Homework 4

CSCI 5525 S20: Machine Learning

Due on Apr. 15th 11:59 pm (Midnight)

Please type in your info:

- **Name:** Pranay Patil
  Student ID: 5592264
  Email: patil122@umn.edu

- **Collaborators, and on which problems**

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems, but each person must write up the final solutions individually. You need to fill in above to specify which problems were a collaborative effort and with whom. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help on online.
- Look up things/post on sites like Quora, StackExchange, etc.

Submission. Submit a PDF using this LaTeX template for written assignment part and submit .py files for all programming part. You should upload all the files on Canvas.

Written Assignment

Instruction. For each problem, you are required to write down a full mathematical proof to establish the claim.

Problem 1. Boosting Derivation.(Total 5 points)

We discussed in class that AdaBoost minimizes the exponential loss greedily. In particular, the derivation of α_t is by finding the minimizer of

$$\epsilon_t(\exp(\alpha_t) - \exp(-\alpha_t)) + \exp((-\alpha_t))$$

where ϵ_t is the weighted classification error of h_t. Show that $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ is the minimizer.

Your answer. $Z_t = \epsilon_t(exp(\alpha_t) - exp(-\alpha_t)) + exp(-\alpha_t)$ To find the minimizer for Z_t we differentiate Z_t w.r.t. α_t,

$$Z_t = \epsilon_t(exp(\alpha_t) - exp(-\alpha_t)) + exp(-\alpha_t)$$
$$\frac{\partial Z_t}{\partial \alpha_t} = \frac{\partial}{\partial \alpha_t}[\epsilon_t(exp(\alpha_t) - exp(-\alpha_t)) + exp(-\alpha_t)]$$
$$\frac{\partial Z_t}{\partial \alpha_t} = \frac{\partial}{\partial \alpha_t}[\epsilon_t(exp(\alpha_t) - exp(-\alpha_t))] + \frac{\partial}{\partial \alpha_t}[exp(-\alpha_t)]$$
$$\frac{\partial Z_t}{\partial \alpha_t} = \epsilon_t[\frac{\partial}{\partial \alpha_t}(exp(\alpha_t) - exp(-\alpha_t))] + \frac{\partial}{\partial \alpha_t}[exp(-\alpha_t)]$$
$$\frac{\partial Z_t}{\partial \alpha_t} = \epsilon_t(exp(\alpha_t) + exp(-\alpha_t)) - exp(-\alpha_t)$$

If α_t is the minimizer of Z_t, $\frac{\partial Z_t}{\partial \alpha_t} = 0$

$$\frac{\partial Z_t}{\partial \alpha_t} = 0$$
$$\epsilon_t(exp(\alpha_t) + exp(-\alpha_t)) - exp(-\alpha_t) = 0$$
$$\epsilon_t(exp(\alpha_t) + exp(-\alpha_t)) = exp(-\alpha_t)$$
$$\epsilon_t = \frac{exp(-\alpha_t)}{(exp(\alpha_t) + exp(-\alpha_t))}$$
$$\frac{exp(-\alpha_t)}{exp(\alpha_t) + exp(-\alpha_t)} = \epsilon_t$$
$$\frac{exp(\alpha_t) + exp(-\alpha_t)}{exp(-\alpha_t)} = \frac{1}{\epsilon_t}$$
$$By\ the\ rule\ of\ dividendo$$
$$if\ \frac{a}{b} = \frac{c}{d}$$
$$then\ \frac{a-b}{b} = \frac{c-d}{d}$$
$$\frac{exp(\alpha_t) + exp(-\alpha_t) - exp(-\alpha_t)}{exp(-\alpha_t)} = \frac{1-\epsilon_t}{\epsilon_t}$$
$$\frac{exp(\alpha_t)}{exp(-\alpha_t)} = \frac{1-\epsilon_t}{\epsilon_t}$$
$$exp(2\alpha_t) = \frac{1-\epsilon_t}{\epsilon_t}$$
$$Taking\ log$$
$$2\alpha_t = \ln\frac{1-\epsilon_t}{\epsilon_t}$$
$$\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$$

Problem 2. Decision Tree.(Total 5 points)

Consider a binary dataset with 400 examples, where half of them belongs to class A and another half belongs to class B.

Next consider two decision stumps (i.e. trees with depth 1) T1 and T2, each with two children. For T1, its left child has 150 examples in class A and 50 examples in class B; for T2, its left child has 0 example in class A and 100 examples in class B. (You can infer what are in the right child.)

- (**3 point**) For each leaf of T1 and T2, compute the corresponding classification error, entropy (base e) and Gini impurity. (Note: the value/prediction of each leaf is the majority class among all examples that belong to that leaf.)
- (**2 points**) Compare the quality of T1 and T2 (that is, the two different splits of the root) based on classification error, weighted entropy (base e), and weighted Gini impurity respectively.

