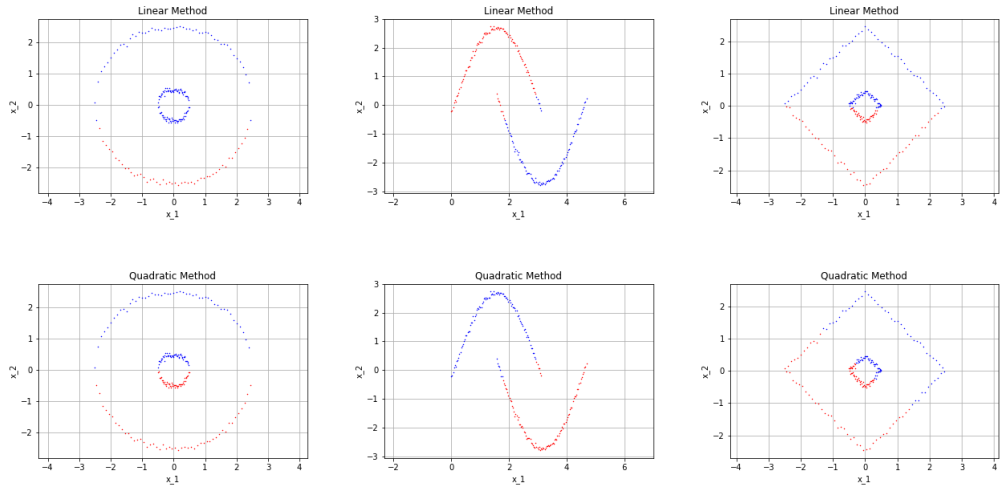
1 Calculate kernel matrix K from $x_{1..n}$
2 Randomly initialize z_{ij} for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant k$, make sure for each i, only one $z_{ij} == 1$
3 **while** not converge:
4 $\quad n_j = \sum_{k=1}^{n} z_{kj}$
5 $\quad s_{2j} = \sum_{i,k=1}^{n} z_{ij} z_{kj} K(\boldsymbol{x_i}, \boldsymbol{x_k})$
6 $\quad$ **for** $i = 1$ **to** n:
7 $\quad\quad s_{1j} = \sum_{k=1}^{n} z_{kj} K(\boldsymbol{x_i}, \boldsymbol{x_k})$
8 $\quad\quad j_i^* = \arg\min_j (K(\boldsymbol{x_i}, \boldsymbol{x_i}) - 2s_{1j}/n_j + s_{2j}/n_j^2)$
9 $\quad$ Update z_{ij} using j_i^*
10 **return** z_{ij}

The inputs are $x_{1..n}$ and k (the number of clusters). Output z_{ij} denotes the cluster assignment of each point.

(vi) The resulting clusters will depend on the initial state of z_{ij}. After some attempts, we get the resulting clusters of rbf kernel as:



We also try linear and quadratic kernels (linear kernel is Manhattan distance, and quadratic kernel is the square of Euclidean distance). The resulting clusters are:



We can see that quadratic kernel is almost the same with Lloyd method. And rbf kernel performs very well.

Problem 3

(i) From this positive example, each time change one feature to the opposite (i.e., 0 for origin 1 in this example, and 1 for 0). For example, if the positive example is $\{101\}$, then we will require $\{001\}$, $\{111\}$ and $\{100\}$. For each new example, if the label is still positive, then this feature doesn't influence the predictiton, i.e., this feature isn't contained in the true hypothesis. Else, if the label changes to 0, then this feature is in true hypothesis. In this case, if this feature (x_i) is 1 in the positive example, then x_i is in the true hypothesis; if this feature is 0 in the positive example, then $\neg x_i$ is in the true hypothesis. Thus, after d queries on each feature, we could get the true hypothesis.

(ii) In this question, since the size of $\mathcal{F}$ is finite, we could apply Occam's Razor principle. Assume k of d features participate in the true hypothesis, then the number of choices of the k features would be C_d^k. Once m features are chosen, there will be 2^k hypotheses since each featue has two possible value. That is, if the true hypothesis contains k features, there will be $C_d^k 2^k$ hypotheses. Then for all possible k, the size of $\mathcal{F}$ will be:

$$\begin{aligned} |\mathcal{F}| &= \sum_{k=0}^{d} C_d^k 2^k \\ &= \sum_{k=0}^{d} C_d^k 2^k 1^{d-k} \\ &= (2+1)^d = 3^d \end{aligned}$$

Applying Occam's Razor principle,

$$\begin{aligned} m &\geqslant C \cdot \frac{1}{\epsilon^2} \ln \frac{|\mathcal{F}|}{\delta} \\ &= C \cdot \frac{1}{\epsilon^2} \ln \frac{3^d}{\delta} \end{aligned}$$

That is, we need $O(\frac{1}{\epsilon^2} \ln \frac{3^d}{\delta})$ examples.

