MACHINE LEARNING - I

UNIT # 6

1

TODAY'S AGENDA

- Discussion on Assignment # 1
- Recap of the previous lecture
  - Regression Tree, Bagging, Random Forest
- Boosting Models
  - Adaboost
  - Gradient Boosting

2

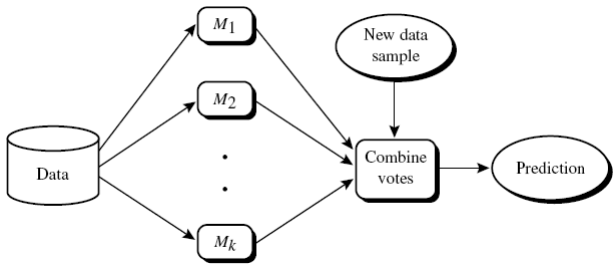
RECAP: REGRESSION TREE

- Regression Tree (and Classification Tree) is a flexible data-driven method.
- It is transparent and easy to interpret.
- Trees are based on separating records into subgroups by creating splits on predictors. These splits create logical rules.
- As with other data-driven methods, trees require large amounts of data.

3

RECAP: ENSEMBLE METHOD

- An ensemble method is an approach that combines many simple "building block" models in order to obtain a single and potentially very powerful model.
- These simple building block models are sometimes known as weak learners.
- Bagging and boosting are examples of ensemble methods, or methods that use a combination of models.



4

ENSEMBLE METHOD (CONT'D)

- The key idea is that different models, such as Logistic Regression or a Support Vector Classifier, or even a Decision Tree, are trained on the same dataset, and each model makes its own prediction; a metamodel then aggregates the predictions of individuals models and outputs a final prediction.
- The result is that the final prediction is more robust and less prone to errors than each individual model.

HOW TO GENERATE BASE LEARNERS

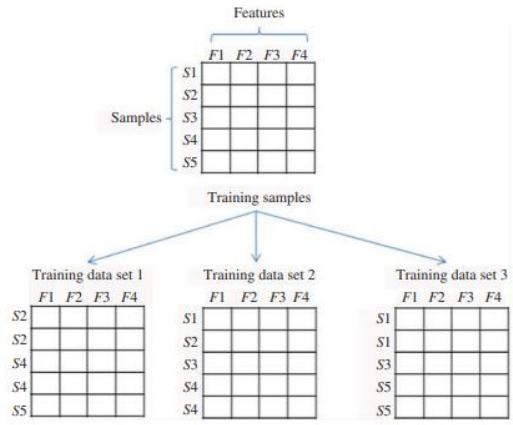
1. Using different learning algorithms for different learning models such as k-nearest neighbor, decision trees, neural networks and least square regression.
2. Using different hyperparameters in the same algorithm to tune different models (for example, different parameters in decision trees).
3. Using different input representations, such as using different subsets of input features in a data set.
4. Using different training subsets of input data to generate different models usually using the same learning methodology.

RECAP: BAGGING

- The decision trees suffer from high variance.
- This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.
- In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets; linear regression tends to have low variance, if the ratio of n to p is moderately large.
- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method.
- To apply bagging to regression trees, we simply construct B regression trees using B bootstrapped training sets and average the resulting predictions.
- These trees are grown deep and are not pruned. Hence each individual tree has high variance, but low bias. And averaging these B trees reduces the variance.

BAGGING (CONT'D)

- By applying the process *T* times, *T* samples of *m* training examples are obtained.
- Then for each sample a base learner can be trained by applying a learning algorithm.
- Bagging adopts the most popular strategies for aggregating the outputs of the base learners, that is, voting for classification and averaging for regression.



RECAP: RANDOM FOREST

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- The split is allowed to use only one of those m predictors.
- A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ – that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

RANDOM FOREST (CONT'D)

- This has a clever rationale. Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- Consequently, all of the bagged trees will look quite similar to each other.
- Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.
- In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

RANDOM FOREST (CONT'D)

- Random forests overcome this problem by forcing each split to consider only a subset of the predictors. Therefore, many splits will not even consider the strong predictor, and so other predictors will have more of a chance.
- We can think of this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable.
- The main difference between bagging and random forests is the choice of predictor subset size m. For instance, if a random forest is built using m = p, then this amounts simply to bagging.