3 pts

1. T1
   a) Left leaf (S=200, p=150/200=0.75)

$$Classification\ error = min\{0.75, 0.25\} = 0.25$$
$$entropy = 0.75 * \ln\left(\frac{1}{0.75}\right) + 0.25 * \ln\left(\frac{1}{0.25}\right) = 0.562335$$
$$Gini\ impurity = 2 * 0.75 * 0.25 = 0.375$$

   b) Right leaf (S=200, p=50/200=0.25)

$$Classification\ error = min\{0.25, 0.75\} = 0.25$$
$$entropy = 0.25 * \ln\left(\frac{1}{0.25}\right) + 0.75 * \ln\left(\frac{1}{0.75}\right) = 0.562335$$
$$Gini\ impurity = 2 * 0.25 * 0.75 = 0.375$$

2. T2
   a) Left leaf (S=100, p=0/100=0)

$$Classification\ error = min\{0, 1\} = 0$$
$$entropy = 0 * \ln\left(\frac{1}{0}\right) + 1 * \ln\left(\frac{1}{1}\right) = 0$$
$$Gini\ impurity = 2 * 0 * 1 = 0$$

   b) Right leaf (S=300, p=200/300=0.67)

$$Classification\ error = min\{0.67, 0.33\} = 0.33$$
$$entropy = 0.67 * \ln\left(\frac{1}{0.67}\right) + 0.33 * \ln\left(\frac{1}{0.33}\right) = 0.634179$$
$$Gini\ impurity = 2 * 0.67 * 0.33 = 0.4422$$

2 pts

1. T1

$$Classification\ error = \frac{50+50}{400} = 0.25$$

$$weighted\ entropy = \frac{200}{400}entropy_{left} + \frac{200}{400}entropy_{right} = \frac{1}{2}(0.562335 + 0.562335) = 0.562335$$

$$weighted\ gini\ impurity = \frac{200}{400}gini_{left} + \frac{200}{400}gini_{right} = \frac{1}{2}(0.375 + 0.375) = 0.375$$

2. T2

$$Classification\ error = \frac{100}{300} = 0.33$$

$$weighted\ entropy = \frac{100}{400}entropy_{left} + \frac{300}{400}entropy_{right} = \frac{1}{4} * 0 + \frac{3}{4} * 0.634179 = 0.4756$$

$$weighted\ gini\ impurity = \frac{100}{400}gini_{left} + \frac{300}{400}gini_{right} = \frac{1}{4} * 0 + \frac{3}{4} * 0.4422 = 0.332$$

Apart from classification error, other impurity measures are higher in the case of stump1 T1. T2 splits data better than T1.

Problem 3. Game Theory for Boosting

(**7 points**) Boosting explored from a game-theoretic perspective.

Some Background: A two-player zero-sum game is specified by matrix M and the two players are denoted the row player and the column player. The game is played by the row player choosing a row i and the column player choosing a column j, and the *loss* for the row player is $M(i, j)$ which is also the *reward* for the column player. Instead of choosing individual rows/columns, we will allow both players to choose distributions over rows/columns, so if row player choose P and column player choose Q, the loss/reward is

$$\sum_i \sum_j P(i)M(i,j)Q(j) \triangleq M(P,Q)$$

If we are acting as row player, we would like to choose a distribution P that achieves low loss, no matter what column player does. In other words, we would like to choose P that minimizes $\max_Q M(P, Q)$. On the other hand, if we were column player, we would like to choose Q that maximizes $\min_P M(P, Q)$. Von Neumann's celebrated minimax theorem states that in fact both these values are same, or in some sense it does not matter which player goes first in the game:

$$\max_Q \min_P M(P,Q) = \min_P \max_Q M(P,Q) \triangleq V$$

where V is called the value of the game. In this problem, we will use boosting to compute the optimal strategy in a particular game.

Let $\mathcal{H}$ be finite, and fix a target concept $c : \mathcal{X} \to \{+1, -1\}$ (not necessarily in $\mathcal{H}$) and sample $S = \{X_i, c(X_i)\}_{i=1}^n$ of size n. (The concept c is just a formulation on how the labels are generated.)

We will form a matrix $M \in \{0,1\}^{n\times|\mathcal{H}|}$ where $M(i,h) = \mathbb{1}\{h(X_i) = c(X_i)\}$. Here, the row player specifies distributions over samples, just as in boosting, and the column player chooses distributions over hypotheses.

a) (**4 points**) Assume that the empirical γ-weak learning assumption holds so that for every distribution P over examples, there exists a hypothesis $h \in \mathcal{H}$ such that $\mathbb{P}_{x \sim P}[h(x) \neq c(x)] \leq 1/2 - \gamma$. What does this mean about the value of the game? (Hint: First, think about what $M(P, Q)$ means. Then, find out what the relationship between V and γ is.)

b) (**3 points**) Let Q^* be distribution achieving value of this game, i.e. $\min_P M(P, Q^*) = V$. Since Q^* is distribution over hypotheses, what is the empirical error over Q^*? (Hint: consider $Q^*(h)$ as the weight α of weak learner h and then show what is the final predictor and what is the empirical error.)

c) (**Hurray!!! Extra credits: 10 points**) Boosting can be viewed as an iterative algorithm to compute Q^* using a weak learner. At every round, we choose P_t, a distribution over samples, and then compute

$$Q_t = \max_Q M(P_t, Q)$$

which is actually a single hypothesis h_t due to linearity. Then we update P_t to be

$$P_{t+1}(x) = \frac{P_t(x)}{Z_t} \times (\exp(-\eta \mathbb{1}\{h_t(x) = c(x)\}))$$

After T rounds, we output $\bar{Q} = \frac{1}{T}\sum_{t=1}^{T} Q_t$ which is a distribution over hypothesis and the final predictor is $H(x) = \text{sign}(\bar{Q}(x))$.

Prove that once $T = \Omega(\log(n)/\gamma^2)$ rounds and for appropriate choice of η, this variant of boosting guarantees (**Hint:** you may find it helpful to look at the Taylor expansion of $\exp(-x)$.)

$$\forall x \in X, \frac{1}{T}\sum_{i=1}^{T} M(x, h_t) > 1/2,$$

which implies that $H(x)$ has zero training error.

d) (**Let's have more!!! Extra credits: 5 points**) Informally, what does this mean about $\bar{Q}$, what is it converging to as $T \to \infty$?

