ML Notes

1 Decision Trees

Sources
https://gdcoder.com/decision-tree-regressor-explained-in-depth/ (https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html)

Keywords
- supervised learning
- DT can be used for classification and regression (our use case)
- Decision trees are predictive models that use a set of binary rules to calculate a target value.

How it works A decision tree is arriving at an estimate by asking a series of questions to the data, each question narrowing our possible values until the model get confident enough to make a single prediction. The order of the questions (True/false form) as well as their content are being determined by the model.

Splitting deciding the splits affects the trees accuracy.
- regression DTs normally use mean squared error (MSE) to decide to split a node in two or more sub-nodes.
- we need to pick an attribute and a value to split on. → try out all combinations. Find the best one by taking the weighted average of the two new nodes.
- Do that until you hit `max_depth` or when you're left with only 1 element.
- It is never necessary to do more than one split at a level because you can just split them again.
- Instead of using the average (MSE) we could also use the median or... or even run a linear regression model ("fitting a line into points") to make a decision.

Prediction
- Traverse tree until you end up in leaf. The prediction is the average of the value of the dependent variable in that leaf node.

Pros
- If there is a high non-linearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method
- It's kinda robust to outliers
- easy to explain

Cons
- DTs are prone to overfitting. That's why they are rarely used and instead other tree based models are preferred like Random Forest and XGBoost.
- Mostly for classifiction. For regression only if the range of the target variable is inside the range of values seen in training.
- Unstable, small changes can change entire tree (variance)→ can be improved by bagging or boosting

Avoiding Overfitting
1. Setting constraints on tree size (hyperparameters)

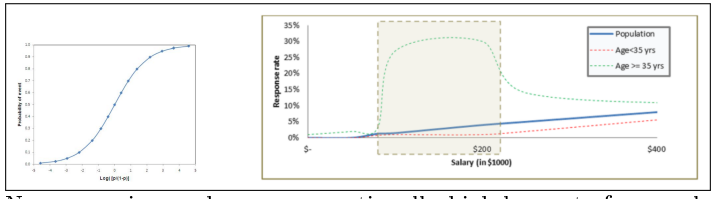
e.g. min-sample-for-node-split, min-leaf-size, max-depth, max-features to consider for split. sklearn lets you define many of these.
1. Pruning
2. Random Forest

Random Forest is an example of ensemble learning, in which we combine multiple machine learning algorithms to obtain better predictive performance. RF can be fast to train, but quite slow to create predictions. Due to the fact that it has to run predictions on each individual tree and then average to create the final prediction.

1.0.1 DT vs (Logit) Regression

- Bsp: chance of buying BWM with regard to salary and age.

Converting variables into buckets (DT) might make the analysis simpler, but it makes the model lose some predictive power because of its indifference for data points lying in the same bucket.
Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. Sth. like this:



Now assuming we have an exceptionally high buy rate for people between 100k and 200k salary that are younger than 35. Both techniques cannot handle this well. DT will fail because it splits one-dimentional. Regression fails as well.

Trick: Combine the upsides of both methods Two possible ways:
1. Introduce a new covariant variable:
$$Z = \begin{cases} 1 & \text{if } 100\text{k}<\text{salary}<200\text{k} \;\&\&\; \text{Age}\geq 35 \\ 0 & \text{o/w} \end{cases}$$
2. Make two alternative models and add the functions in the regression formula as following $H(x) = (1-z) * f_x + z * g_x$, where z is the split in 1.

Note: this technique fails when overall covariance between two terms is high. This is because we will have to create too many buckets and, therefore, too many variables to be introduced in the regression model.

1.0.2 ID3

The ID3 algorithm builds decision trees using a top-down greedy search approach through the space of possible branches with no backtracking (until the branches have 0 entropy).
1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the unused attributes of the set S and calculates Entropy(H) and Information gain(IG) of this attribute.
3. It then selects the attribute which has the smallest Entropy Largest Information gain.
4. The set S is then split by the selected attribute and the algorithm continues to recur on each subset, considering only attributes never selected before.