11

PSEUDO CODE

Initialize the number of trees to build (n_trees), the size of the random subsets of the features (feature_subset_size), and the size of the random subsets of the data (data_subset_size)

For each tree in 1 to n_trees:

    Randomly select feature_subset_size features from the full set of features

    Randomly select data_subset_size samples from the training data

    Train a decision tree on the selected subset of data and features

    Store the trained decision tree

For each sample in the test data:

    Predict the target value using each of the n_trees

    Perform majority voting among the predictions of the n_trees to get the final prediction

Return the final predictions

12

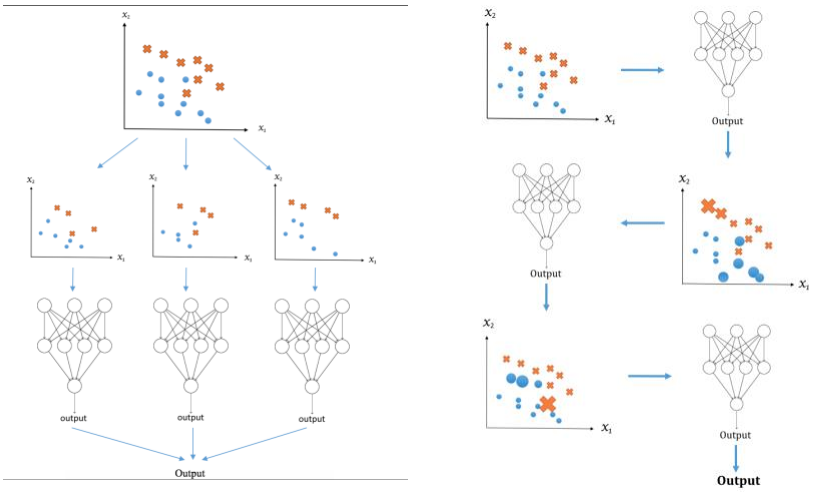
HYPER PARAMETER OPTIMIZATION

- Since hyperparameters are not learned from the data, but must be tuned, we perform a GridSearch on two hyperparameters of the Decision Tree, that is max_depth, and min_samples_leaf, which describes the maximum depth of the tree and the minimum percentage of samples per leaf, respectively.
- In this way, we are going to search for the set of optimal hyperparameters that identifies the optimal learning algorithm and that allows to obtain the best model performances.
- Note that setting n_jobs equal to -1 has the effect that all the available cpu cores are used in the computation phase.

RECAP: BOOSTING

- In boosting, we build the ensemble model incrementally by training each base model estimator sequentially.
- The idea is to combine several weak base learners to form a powerful ensemble. Weak learners are trained sequentially over multiple iterations of the training data with weight modifications inserted at each retrain phase.
- At each retraining of a weak base learner, higher weights are assigned to those training instances which were misclassified previously.
- Each predictor learns from the errors of its predecessor. In particular, the idea is that new trees are created to reduce the residual errors in the predictions from the existing sequence of trees.

BAGGING VS BOOSTING



15

ADAPTIVE BOOSTING (ADABOOST)

- This algorithm was introduced by Freund and Schapire (1997). In AdaBoost each predictor pays more attention to the instances wrongly predicted by its predecessor, by constantly changing the weights of training instances.
- Furthermore, each predictor is assigned a coefficient α that weights its contribution in the ensemble's final prediction.
- Note that α depends on the predictor's training error: basically, we fit the first model on the initial dataset, and the training error for model one is determined. This error can then be used to determine α1, which is predictor one coefficient.

16

ADABOOST (CONT'D)

- α1 is then used to determine the weights α2 of the training instances for model two: here, the incorrectly predicted examples acquire higher weights, and are used to train model two.
- In this case, the predictor is forced to pay more attention to the incorrectly predicted examples. This process is repeated sequentially, until the N predictors forming the ensemble are trained.
- An important parameter used in training is the learning rate $\eta \in (0, 1)$: the learning rate, also called shrinkage, is used to prevent overfitting since it reduces the influence of each individual learner and leaves space for future ones to improve the overall ensemble.

