CS 475 Machine Learning: Homework 2
Supervised Classifiers 2: Non-linear Classifiers
Due: Thursday October 10, 2019, 11:59pm
100 Points Total Version 1.1

Student name: Spencer Loggia

Make sure to read from start to finish before beginning the assignment.

1 Programming (50 points)

In this assignment, you will implement two regression models: regression trees and gradient boosted regression trees. You will be using the same Python framework as before. Remember: **do not change the names of any of the files or command-line arguments that have already been provided.**

1.1 Regression using trees

You will implement two tree-based methods for regression: regression trees and gradient-boosted regression trees (GBRTs). Regression trees are a base object in GBRTs, so you should implement regression trees first, and then use that implementation to build the GBRT. You will be using these methods to predict the price of a house given 37 features describing the house. The data was extracted from here (`https://www.kaggle.com/c/house-prices-advanced-regression-techniques`) with some minor preprocessing.

1.1.1 Regression Tree (20 points)

Regression tree features: Regression trees, unlike decision trees, output a real value. So we are given a dataset of n examples, $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$. Since we are doing *regression*, we will not use log-likelihood or information gain to decide which features to split on. Instead, we will minimize the **residual sum of squared error**, given the regression tree's prediction function $f : \mathbb{R}^D \mapsto \mathbb{R}$,

$$\mathcal{E}(f) \overset{\text{def}}{=} \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{1}$$

To support real-valued features, we will recursively split feature dimensions d based on feature thresholds θ. A schematic of such a tree is given in figure 1.

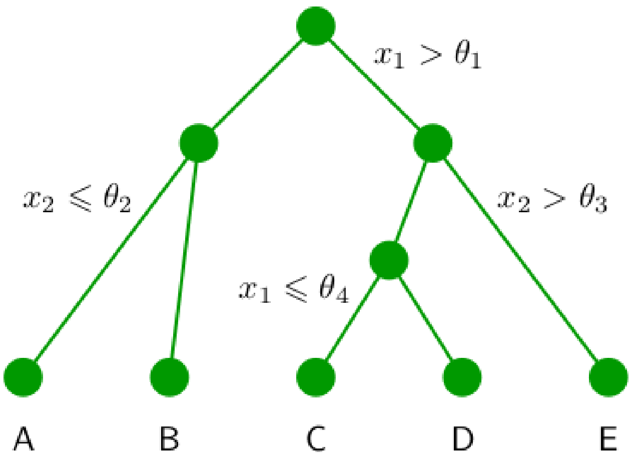


Figure 1: Illustration of regression tree (image credit to Chris Bishop's book). In this figure x_1 and x_2 are the first and second features of a specific example x.

Growing the tree: Much like the decision tree setting, each node in the decision tree partitions the data of its parent node. At each node, we will consider partitioning the data at that node along each feature dimension. Partitioning a dimension will be done by a numeric threshold. Given the data at the node, $(X, \boldsymbol{y})$, a feature dimension d, and a threshold θ, we get two subsets of data indices.[1]

$$L_{d,\theta}(X) \overset{\text{def}}{=} \{i \mid X_{i,d} < \theta\} \tag{2}$$

$$R_{d,\theta}(X) \overset{\text{def}}{=} \{i \mid X_{i,d} \geq \theta\} \tag{3}$$

We will score a split according to the residual sum of squared errors from the split-conditional mean[2]

$$s(L, R, \boldsymbol{y}) \overset{\text{def}}{=} \sum_{i \in L}(y_i - \mu_L)^2 + \sum_{i \in R}(y_i - \mu_R)^2 \tag{4}$$

where μ_L and μ_R are the empirical mean of y given the respective set of points, $\mu_L \overset{\text{def}}{=} \frac{1}{|L|}\sum_{i \in L} y_i$, and similarly for μ_R. We will pick the split that minimizes

$$(d^*, \theta^*) = \underset{d \in \{1,\ldots,D\},\ \theta \in \mathbb{R}}{\operatorname{argmin}} \ s(L_{d,\theta}(X), R_{d,\theta}(X), \boldsymbol{y}) \tag{5}$$

In order to minimize this function with respect to real-valued thresholds $\theta \in \mathbb{R}$, it turns out that we can limit the search to the following *finite* set

$$\Theta_d(X) \overset{\text{def}}{=} \{X_{i,d} \mid i \in \{1, \ldots, n\}\} \tag{6}$$

This gives is the following discrete optimization problem,

$$(d^*, \theta^*) = \underset{d \in \{1,\ldots,D\},\ \theta \in \Theta_d(X)}{\operatorname{argmin}} \ s(L_{d,\theta}(X), R_{d,\theta}(X), \boldsymbol{y}) \tag{7}$$

Given the choice of the splitting criteria, (d^*, θ^*), we create two children nodes with the appropriate L and R sets. And, recursively follow the same steps.