Your answer. a Matrix M has rows and columns indexed by data samples and hypothesis H. M(P, Q) defines the expected accuracy when the data distribution is P and the hypothesis distribution is Q. For a given value of i - x hypothesis - h, $M(i, h) = 1\{h(X_i) = c(X_i)\}$
From the weak learners assumptions we get

$$P_{x \sim P}[h(x) \neq c(x)] \leq 1/2 - \gamma \tag{1}$$

Now, from the definition of M, and minmax theorem we get the following

$$\min_P \max_Q M(P, Q) = V$$
$$\min_P \max_Q M(P, Q) = \min_P \max_{h \epsilon H} M(P, h)$$
$$From\ the\ definition\ of\ M\ \ M(P, h) = P_{x \sim P}[h(x) = c(x)]$$
$$\min_P \max_Q M(P, Q) = \min_P \max_{h \epsilon H} P_{x \sim P}[h(x) = c(x)] = V$$

Now for a optimal value of P^*

$$P_{x \sim P^*}[h(x) = c(x)] = V \tag{2}$$
$$P_{x \sim P^*}[h(x) \neq c(x)] = 1 - V \tag{3}$$

From equations 1 and 3 we get

$$1 - V \leq 1/2 - \gamma$$
$$V \geq 1/2 + \gamma$$

Your answer. b Final predictor is given by,

$$y = sign(\sum_{h \epsilon H} Q^*(h)h(x))$$

Empirical error over Q^* is $P_{h \sim Q^*, x \sim P}[h(x) \neq c(x)]$
From problem a we have seen that

$$Empirical\ error = P_{h \sim Q^*, x \sim P}[h(x) \neq c(x)]$$
$$P_{h \sim Q^*, x \sim P}[h(x) = c(x)] = 1 - Empirical\ error$$
$$Now, P_{h \sim Q^*, x \sim P}[h(x) = c(x)] \geq 1/2 + \gamma$$
$$1 - Empirical\ error \geq 1/2 + \gamma$$
$$1 - \gamma \geq Empirical\ error$$
$$Empirical\ error \leq 1 - \gamma$$

Your answer. c From the definition of M, assumption of γ-learnability and above problem we get,

$$M(P_t, h_t) = Pr_{x \sim P_t}[h_t(x) = c(x)] \geq \frac{1}{2} + \gamma \tag{4}$$

From theorem 6.1 from texbook 'Boosting', For any matrix M with m rows and entries in [0, 1], and for any sequence of mixed strategies $Q_1, ..., Q_T$ played by the environment, the sequence of mixed strategies $P_1, ..., P_T$ produced by algorithm MW with parameter η satisfies,

$$\sum_{t=1}^{T} M(P_t, Q_t) \leq \min_P [a_\eta \sum_{t=1}^{T} M(P, Q_t) + c_\eta RE(P \| P_1)]$$
$$where$$
$$a_\eta = \frac{\eta}{1 - \exp^{-\eta}}$$
$$c_\eta = \frac{1}{1 - \exp^{-\eta}}$$

Here $RE(P \| P^{'})$ is distance between two probability distributions P and $P^{'}$ over $\{1, \ldots, m\}$ called relative entropy, also known as Kullback-Leibler divergence.

From Corollary 6.3, If MW is used with P1 set to the uniform distribution, then its total loss is bounded by,

$$\sum_{t=1}^{T} M(P_t, Q_t) \leq a_\eta \min_P \sum_{t=1}^{T} M(P, Q_t) + c_\eta \ln m$$

From theorem 6.1, corollary 6.3 and Taylor series expansion of $\exp^{-x}$, we can prove corollary 6.4, which is stated as follows,
Average per-trial loss suffered by the learner is

$$\frac{1}{T}\sum_{t=1}^{T} M(P_t, Q_t) \leq \min_P \frac{1}{T}\sum_{t=1}^{T} M(P, Q_t) + \Delta_T \tag{5}$$

$$\eta = \ln(1 + \sqrt{\frac{2\ln m}{T}})$$
$$\Delta_T = \sqrt{(\frac{2\ln m}{T})} + \frac{\ln m}{T} = O(\sqrt{\frac{\ln m}{T}})$$

From equation 4,5 and an appropriate value of η,

$$\frac{1}{2} + \gamma \leq \frac{1}{T}\sum_{t=1}^{T} M(P_t, h_t) \leq \min_{x \epsilon X} \frac{1}{T}\sum_{t=1}^{T} M(x, h_t) + \Delta_T$$
$$And\ for\ all\ x$$
$$\frac{1}{T}\sum_{t=1}^{T} M(x, h_t) \geq \frac{1}{2} + \gamma - \Delta_T > \frac{1}{2}$$

Now this implies,

$$\gamma - \Delta_T > 0$$
$$\gamma > \Delta_T$$
$$From\ corollary\ 6.4$$
$$\gamma > \sqrt{(\frac{2\ln m}{T})} + \frac{\ln m}{T}$$
$$\gamma^2 > \frac{2\ln m}{T}$$
$$T > \frac{\ln n}{\gamma^2}; m = n$$
$$T = \Omega(\frac{\ln n}{\gamma^2})$$

Your answer. d Consider following equation derived in problem c

$$\frac{1}{T}\sum_{t=1}^{T} M(x, h_t) \geq \frac{1}{2} + \gamma - \Delta_T > \frac{1}{2}$$

The summation term on LHS is the number of hypothesis h_t which agree with c on instance x. As T gets large, Δ_T approaches 0, a minimum margin of at least 2γ is obtained asymptotically.

