# Homework-4

Given

Design matrix = $X$

Covariance matrix = $C = \frac{1}{n} X^T X$

Eigenvectors: $V_1, V_2 \ldots V_k$

$\tilde{x_i}$ is the orthogonal projection of $x_i$, Column vector representing a data point, orthogonal to $V_1$

$$\tilde{x_i} = (I - V_1 V_1^T) x_i$$

$\tilde{X}_T = [\tilde{x_1}, \ldots \tilde{x_n}]$ deflated matrix, lies in the direction of the first principal eigenvectors

$$\tilde{X}_T = (I - V_1 V_1^T) X^T$$

$$C = \frac{1}{n} X^T X = V \Lambda_n V^T$$

Let's suppose that the kernel matrix is $K = XX^T$

So, $K = XX^T = U \Lambda_m U^T$

The columns $v_i$ of the orthonormal matrix $V$ are the eigenvectors of $C$ and the columns $u_i$ of the orthonorm

matrix U are the eigenvectors of K. Considering an eigen-vector eigen-value pair $u, \lambda$ of K.

$$C = \frac{1}{n}(X^T U) = X^T X X^T U = X^T K U = \lambda X^T U$$

$X^T U$, $\lambda$ is an eigenvector-eigenvalue pair for C.

$$\|X^T U\|^2 = U^T X X^T U = \lambda$$

So the normalised eigenvectors of C is $V_i = \lambda^{-\frac{1}{2}} X^T U$.

$$U = \lambda^{-\frac{1}{2}} X V_i$$

Deflating X,

$$\tilde{X} = X - UU' X = X - \lambda^{\frac{1}{2}} U V_i' = X - X V_i V_i'$$

$$\tilde{C} = \tilde{X}^T \tilde{X} = \frac{1}{n}(X - UU^T X)^T (X - UU^T X)$$

$$= \frac{1}{n}(X^T X - X^T U U^T X)$$

$$\boxed{= \frac{1}{n}(X^T X - \lambda V_i V_i^T)}$$

(10) $C v_j = \lambda_j v_j$

$\tilde{C} = C - \lambda_j v_j v_j^T$

$(C - \lambda_j v_j v_j^T) v_j$

$= C v_j - \lambda_j v_j v_j^T v_j$

$= \lambda_j v_j - \lambda_j v_j (v_j^T v_j)$

if $j \neq 1$, then $(C - \lambda_j v_j v_j^T) v_j = \lambda_j v_j - \lambda_j v_j(0) = \lambda_j v_j$

Thus $(C - \lambda_j v_j v_j^T) = \tilde{C}$ has the same eigenvectors as $C$ and the same eigenvalues as $C$ except that the largest one has been replaced by 0.

(11) Hotelling deflation assumes that the largest eigenvalue $\lambda_1$ and an associated eigenvector $v(1)$ of $C$ has been obtained from its deflation matrix $\tilde{C}$, which has the same eigenvalues $\lambda_2 \ldots \lambda_j$ as $C$ except that $\tilde{C}$ has eigenvalue 0 with eigenvector $v(1)$ instead of eigenvalue $\lambda_1$.

So if $v$ be the first principal eigenvectors of $\tilde{e}$

$U = V_2$ because the first eigenvectors is zero

(d) After $k$ iterations,

$$C^k U_0 = \lambda_1^k \left[ c_1 v_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \ldots + c_n \left(\frac{\lambda_n}{\lambda_1}\right)^k v_n \right] \quad [U_0 = c_1 v_1 + \ldots + c_n v_n]$$

Since here $\lambda_1$ dominates, the ratio $\left(\frac{\lambda_i}{\lambda_1}\right)^k \to 0$ as $k \to \infty$ for all $i$. Therefore, $C^k U_0 = \lambda_1^k c_1 v_1$ gets better as $k$ grows. Here $c_1 \ldots c_n$ is constant.

So, For large powers of $k$ we will obtain a good approximation of the dominant eigenvector.

For $k$ iteration, the computational complexity is $O(k^2)$.

In each iteration if we do $v_i = \frac{Cv_{i-1}}{\| Cv_{i-1} \|_2}$,

this can be a better approach because the

Powero method tends to produce approximations with large entries. In practice it is best to 'scale down' each approximation before proceeding to the next iteration. The way to accomplish this scaling is to determine the component of Cvi that has the largest absolute value.

e) The computational complexity is: kkm^2

(See code)

Problem 2

a) We have to set $r$ and $c$ such that $\min \|q-x\| \leq r$ and $\|q-x'\| \leq cr$. More specifically, an $r$-near query with constant factor $c$ will return a $j$ such that $\|q-x_j\| \leq cr$ if there is some $i$ so that $\min_i \|q-x_i\| \leq r$.