[1] L of left of the threshold, R for right of the threshold.

[2] This is a reasonable heuristic because it is an upper bound: We know that we can achieve *at least* as small as this residual value if we were to create leaf nodes on the next step (described later).

Base cases: Please use the following base cases for building decision trees:

1. There are ≤ 1 examples assigned to the node.
2. All features have zero variance in the examples assigned to the node
3. The path to the node has reached the maximum depth.[3]

Leaf nodes: When we hit a base case, we will create a leaf node. Each leaf node predicts a constant value. The choice for the constant value that optimizes the residual sum of squares is the mean. Therefore, leaves will predict the mean of its y data.

Remarks:

- Feature dimension can be split multiple times.
- Do not consider a split that results in empty sub-trees as this will result in an undefined residual sum of squared error.
- A feature should not be considered for a split if the feature has no variance across the samples assigned to the node in order to avoid empty sub-trees (i.e., nodes with no samples assigned).

Speedup: If implemented naively, the split-selection optimization algorithm runs in time $\mathcal{O}(N^2 D)$ because it recomputes the residual sum of squares (in time $\mathcal{O}(N)$) for all possible features and thresholds (of which there are $\mathcal{O}(DN)$ many). There are faster implementations that can run in $\mathcal{O}(ND + N \log N)$. However, we will leave such exploration up to students. We will give full credit for the slower implementation.

[3] The depth of a node in a tree is the total length of the path between that node and the root of the tree. A tree of only a single root node has a depth of 0. In a decision tree, each node contains either a feature test or a value. Therefore, a tree of depth 0 can only contain a single node, which must just be a prediction value. A tree with a single feature test, which leads to a predicted value on both subsequent nodes, has depth 1 since each leaf is one step away from the root.

1.1.2 Gradient-Boosted Regression Tree (20 points)

Gradient-Boosted Regression Trees (GBRT) use regression trees iteratively. At a high level, in each round of boosting, a regression tree is trained to predict the data points that were most difficult to accurately predict by trees in previous iterations of boosting (For more details see section 14.3-14.4 in Bishop: `https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf`.) for reference on GBRTs.

GBRT Pseudocode: The following pseudocode describes the GBRT algorithm. We assume that there are N training examples, where a given example i has label y_i and feature vector x_i. M is the model hyper-parameter (n-estimators) representing the number boosting (training) iterations. $0 < \lambda \leq 1$ is the hyper-parameter (regularization-parameter) used to regularize the regression function.

```
1. Initialize the GBRT model ($F_0(x_i)$) to an the mean of your labels:
   $F_0(x_i) = \frac{1}{N}\sum_{i=1}^{N} y_i$
2. For iteration $m = 1$ to $M$:
    (a) Calculate the residuals given the current GBRT model. Specifically, use $F_{m-1}$ to make predictions on each of the training examples ($F_{m-1}(x_i)$). Then, using the predictions, calculate the residuals according to:
            $g_i^{(m)} = y_i - F_{m-1}(x_i) \quad \text{for } i = 1 \ldots N$    (8)
    (b) Fit a regression tree $h_m(x)$ to the training data $\{(x_i, g_i^{(m)})\}$ (Use your implementation from section 1.1.2. Make sure to use the max depth parameter, which is give as a command-line option.)
    (c) Update the model: $F_m(x) = F_{m-1}(x) + \lambda h_m(x)$
3. Return $F_M(X)$
```

1.1.3 Implementation Details

The two algorithms you implement in this assignment will be selected using arguments passed to the command-line argument `--algorithm`. The arguments are `regression-tree` and `gradient-boosted-regression-tree`.

Your code must correctly implement the behavior described by the following command-line options, which are passed in as arguments to the constructor of your implementation in `models.py`.