Also

From corollary 6.3, theorem 6.1 and assumption of γ-learnability, we get (following similar logic as problem c)

$$\frac{1}{2} + \gamma \leq \frac{1}{T}\sum_{t=1}^{T} M(P_t, h_t) \leq a_\eta \min_{x\epsilon X} \frac{1}{T}\sum_{t=1}^{T} M(x, h_t) + \frac{c_\eta \ln m}{T}$$

$$\frac{1}{2} + \gamma \leq a_\eta \min_{x\epsilon X} \frac{1}{T}\sum_{t=1}^{T} M(x, h_t) + \frac{c_\eta \ln m}{T}$$

$$Rearranging,$$

$$\min_{x\epsilon X} \frac{1}{T}\sum_{t=1}^{T} M(x, h_t) \geq \frac{1}{a_\eta}[(\frac{1}{2} + \gamma) - \frac{c_\eta \ln m}{T}]$$

$$Substituting\ a_\eta\ and\ c_\eta\ from\ problem\ c\ and\ \eta = 2\alpha\ and\ simplifying$$

$$\min_{x\epsilon X} \frac{1}{T}\sum_{t=1}^{T} M(x, h_t) \geq (\frac{1}{2} + \gamma) - \alpha(\frac{1}{2} + \gamma) - \frac{\ln m}{2\alpha T}$$

When T is very large, $\frac{\ln m}{2\alpha T}$ becomes negligible. Therefore asymptotically the margins come within $\alpha(1 + 2\gamma) \leq 2\alpha of 2\gamma$, the best possible margin for the given -weak learning assumption

Programming Assignment

Instruction. For each problem, you are required to report descriptions and results in the PDF and submit code as python file (.py) (as per the question).

- **Python** version: Python 3.
- Please follow PEP 8 style of writing your Python code for better readability in case you are wondering how to name functions & variables, put comments and indent your code
- **Packages allowed**: numpy, pandas, matplotlib
- **Submission**: Submit report, description and explanation of results in the main PDF and code in .py files.
- Please **PROPERLY COMMENT** your code in order to have utmost readability otherwise **1 MARK** would be deducted

Programming Common

This programming assignment focuses on boosting and bagging approaches.

Problem 4. "Blind" boosting without training data.

Problem 4.1. Blind boosting for regression (9 Points)

Imagine we do not have the training dataset visible but a scoring function is made available to you. This score function gives us an idea about how well your model is doing on the test dataset. Higher score indicates worse performance. The only knowledge you have is that the predict target, $y \in \mathbb{R}$. Can we achieve better performance than a completely random guessing using **just** the knowledge of how well your model does on test dataset. If **yes**, then explain how? You have to implement this model. Show how the score changes as we increase the number of weak learners in the boosting model (show a plot).
(Hint: Please try to use the idea of gradient boosting and also try to randomly guess the gradient.)

Note: For getting the score, use the ***score*** function in **test_score.py** script by importing it in your **hw4_boosting_regression.py** file you would submit as solution for this question. The function ***score*** accepts only one parameter which is of type 'numpy.ndarray' and has shape of (N,1) where N is the number of samples predicted passed as argument to this function. Here, we have 21283 samples in testing dataset therefore $N = 21283$.

Keep the **test_score.py** and **true_values_regression.csv** which contains the true values of test dataset in the same location where **hw4_boosting_regression.py** file would be. This ***score*** basically calculates how much error is made in predictions by comparing with true value in **true_values.csv** testing dataset.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw4_boosting_regression.py).

Your answer We will use gradient boosting for this problem. Following algorithm will work, as will try minimizing the gradient computed in the last iteration.

1. Initialize a random model h_t
2. Compute error for the model/prediction
3. Compute alpha for this h_t
4. Randomly guess gradient values, by limiting the sum over gradients to the error obtained in this iteration
5. Add this gradient to the models array
6. Repeat this algorithm