We can do this by solving nearest neighbor checking $r$ with $r = 1, 2, 4, 8, \ldots R$

We only need to give output if there is a point within $r$, even if something within $cr$.

b) Given,

$h(x_i^\circ) = x_i^\circ[a]$

For each point $x_i^\circ, x_j^\circ \in \{0,1\}^d$.

* $Pr(h(x_i^\circ) = h(x_j^\circ)) \geq P_1$

where $P_1 = 1 - \frac{r}{d} \approx e^{-r/d}$ if $dist(x_i^\circ, x_j^\circ) \leq r$.

* $Pr(h(x_i^\circ) = h(x_j^\circ)) \leq P_2$

where $P_2 = 1 - \frac{cr}{d} \approx e^{-cr/d}$ if $dist(x_i^\circ, x_j^\circ) \geq cr$

Here $\boxed{P_1 > P_2}$

c) $Pr(g(x_i^\circ) = g(x_j^\circ)) \geq P_1^K$ [lower bound] $[d(x_i^\circ, x_j^\circ) \leq r]$

$Pr(g(x_i^\circ) = g(x_j^\circ)) \leq P_2^K$ [upper bound] $[d(x_i^\circ, x_j^\circ) \geq cr]$

d) $Pr(g_b(x_i^\circ) = g_b(x_j^\circ)) \geq 1 - \frac{1}{n}$ [LB] $[d(x_i^\circ, x_j^\circ) \leq r]$

$Pr(g_b(x_i^\circ) = g_b(x_j^\circ)) \leq \frac{1}{n}$ [UB] $[d(x_i^\circ, x_j^\circ) \geq cr]$

e) $K = \frac{\ln(n)}{\ln(1/P_2)}$

This implies that for fixed b, the expected number of $x'$ that map to the same bucket as $q$ is at most $n \times \frac{1}{n} = 1$.

Applying linearity of expectation over all $l$ buckets, the expected number of false positive is at most $l$.

By Markov's inequality the probability there are more than $4l$ false positives is therefore at most $\frac{1}{4}$ and at least $\boxed{\frac{3}{4}}$.

According to first event, for any b

$$\Pr[g_b(x^*) \neq g_b(q)] \leq 1 - P_1^K = 1 - n^{-\rho} = 1 - \frac{1}{n^\rho} \quad [K = \frac{\ln(n)}{\ln(1/P_2)}, \rho = \frac{\ln(P_1)}{\ln(P_2)}$$

As $l = n^\rho$,

$$\Pr[g_b(x^*) \neq g_b(q) \forall i] \leq (1 - \frac{1}{n^\rho})^{n^\rho} \leq \frac{1}{e}$$

So, the first event holds with probability at least $\boxed{1 - \frac{1}{e}}$

With union bound the probability of both event holding at least: $\boxed{1 - \frac{1}{4} - 1/e \geq \frac{1}{3}}$

2f) In expectation, there are $O(L)$ points in the data set which map to same hash functions as q for some K. we need to examine these points to check if any of them within distance $cr_0$ from q

NO, it is not guaranteed that there is a point with distance at most $cr_0$ because the probability of upper bound is very low (only $1/n$).

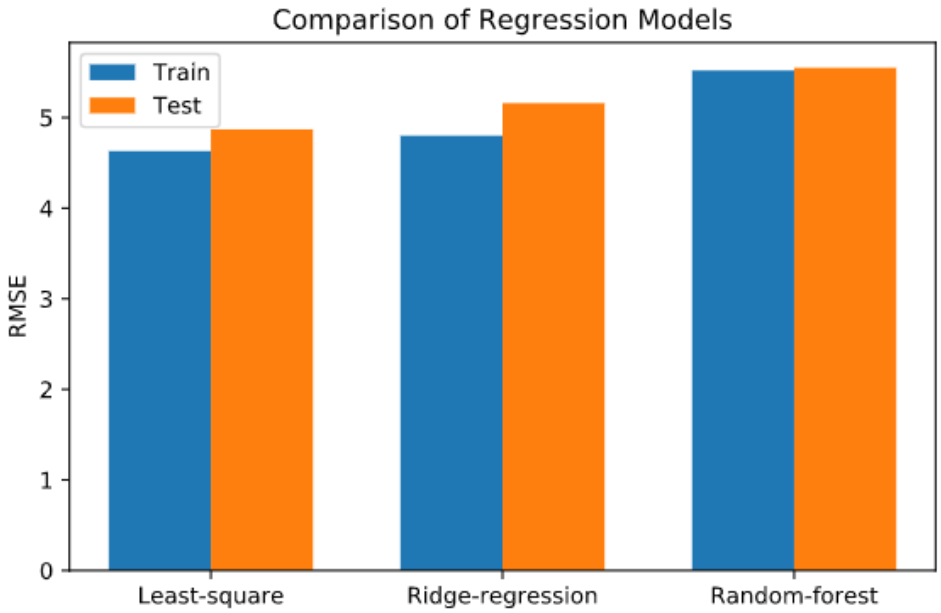
**Problem 3(a)**

Random Forest Boston Housing

Train RMSE: 5.515574182181881

Test RMSE: 5.551093717718717

Performance of RF comparing to least square and regression




From the above plot it is evident that both in train and test cases for Boston housing data Least-Square is performing best than random forest and ridge regression.