- `--max-depth` The maximum depth a tree can be grown to. The default is 3. This is used for both `regression-tree` and `gradient-boosted-regression-tree`.
- `--n-estimators` The number of number of iterations (or equivalently the number of regression trees) used for `gradient-boosted-regression-tree`. By default, this value is 100.
- `--regularization-parameter` The regularization parameter used by `gradient-boosted-regression-tree`. By default, this value is 0.1.

1.1.4 Examples

As an example, the following trains a regression tree on the training data given in a file `train.txt` that we parse and load for you:

```
python3 main.py --mode train --algorithm regression-tree --model-file regression.tree.model \
                --train-data train.txt
```

To run the trained model on development data:

```
python3 main.py --mode test --model-file regression.tree.model --test-data dev.txt \
                --predictions-file dev.predictions
```

We provide the script `compute_mean_square_error.py` to evaluate the root mean squared error of your model's predictions:

```
python3 compute_root_mean_square_error.py dev.txt dev.predictions
```

1.1.5 Jupyter Notebook (10 points)

We have provided a Jupyter Notebook (`Regression-Experiments.ipynb`) containing several experiments that explore differences between regression trees and GBRTs, as well as the effects of hyper-parameter choice on the two models. After you have finished your implementations of regression trees and GBRTs, run the entire notebook and complete the prompt at the bottom of the notebook. Then create a PDF file containing the notebook after it has been run so we can see all of your output.

2 Analytical (50 points)

Instructions We have provided this LaTeX document for turning this homework. We give you one or more boxes to answer each question. The question to answer for each box will be noted in the title of the box. **Do not type anything outside the boxes. Leave the rest of the document unchanged. There is a box for your name on the first page of this document.**

For written answers, replace the \TextRequired (**Place Answer Here**) command with your answer. For the following example *Question 0.1* , you would place your answer where \TextRequired (**Place Answer Here**) is located,

Place Answer Here

Do not change the height or title of the box. If your text goes beyond the box boundary, it will be cut off. We have given sufficient space for each answer, so please condense your answer if it overflows. The height of the box is an upper bound on the amount of text required to answer the question - many answers can be answered in a fraction of the space. Do not add text outside of the boxes. We will not read it.

For answers that require multiple equations, such as a derivation, place all equations within the specified box. You may include text short explanations if you wish (as shown in *Question 0.4*). You can put the equations in any format you like (e.g. within $ or $$, the \equation environment, the \align environment) as long as they stay within the box.

$$\begin{aligned}
&x + 2 && \text{x is a real number}\\
& && \text{the following equation uses the variable } y\\
&y + 3
\end{aligned}$$

- **Do not change any formatting in this document, or we may be unable to grade your work. This includes, but is not limited to, the height of textboxes, font sizes, and the spacing of text and tables. Additionally, do not add text outside of the answer boxes. Entering your answers are the only changes allowed.**
- **We strongly recommend you review your answers in the generated PDF to ensure they appear correct. We will grade what appears in the answer boxes in the submitted PDF, NOT the original latex file.**

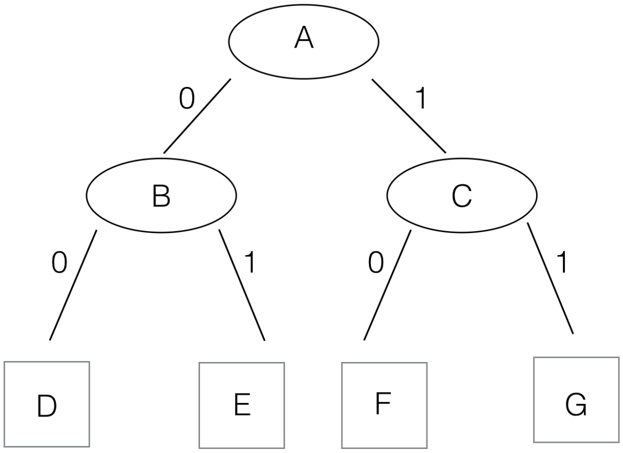
1) Decision Trees (9 points)

Consider the classification task where you want to predict y given x_1, x_2, x_3, x_4, x_5.

x_1	x_2	x_3	x_4	x_5	y
1	0	0	0	1	1
1	0	1	0	1	1
0	1	0	1	1	1
0	0	0	1	1	0
1	1	1	0	1	0
1	0	1	1	1	0
1	0	0	1	1	0
0	1	0	0	1	0

(1) Construct a decision tree based on the above training examples following the algorithm we specified in class using the information gain criterion and a maximum depth of 2. As an additional base case, stop expanding if all training examples have the same label. You may use each feature at most once along any path from the root to a leaf.