17

Algorithm: Adaboost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Equation 6.66)
(6) **if** $error(M_i) > 0.5$ **then**
(7) reinitialize the weights to $1/d$
(8) go back to step 3 and try again;
(9) **endif**
(10) **for** each tuple in D_i that was correctly classified **do**
(11) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(12) normalize the weight of each tuple;
(13) **endfor**

18

BOOSTING ALGORITHM (CONT'D)

To use the composite model to classify tuple, X:

(1) initialize weight of each class to 0;
(2) for $i = 1$ to k do // for each classifier:
(3) $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(X)$; // get class prediction for X from M_i
(5) add w_i to weight for class c
(6) endfor
(7) return the class with the largest weight;

LEARNING RATE IN ADABOOST

- AdaBoost does not have an explicit learning rate parameter like some other boosting algorithms.
- Instead, AdaBoost uses the weight of each weak classifier to determine its contribution to the final ensemble.
- Specifically, each weak classifier is assigned a weight that is proportional to its accuracy on the training set, with more accurate classifiers given greater weight.
- The weight assigned to each weak classifier can be thought of as a type of learning rate, as it controls the influence that the classifier has on the final prediction.
- However, this weight is determined adaptively based on the classifier's accuracy, rather than being set as a fixed parameter.

IN SKLEARN

- Weight=learning rate*log(1-e/e), where e is the error
- By default, it is provided a value of 1.
- The learning rate depends highly upon the number of n_estimators. By default, it is set to 1 but it can be increased or decreased depending on the estimators used.
- Generally, for a large number of n_estimators, we use a smaller value of learning rate.

GRADIENT BOOSTING

- The Gradient Boosting is another very popular ensemble method, proposed by Friedman (2001) that combines multiple decision trees to create a more robust model.
- In contrast to AdaBoost, the weights of the training examples are not adjusted but each predictor is trained using the residual errors of its predecessor as labels.
- Although gradient boosted trees are very popular within the ML community, we should note that gradient boosted trees notably uses extremely shallow trees, say of depth ranging from one to five, which makes the model smaller in terms of memory and makes predictions fasters.
- Those shallow trees play the role of weak learners, and the performance is improved by adding more and more shallow trees to the predictor.

EXAMPLE: GB*

Car	Origin	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model	MPG
Chevrolet Chevette	US	4	98	70	2120	15.5	80	32.1
Ford Grenada	US	6	250	98	3525	19	77	18.5
Mazda GLC	Japan	4	86	65	2110	17.9	80	46.6
Toyota Corolla	Japan	4	113	95	2278	15.5	72	24
Plymouth Fury	US	6	225	95	3785	19	75	18

- The First thing we do is calculate the average. This is the first attempt at predicting every car's MPG. In other words, if we stopped right now, we would predict the MPG of the cars as the average of all the MPG.
- Here the average will come as (32.1+18.5+46.6+24+18/5) = 27.84
- However, Gradient Boost doesn't stop here. The next thing we do, is build a tree based on errors from the first tree.
- The errors that the previous tree made are the difference between observed and predicted MPGs.
- *Source: https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2

EXAMPLE: GB (CONT'D)

Car	Origin	Cylinders	Weight	Acceleration	Model	MPG	*Average*	*Residuals*
Chevrolet Chevette	US	4	2120	15.5	80	32.1	*27.84*	*4.26*
Ford Grenada	US	6	3525	19	77	18.5	*27.84*	*-9.34*
Mazda GLC	Japan	4	2110	17.9	80	46.6	*27.84*	*18.76*
Toyota Corolla	Japan	4	2278	15.5	72	24	*27.84*	*-3.84*
Plymouth Fury	US	6	3785	19	75	18	*27.84*	*-9.84*

- Now we combine the original leaf with the new tree to make a new prediction of an individual from the training data. But this might lead to overfitting.
- Gradient Boost deals with this problem by using a learning rate to scale the contributions from the new tree.
- So, the predicted value for the first row: 27.84 + 0.1 * 4.26 = 28.266

EXAMPLE: GB (CONT'D)

Car	Cylinders	Weight	Acceleration	Model	MPG	Residuals	Predicted	New_residual
Chevrolet Chevette	4	2120	15.5	80	32.1	4.26	28.266	3.834
Ford Grenada	6	3525	19	77	18.5	-9.34	26.906	-8.406
Mazda GLC	4	2110	17.9	80	46.6	18.76	29.716	16.884
Toyota Corolla	4	2278	15.5	72	24	-3.84	27.456	-3.456
Plymouth Fury	6	3785	19	75	18	-9.84	26.856	-8.856