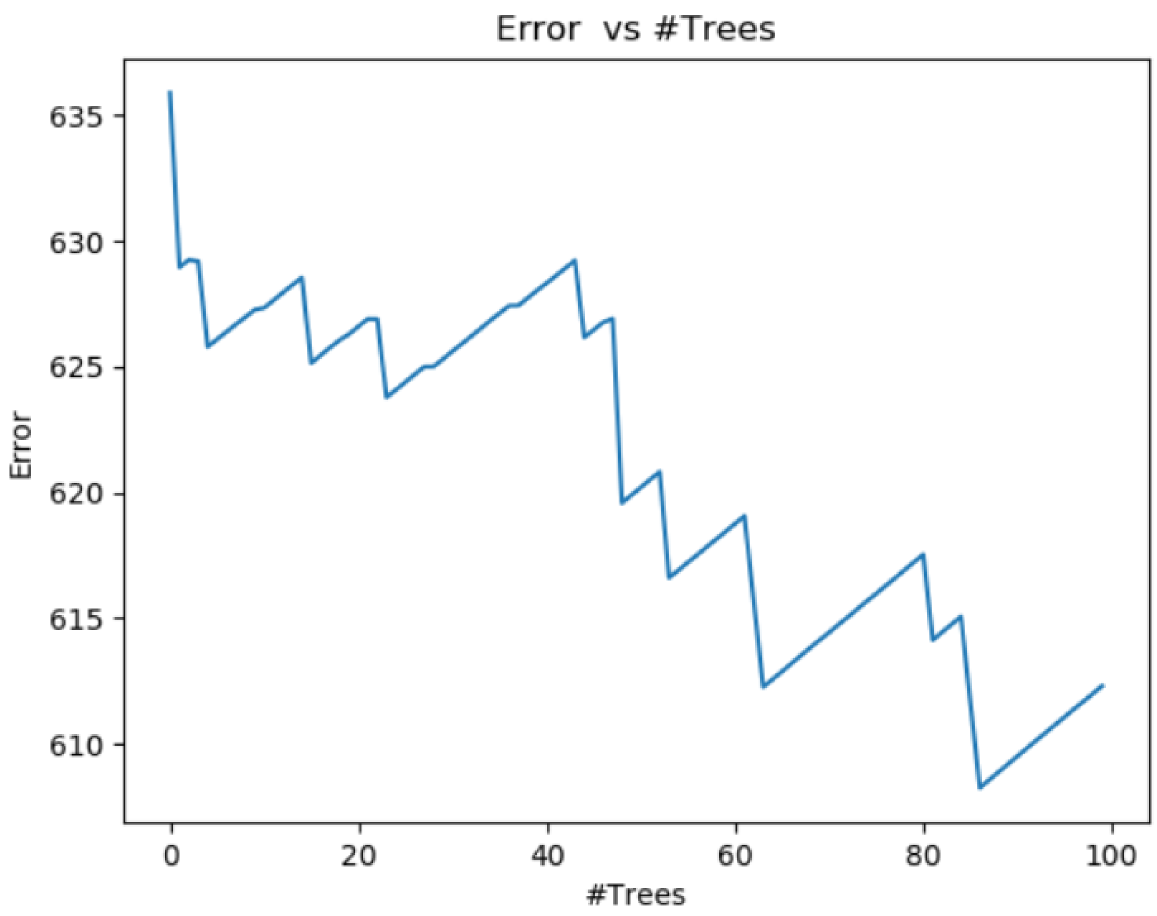


Figure 1: Boosting Regression

While computing prediction, we take weighted sum of all models. As iterations go high, we try to minimize the gradient with respect to prediction values till this round. Hence the error will go down as iterations increase. The drop is not smooth since we are randomly generating the gradient.

Problem 4.2. Blind boosting for classification (9 Points)

Imagine we do not have the training dataset visible but a error rate function is made available to you. This function gives us an idea about how accurate the model is on the test set. The only knowledge you have is that the predict target, $y \in \{0, 1\}$. Can we achieve better performance than a completely random guessing using **just** the knowledge of how well your model does on test dataset. If **yes**, then explain how? You have to implement this model. Show how the score changes as we increase the number of weak learners in the boosting model (show a plot).

Note: For getting the error rate, use the ***error_rate*** function in **test_error_rate.py** script by importing it in your **hw4_boosting_classification.py** file you would submit as solution for this question. The function ***error_rate*** accepts only one parameter which is of type 'numpy.ndarray' and has shape of (N,1) where N is the number of samples predicted passed as argument to this function. Here, we have 21283 samples in testing dataset therefore $N = 21283$.

Keep the **test_error_rate.py** and **true_values_classification.csv** which contains the true values of test dataset in the same location where **hw4_boosting_classification.py** file would be. This ***score*** basically calculates how much error is made in predictions by comparing with true value in **true_values.csv** testing dataset.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw4_boosting_classification.py).

Your answer We will use gradient boosting for this problem. Following algorithm will work, as will try minimizing the gradient computed in the last iteration.

1. Initialize a random model h_t
2. Compute error for the model/prediction
3. Compute alpha for this h_t
4. Randomly guess gradient values, by limiting the sum over gradients to the error obtained in this iteration
5. Add this gradient to the models array
6. Repeat this algorithm