Using the decision tree schematic below, specify the correct feature number for internal nodes A, B, and C. For each nodes D, E, F, and G, specify the correct label. Put answers in the pre-formatted table by the "?" with the correct feature or label.



Node	Id or label
A=	4
B=	2
C=	2
D=	1
E=	0
F=	0
G=	1

(2) Apply the decision tree learned in part 1 of this question to classify the following new data points. Replace the "?" in the table below with the classifier's prediction.

x_1	x_2	x_3	x_4	x_5	y
0	1	1	1	1	1
1	1	0	1	1	1
1	1	0	0	1	0

(3) For the training dataset in part 1, can any of the methods listed in the table below obtain a *training accuracy* of 100% using only the features given? Answer below by replacing the "?" with "Y" or "N".

Method	*Y/N*
Decision tree of depth 1	N
Decision tree of unlimited depth	Y
Logistic regression	N
Perceptron	N
Linear Kernel SVM	N
Quadratic Kernel SVM	N
AdaBoost with decision stumps	N

2) Kernels (10 points)

(1) Give $\mathcal{O}(\cdot)$ bounds for the different cases below in terms of the following quantities:

number of training examples	n
number of support vectors	s
feature dimensionality	d
kernel evaluation time	k
size of an example	e

Suppose in each case below that we are predicting labels for m test examples

The running time for a kernel SVM	$\mathcal{O}(sekm)$ *dim(s)=e support vectors into kernel
The running time for a primal SVM	$\mathcal{O}(dm)$ *explicitly compare to stored hyperplane
The space complexity for a kernel SVM	$\mathcal{O}(se)$ *store SVs each of dim(example)
The space complexity for a primal SVM	$\mathcal{O}(d)$ *store d dimensional hyperplane

(2) Suppose I trained a kernel SVM using a Gaussian kernel: $K(x, x') \overset{\text{def}}{=} \exp\left(\frac{-||x-x'||^2}{2\sigma^2}\right)$, where σ is a positive scalar. If my training set accuracy is very low, should I increase or decrease σ? Why?

If you are getting very low accuracy on the training data you are likely under-fitting. The solution to this is to decrease the magnitude of the σ variable - which is the variance parameter. This will ”tighten” the Gaussian kernel and improve accuracy on the train data, though reducing it to much may lead to overfitting.

3) AdaBoost (10 points)

Assume a one-dimensional dataset with $x = \langle -1, 0, 1 \rangle$ and $y = \langle -1, +1, -1 \rangle$. Consider three weak classifiers:

$$h_1(x) = \begin{cases} 1 & \text{if } x > \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}, \qquad h_2(x) = \begin{cases} 1 & \text{if } x > -\frac{1}{2} \\ -1 & \text{otherwise} \end{cases}, \qquad h_3(x) = 1.$$

(1) Show your work for the first 2 iterations of AdaBoost. You must use the method from the lecture slides to get full credit.[4] In the table below, replace the "?" in the table below with the value of the quantity at iteration t. $h_t \in \{1, 2, 3\}$, weak learner's weight in the ensemble, α_t, error rate, ε_t. Additionally, give the distribution over example at iteration t, $D_t(1), D_t(2), D_t(3)$. Use log with base e in your calculations.

t	h_t	α_t	ε_t	$D_t(1)$	$D_t(2)$	$D_t(3)$
1	2	3.47	.33	.33	.33	.33
2	2	0	.50	.25	.25	.50
**assuming Di is input distribution to that row						

(2) After running AdaBoost for 10 iterations, what is the error rate of the ensemble?

the error rate of the ensemble after ten iterations will be .33

[4] The method from class is equivalent to the algorithm in Figure 1 of `https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf`

(3) In general, getting near-perfect training accuracy in machine learning leads to overfitting. However, AdaBoost can get perfect training accuracy within overfitting. Give a brief justification for why that is the case.