- We calculate the residuals again with the new predicted MPGs.
- The new Residuals are smaller than the older residuals. So, this is a step in the right direction.
- Now, we combine the tree with the previous tree and the initial leaf. For example, in the first row:
- 27.84 + (0.1*4.26) + (0.1*3.83) = 28.649

Car	Cylinders	Weight	Acceleration	Model	MPG	Residuals	Predicted	New_residual	New_Predicted
Chevrolet Chevette	4	2120	15.5	80	32.1	4.26	28.266	3.834	28.6494
Ford Grenada	6	3525	19	77	18.5	-9.34	26.906	-8.406	26.0654
Mazda GLC	4	2110	17.9	80	46.6	18.76	29.716	16.884	31.4044
Toyota Corolla	4	2278	15.5	72	24	-3.84	27.456	-3.456	27.1104
Plymouth Fury	6	3785	19	75	18	-9.84	26.856	-8.856	25.9704

GRADIENT BOOSTING (CONT'D)

- Practically, there are (at least) three important parameters that one should take care of when tuning a gradient boosted tree:
  1. Number of trees in the ensemble (n_estimators), which controls the model complexity;
  2. Learning rate (learning_rate), which controls how strongly each tree tries to correct the mistakes of the previous trees;
  3. Pre-pruning (max_depth), which controls the number of levels for each tree.
- Note that a higher learning rate translates into a more complex model, since under this scenario each tree can make stronger corrections on the training set.

GRADIENT BOOSTING

- Gradient boosted trees often use very shallow trees, of depth one to five, which makes the model smaller in terms of memory and makes predictions faster.
- Gradient boosted trees are frequently the winning entries in machine learning competitions, and are widely used in industry.
- They are generally a bit more sensitive to parameter settings than random forests, but can provide better accuracy if the parameters are set correctly.

GRADIENT BOOSTING (CONT'D)

- An important parameter of gradient boosting is the learning_rate, which controls how strongly each tree tries to correct the mistakes of the previous trees.
- A higher learning rate means each tree can make stronger corrections, allowing for more complex models.
- In contrast to random forests, where a higher n_estimators value is always better, increasing n_estimators in gradient boosting leads to a more complex model, which may lead to overfitting.
- A common practice is to fit n_estimators depending on the time and memory budget, and then search over different learning_rates.

GRADIENT BOOSTING (CONT'D)

- Another important parameter is max_depth (or alternatively max_leaf_nodes), to reduce the complexity of each tree. Usually max_depth is set very low for gradient boosted models, often not deeper than five splits.
- Their main drawback is that they require careful tuning of the parameters and may take a long time to train.
- The algorithm works well without scaling and on a mixture of binary and continuous features.
- As with other tree-based models, it also often does not work well on high-dimensional sparse data.

PSEUDO CODE

1. Initialize the model F(x) to some constant value (e.g. the mean of the target variable).
2. For each iteration t = 1 to T, do the following:
   a) Compute the negative gradient vector u_t(x) with respect to F(x).
   b) Train a weak learner (e.g. regression tree) h_t(x) to predict u_t(x).
   c) Compute the step size η_t using a line search or other optimization method.
   d) Update the model by adding the output of the weak learner multiplied by the step size to the previous model: F_t(x) = F_{t-1}(x) + η_t * h_t(x).
3. Return the final model F_T(x) as the ensemble of weak learners.

CHATGPT

- In Gradient Boosting, we iteratively train a series of weak learners (e.g. regression trees) to predict the negative gradient of the loss function with respect to the current model.
- This negative gradient is essentially the residual error between the true target values and the current model's predictions.
- At each iteration, we fit a weak learner to the negative gradient, and we use the output of the weak learner to update the current model. Specifically, we add the output of the weak learner (multiplied by a step size) to the previous model to obtain the updated model.
- This process of iteratively adding weak learners to the ensemble helps to reduce the residual errors and improve the overall performance of the model.

CHATGPT (CONT'D)

- So in each iteration, we are essentially learning to correct the errors of the previous model by fitting a weak learner to the negative gradient (residual error) of the loss function.
- We then use the output of the weak learner to update the model, and we repeat the process until convergence or some stopping criterion is met.