**Problem 3(b)**

Credit-g-dataset:

Train Accuracy:  0.7114285714285714

Test Accuracy:  0.7066666666666667

Breast-cancer dataset:

Train Accuracy: 0.8844221105527639

Test Accuracy: 0.9064327485380117

4(a) The computational complexity of optimizing a tree of depth d in terms of m and n : O(mnlogn)

4(b) Applications with high dimensional sparse output requires most expensive computation in GBDT training. GBDT builds a regression tree that fits the residual from the previous tree. So at least O(MN) time and memory are required to build GBDT trees.

To solve this problem, we can solve a $L_0$ regularized optimization problem to enforce the prediction of each leaf node in each tree to have only a small number of non-zero elements/labels.

Problem 4(c)

We can train an individual decision tree in parallel in thread

Training a decision tree is an iterative process that requires looping over all possible ways of splitting the current leaf nodes into new leaf nodes. So each ~~del~~ decision tree takes a lot of time for computation. So these decision trees can be run in parallel threads.

Problem 4(d)

1) In gradient descent the goal of calculating gradients and updating $\theta$ accordingly is to optimize training loss. where in gradient boosting, one can fit a weak classifier to the data and a loss function with respect to classifier.

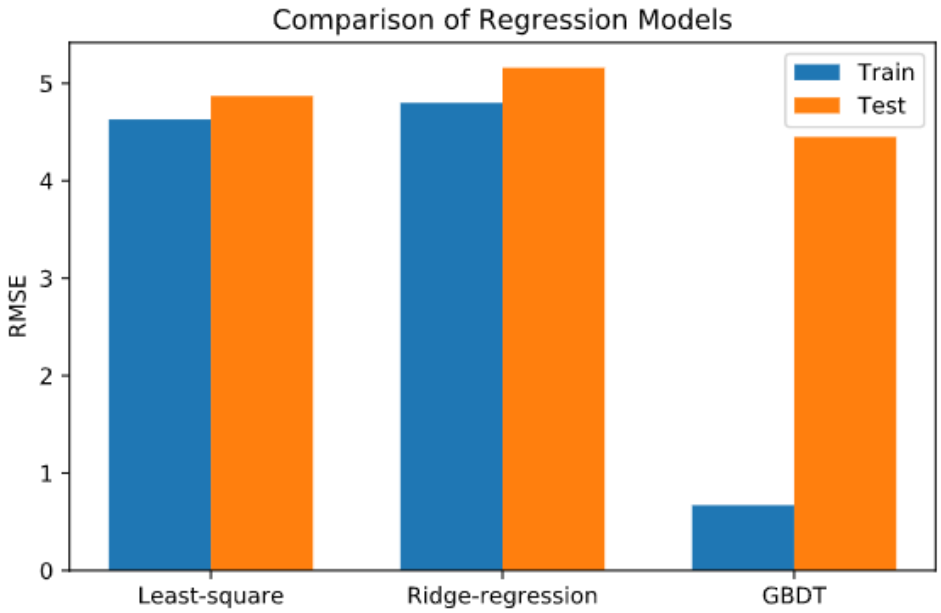
2) Gradient descent descends the gradient by introducing changes to parameters, whereas gradient boosting descends the gradient by introducing new models.

Problem 4(e)

Boston Housing Price GBDT

Train RMSE: 0.6733030286580176

Test RMSE: 4.448467175942173




From the above plot it is evident that both in train and test cases for Boston housing data GBDT is performing best than least-square and ridge regression.

**Problem 4(f)**

Credit-g-dataset:

Train Accuracy: 0.9742857142857143

Test Accuracy: 0.7466666666666667

Breast-cancer dataset:

Train Accuracy: 1.0

Test Accuracy: 0.9707602339181286

**Problem 4(g)**

For all cases, GBDT's accuracy and RMSE score is better than random forest. For RF each iteration the classifier is trained independently from the rest. Where GBDT is a boosting method, which builds on weak classifiers. It adds a classifier at a time, so that the next classifier is trained to improve the already trained ensemble, that makes the GBDT stronger than another classifier.