Empirically, it makes sense that since only one of many classifiers is updated at a time on a modified distribution that the ”voice” of a single classifier that has overfit will be drowned out by the others. From the slides, it can be proven that ADABoost generalization depends on the margin of examples not the number of training rounds.

(4) Give big-$\mathcal{O}$ bounds for the time, and space complexity of a boosting algorithm run for T training iterations. Suppose that each weak learner takes at most time H to execute and at most S space to store.

What is the time complexity of making predictions on M testing examples?

$\mathcal{O}(MTH)$

What is the space complexity of storing the model?

$\mathcal{O}(TS)$

(5) The AdaBoost algorithm makes calls to a weighted classification solver which approximately solves the weighted 0/1-loss problem.[5] In other words, the classifier training algorithm is a function from a weighted dataset $\{(w_i, x_i, y_i)\}_{i=1}^n$ to a classifier which approximately minimizes,

$$\operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^{n} w_i \mathbf{1}(h(x_i) \neq y_i) \tag{9}$$

Suppose we have a new implementation of a classifier and want to use it in AdaBoost. However, the classifier's training algorithm only accepts a dataset of x-y pairs, $\{(x_i, y_i)\}_{i=1}^n$ and thus it solves,

$$\operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(h(x_i) \neq y_i) \tag{10}$$

Give a give a principled method to *approximate* the weighted classification solution (9) given access to a training algorithm that only accepts x-y pairs (10). Briefly sketch your idea and why it is a reasonable approximation.

The basic idea behind weighting the examples fed to the classifier in ADA-Boost is to make the new classifier "care" more about getting examples that were classified wrong in the past right. Normally this is done by increasing the relative contribution to the loss function of previously incorrect examples, so that classifiers that get them wrong again get higher loss. If we cannot pass and updated weight vector into the classifier, it will choose to correctly classify examples that have already been classified correctly, and thus ADA-Boost is broken. However, we can prevent the classifier from choosing the "easy" examples by omitting them. We may choose a threshold $0 < \theta < 1$, if the weight associated with an example is below θ then we will not pass it to the classifier. A good choice of theta may be $1/m$, where m is the total number of examples in the data-set, because then we will omit examples that have been predicted better than random chance. This will allow ADA-Boost to function, though probably not as well as before since we have lost information about the exact magnitude of w_i

[5] The approximation may come from minimizing an upper bound on 0/1 loss such as hinge loss.

3 What to Submit

In each assignment you will submit three things.

1. **Submit your code (.py files) to cs475.org as a zip file**. **Your code must be uploaded as code.zip with your code in the root directory**. By 'in the root directory,' we mean that the zip should contain `*.py` at the root (`./*.py`) and not in any sort of substructure (for example `hw1/*.py`). One simple way to achieve this is to zip using the command line, where you include files directly (e.g., `*.py`) rather than specifying a folder (e.g., `hw1`):

   ```
   zip code.zip *.py
   ```

   A common mistake is to use a program that automatically places your code in a subfolder. It is your job to make sure you have zipped your code correctly.

   We will run your code using the exact command lines described earlier, so make sure it works ahead of time, and make sure that it doesn't crash when you run it on the test data. A common mistake is to change the command line flags. If you do this, your code will not run.

   Remember to submit all of the source code, including what we have provided to you. We will include `requirements.txt` but nothing else.

2. **Submit your writeup to gradescope.com**. **Your writeup must be compiled from latex and uploaded as a PDF**. The writeup should contain all of the answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and to use the provided latex template for your answers following the distributed template. You will submit this to the assignment called "Homework 2: Supervised Classifiers 2: Written".

3. **Submit your Jupyter notebook to gradescope.com**. Create a PDF version of your notebook (File → Download As → PDF via LaTeX (.pdf)). Be sure you have run the entire notebook so we can see all of your output. You will submit this to the assignment called "Homework 2: Supervised Classifiers 2: Notebook". When you submit on Gradescope, you will indicate which output box corresponds to which questions.

You will need to create an account on `http://gradescope.com` and signup for this class. The course is `https://gradescope.com/courses/21552`. Use entry code `M6ZX2X`. See this video for instructions on how to upload a homework assignment: `https://www.youtube.com/watch?v=KMPoby5g_nE`.

4 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: `https://piazza.com/class/jkqbzabvyr15up`.