(iii) Without loss of generality, assume that the true hypothesis is a conjunction of the first k features. And S_1 dontes the sequence of positive label of these k features, S_2 denotes one of the 0 label sequence. Let the 2^{d-k} examples contain all possible sequences of the last $d-k$ features (this makes sure that all examples are unique). The first k features of these examples with $x_{k+1} = 1$ will be S_1, and the first k features of these examples with $x_{k+1} = 0$ will be S_2. Then, in these examples, the label will be 1 for $x_{k+1} = 1$ and 0 for $x_{k+1} = 0$. Then $f = x_{k+1}$ will be consistent on these 2^{d-k} examples. Then apply this function to all the data points, among the 2^{d-k} points which should be labeled 1, there will be half of them being labeled 0; among the $2^d - 2^{d-k}$ points which should be labeled 0, there will be half of them being labeled 1. That is, the classification error is 50%.

All above assumes that $k < d$. Then if $k = d$, only one example will be given, let

this example be S_1. Then $f = 1$ will be consistent to this example, but it will cause a classification error larger than 50% on the whole set of data points.

Problem 4

(i) All terms which contain $\boldsymbol{x}_i$ are:

$$\sum_{j:\ j\neq i}(||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij})^2+\sum_{j:\ j\neq i}(||\boldsymbol{x}_j-\boldsymbol{x}_i||-D_{ji})^2+(||\boldsymbol{x}_i-\boldsymbol{x}_i||-D_{ii})^2$$

$$=2\sum_{j:\ j\neq i}(||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij})^2$$

Hence, the derivative of the discrepancy function with respect to a location $\boldsymbol{x}_i$ is:

$$\frac{\partial f}{\partial \boldsymbol{x}_i}=2\sum_{j:\ j\neq i}\frac{\partial}{\partial \boldsymbol{x}_i}(||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij})^2$$

$$\frac{\partial f}{\partial x_{i1}}=2\sum_{j:\ j\neq i}\frac{\partial}{\partial x_{i1}}(\sqrt{(x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2}-D_{ij})^2$$

$$=4\sum_{j:\ j\neq i}(||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij})\frac{\partial}{\partial x_{i1}}\sqrt{(x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2}$$

$$=2\sum_{j:\ j\neq i}\frac{||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij}}{||\boldsymbol{x}_i-\boldsymbol{x}_j||}\frac{\partial}{\partial x_{i1}}(x_{i1}-x_{j1})^2$$

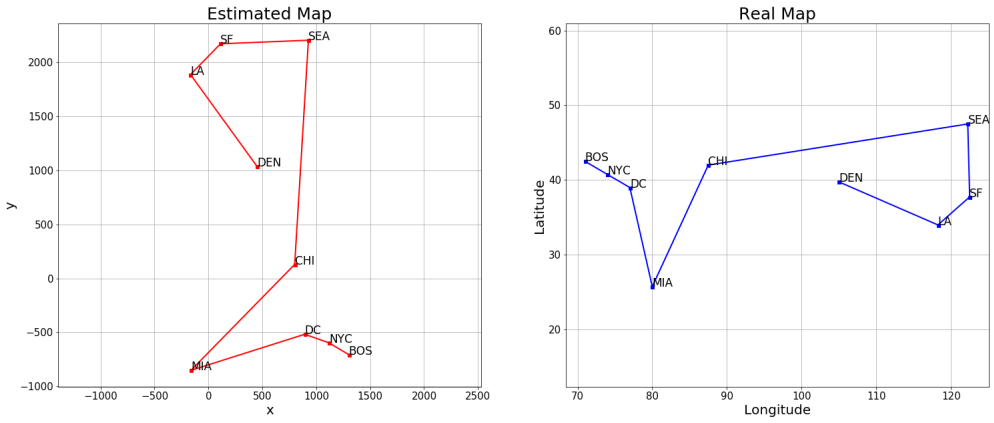
$$=4\sum_{j:\ j\neq i}\frac{||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij}}{||\boldsymbol{x}_i-\boldsymbol{x}_j||}(x_{i1}-x_{j1})$$

It's similar for x_{i2}. Hence,

$$\frac{\partial f}{\partial \boldsymbol{x}_i}=4\sum_{j:\ j\neq i}\frac{||\boldsymbol{x}_i-\boldsymbol{x}_j||-D_{ij}}{||\boldsymbol{x}_i-\boldsymbol{x}_j||}(\boldsymbol{x}_i-\boldsymbol{x}_j)$$

(ii) We use gradient descent method to get the location of cities. We randomly choose the starting point of each city.

(iii) The estimated map and the real map (drawn using lon/lat information) is shown below:



Since the initial position of each city is randomly selected, the estimated map will only be able to depict the relative location. And we can see that the estimated relative location is similiar with the real location.