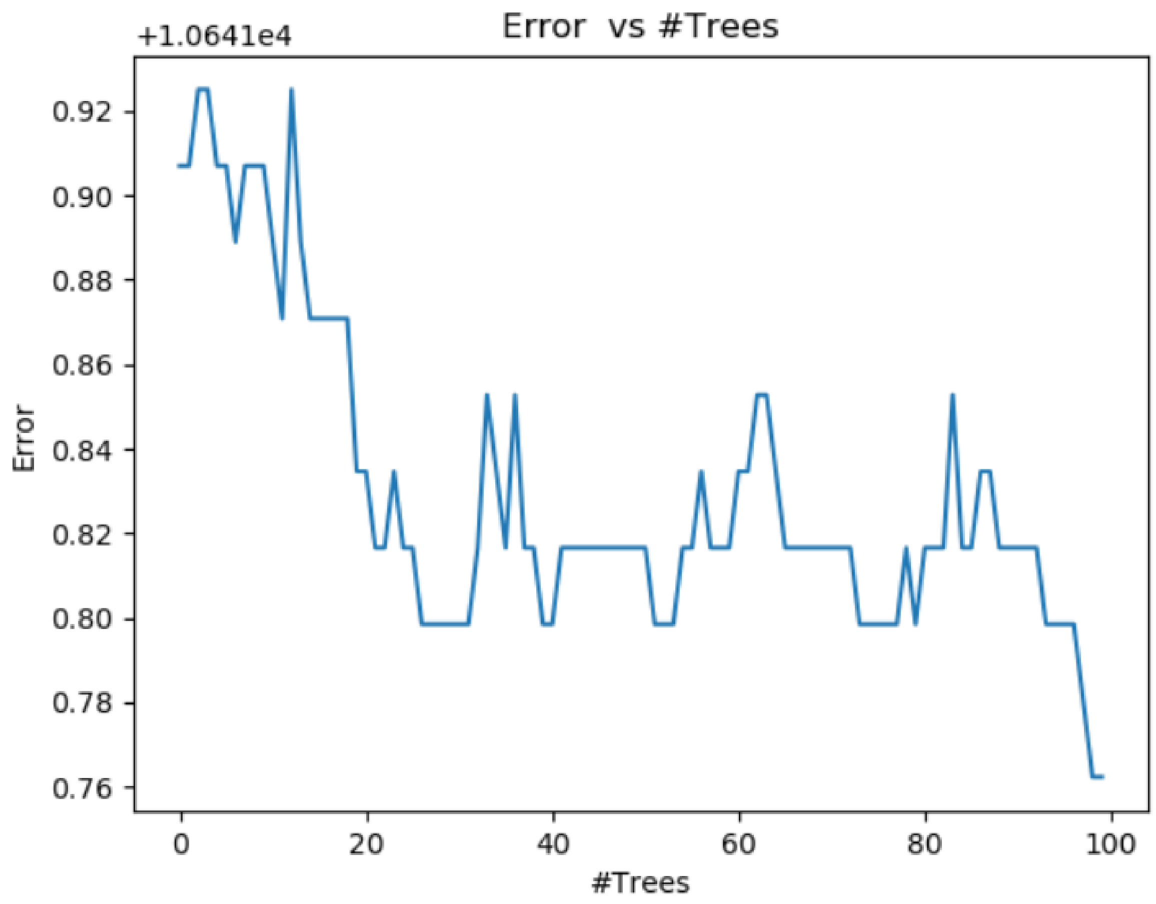


Figure 2: Boosting Classification

While computing prediction, we take weighted sum of all models. As iterations go high, we try to minimize the gradient with respect to prediction values till this round. Hence the error will go down as iterations increase. The drop is not smooth since we are randomly generating the gradient.

Problem 5. Adaboost.

(**Total 12 Points**) For this problem, you will use a cancer dataset as in the file cancer_train.csv and cancer_test.csv. You can find this file in the canvas file folder named hw4. The goal is to predict whether there is a cancer: 0 or 1, included in column y.