Attribute Selection Method can be Entropy, Information Gain, Gini Index, Gain Ratio, Reduction in Variance, Chi-Square, etc.

2 Gradient Boosted DT

https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4
Similar to AdaBoost in the sense that both use ensemble of trees to predict a target label. However, unlike AdaBoost, GB trees have depth larger than 1. In practice, you'll typically see GB being used with a maximum number of leaves of between 8 and 32. In GB all trees are equally important.
1. When tackling regression problems, we start with a leaf that is the initial guess / average value of the variable we want to predict.
2. Calculate residuals (actual value vs predicted)
3. Build a tree with the goal of predicting the residuals. Take average if there's more residuals than leafs.
4. Pass samples through the tree
5. Prediction = average/init price + learning rate * residual predicted by the tree
6. Compute new residuals by subtracting actual values from predicted value in 5.
7. repeat 3 - 6 until the hyperparam `numtrees` is reached
8. make the ensemble (initial mean + l-rate * residual$_i$, $\forall$tree i)

to prevent overfitting, we introduce a hyperparameter called learning rate. When we make a prediction, each residual is multiplied by the learning rate. This forces us to use more decision trees, each taking a small step towards the final solution.

Math https://www.youtube.com/watch?v=2xudPOBz-vs
- Need differentiable loss function

Most common for regression: $L(y_i, F(x)) = \frac{1}{2}(\text{observed} - \text{predicted})^2$. MSE would be the same but $\frac{1}{n}$. $\frac{1}{2}$ would not be necessary to determine the best fit, it's only there to make derivatives easier. $\frac{d}{d\,\text{predicted}} \frac{1}{2}(\text{observed} - \text{predicted})^2 = -(\text{observed} - \text{predicted})$.

1. Initialize model with const value $F_0(x) = \underset{\gamma}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

   Where γ is the predicted value we want to find (done by setting the sum of derivatives =0).

Now the first iteration ($m = 1$)

1. Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1 \ldots n$

   This is just the derivative of the loss function with respect to the predicted value. Which is (observed − predicted) for the

case where we have $\frac{1}{2}$ (we have the extra minus that cancels). So there it's equal to the actual residual. But since one can take any loss function, we call it pseudo residual. We the plug in the F_{m-1} which is the last prediction. We do this for r_{im}, where i is the sample number and m is the current tree.

2. Fit the regression tree to the r_{im}values and create terminal regions R_{jm} for $j = 1 \dots J_m$. They may contain more than one r_{im} now.
3. For these regions, for $j = 1..J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
   This is the step where we create 1 value from the samples that ended up in the same leaf. Need the sum when more than one sample end up in one leaf. In we use the $\frac{1}{2}()^2$ loss function, this nicely simplifies to the average of the r_{im}'s in their corresp region.
4. Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ where ν is the learning rate. "Add up the output values (γ_{jm}'s for all the leaves R_{jm} that contain a sample x."

2.1 Differential Privacy

- https://www.youtube.com/watch?v=MOcTGM_UteM
- Differential Privacy protects the data before it enters the model.
- Example fitness data, and Eve wants to find out whether Bob is in the data.

We want to create a function f that wenn applied to the whole dataset, should be as similar as possible to the function applied to the dataset w/o Bob: $f(D_n) = f(D_{n-1})$. In terms of probability distributions: $\frac{P(f(D_n))}{P(f(D_{n-1}))} = e^{\epsilon}$. $\epsilon = 0$ would achieve peak privacy. $\Rightarrow$ Add random noise to the data. $f(D_n)$=D$_n$ + noise. But the more noise you add the less accurate the data becomes. That means $\epsilon = 0$ would add so much noise that the data would not be useful anymore. Therefore it works best for last datasets, as good privacy in small ones makes them kinda unusable.

3 Theos Thesis