a) (**7 Points**) Implement an Adaboost algorithm with 100 weak learners. You are allowed to use decision tree classifier with maximum depth = 1 from scikit standard library (sklearn link). **No other function from scikit learn is allowed**. Use gini as quality measure for the decision tree splits and use the default parameters.

b) (**2 Points**) Plot the misclassification error on both train and test set as the number of weak learners increase.

c) (**3 Points**) plot the margin distribution when the number of weak learners = [25, 50, 75, 100]. In total you will show 4 plots.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw4_adaboost.py).

Your answer a

1. Training accuracy - 92.97%
2. Test accuracy - 87.72%

Your answer. b Please refer figure 3

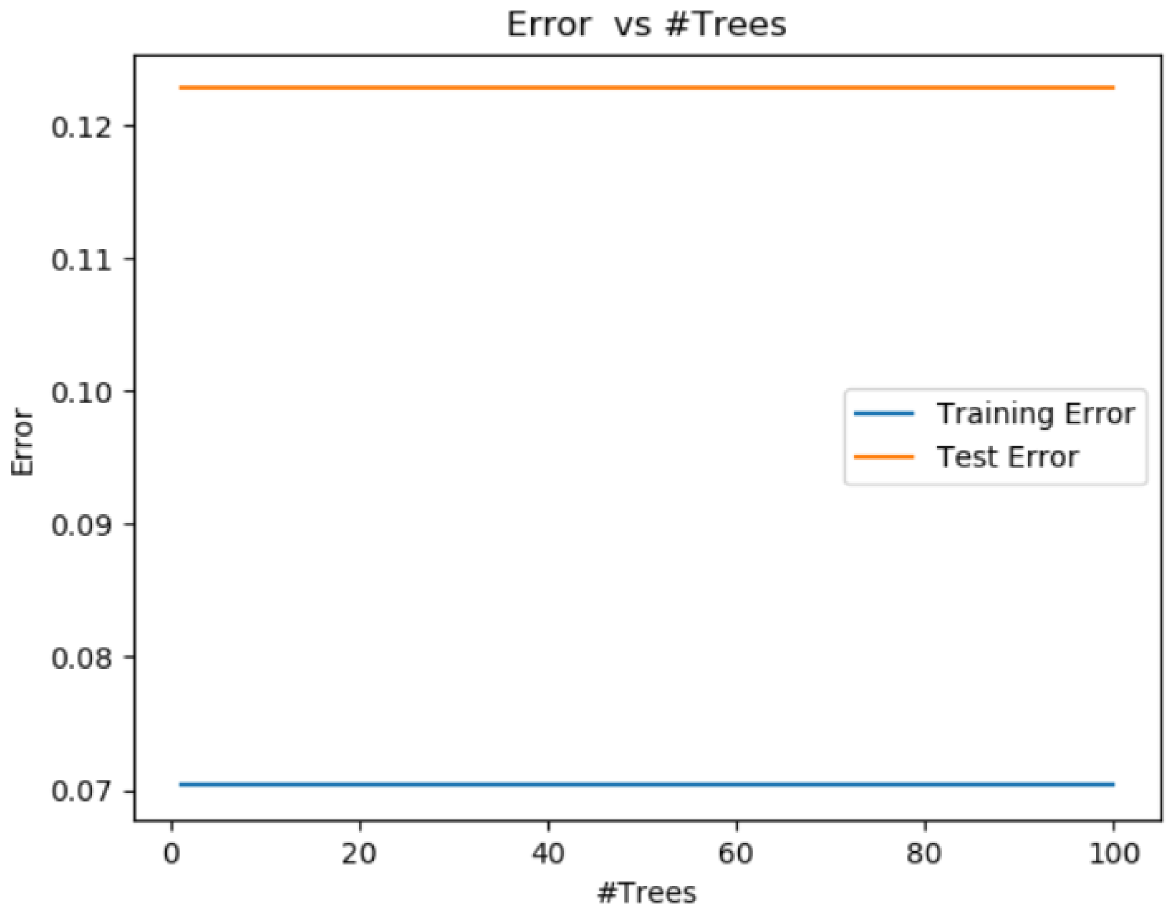
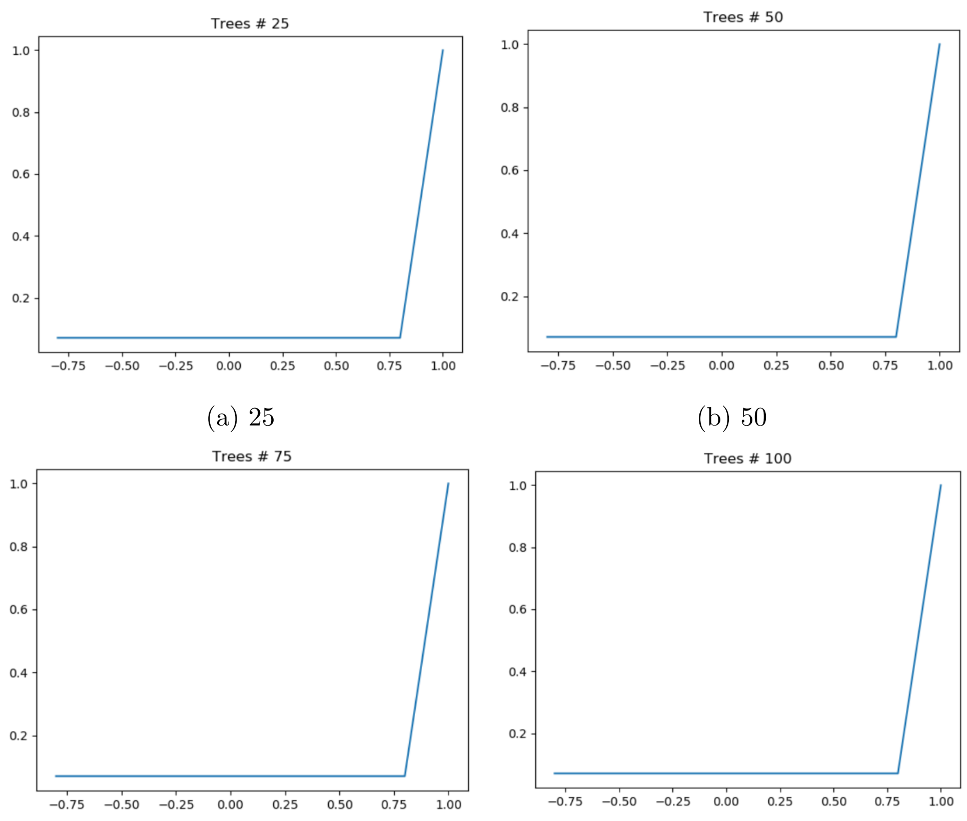


Figure 3: Adaboost Error vs Trees

Your answer. c Please refer figure 4



(a) 25 (b) 50

(c) 75 (d) 100

Figure 4: Cumulative margin distribution

Problem 6. Random forest.

(**Total 13 Points**) For this problem, you will use a health dataset as in the file health_train.csv and health_test.csv. You can find this file in the canvas file folder named hw4. The goal is to predict the overall health of an individual: either 0 for poor health or 1 for good health, included in column y.

a) (**7 Points**) Implement a random forest of 100 decision trees. You are allowed to use decision tree classifier from scikit standard library (sklearn link). **No other function from scikit learn is allowed**. Use gini as quality measure for the decision tree splits and use the default parameters.

b) (**3 Points**) Vary the size of random feature sets as [50, 100, 150, 200, 250] and fit the model. After the model is fit, plot the accuracy on train and test set vs size of the random feature set.

c) (**3 Points**) Vary the number of estimators i.e. number of decision trees as [10, 20, 40, 80, 100] for feature size 250 and plot the accuracy on train and test set for number of estimators.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw4_random_forest.py).

Your answer a

1. Training accuracy - 91.66%
2. Test accuracy - 79.5%

Your answer b Please refer figure 5

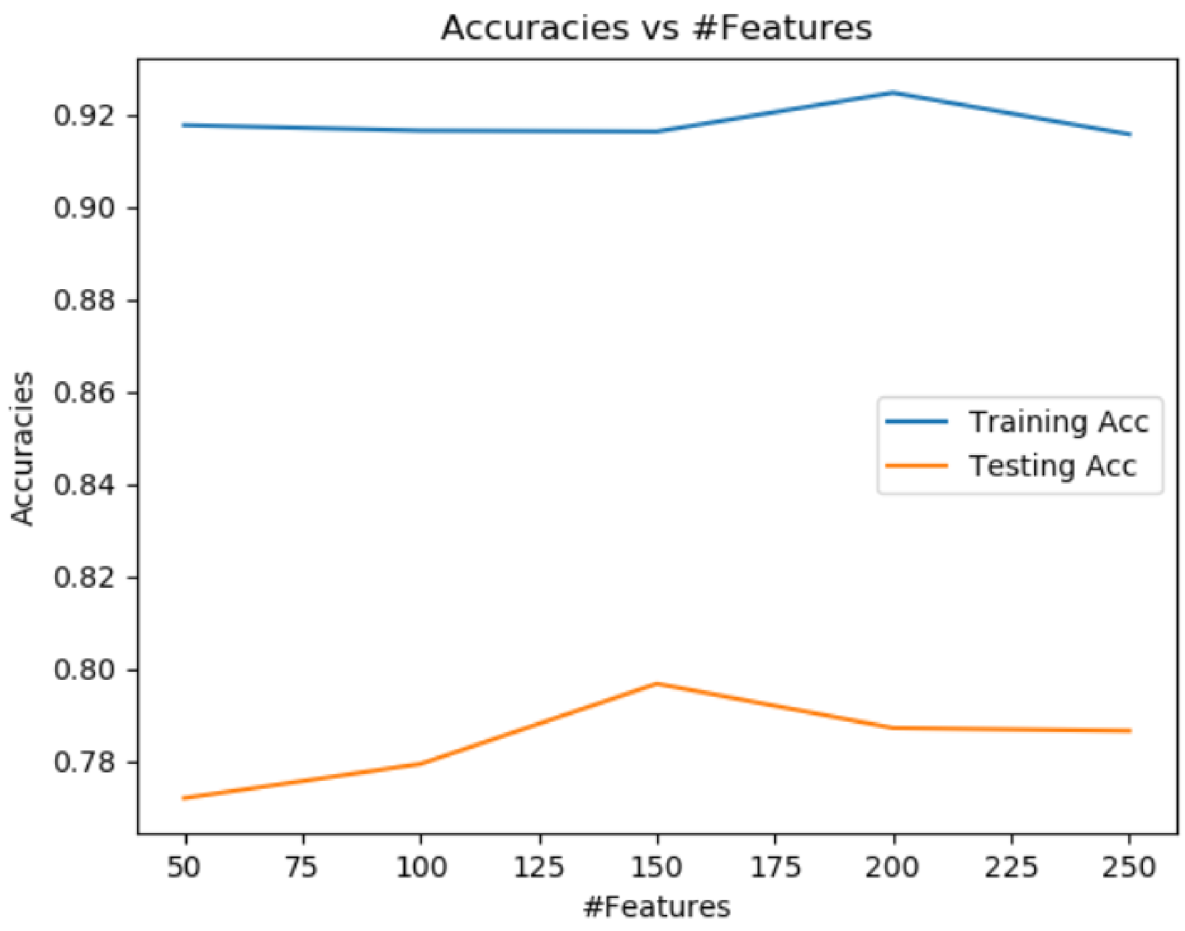


Figure 5: Random Forest - Accuracy vs Feature

Your answer c Please refer figure 6

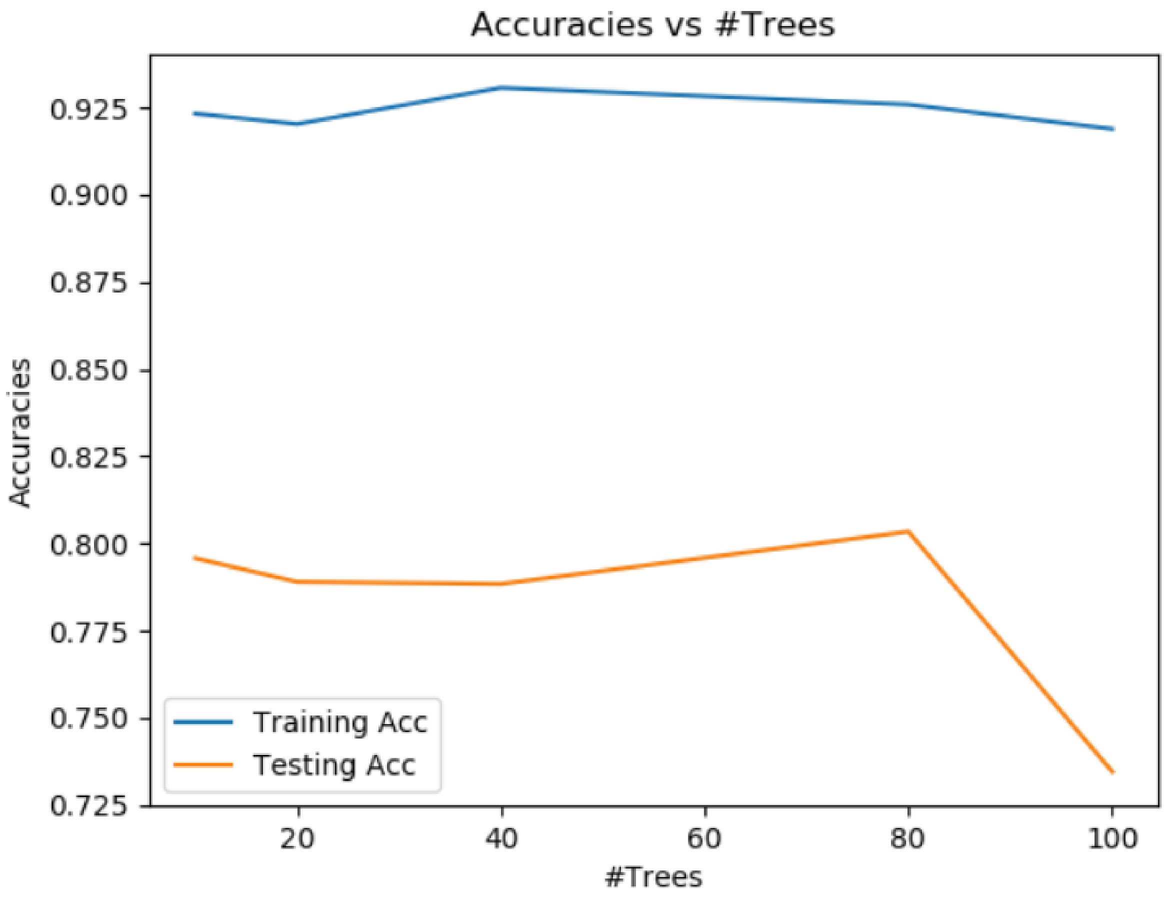


Figure 6: Random Forest - Accuracy vs Trees