Classification

Decision Tree

How is a Decision Tree fit?

- Which variables to include on the tree?
- How to choose the threshold?
- When to stop the tree
  Key Idea
- to choose the feature that has the lowest "impurity" to split our tree,
- reach the decision as fast as possible with smallest height possible.

highest purity means the samples contain mostly the same classes. Vice versa, lowest impurity means that samples contain mostly exactly equal classes.
measure impurity is using Gini index

$$I_G = 1 - \sum_{i=1}^{c} p_i^2$$

,learn Ig

How do we find all threshold for continuous values?

We can first sort. Then we are identify critical value using the midpoint between all consecutive values.
three ways can stop tree:
-For any leaf reaching 0 impurity (i.e., gini index), we stop splitting that leaf.
-For any splitting, where the splitting results in worse gini index by a certain amount, when compared to the parent gini index
-We stop when the tree reaches a certain maximum height we set
we can then use the best decision note as our split node. Then when we go to the next node, we have to repeat again. This algorithm is called CART (Classification and Regression Trees) algorithm, where the recursion keeps on going until certain stop criteria,
Decision Trees are more powerful than other classification in a sense that it can work very well given heterogenous features. However, **the downsides is high possibility of over-fitting** multiple decision trees(random forests) expect better results
— — — — — — — — — — —

Random Forest

DT>overfit. solution> construct multiple trees to reduce variances,
Bagging
B=5,want5trees, D=total datasets du toB=(5), 5datasets needs to change a little bit.
*How?useBagging(Bootstrapping).ReplacementSampling*Each of the tree is trained on a subset of bootstrapping sample and then perform some sort of aggregation of the decision.
Boostrapping the data plus performing some sort of aggregation (averaging or majority votes) is called boostrap aggregation or bagging.When sampling is performed without replacement, it is called pasting. In other words, both bagging and pasting allow training instances to be sampled several times across multiple predictors, but only bagging allows training instances to be sampled several times for the same predictor.
—
Random Forest
A random forest is constructed by bagging random" subset of
Q≤n: features are considered as splitting variables.
randomize features Rule of thumb: q=sqrt(n) for classification trees,
q=n/3 for regression trees
provide feature importance by calculating the decrease in impurity involving that feature, weigthed by how many samples reach that node.
OutofBag(OOB error):measuring prediction error of RF

Hyper Parameter Tuning:max_depth, n_estimators
Note:
In bagging > n estimator (trees) using any algorithm,
RF > n estimator using Decision Trees only.
n=numberofsamples
N=totalnumofsamples
n<=N (in bagging)
Adv of RF:
-The power to handle large data sets with higher dimensionality
-classification and regression
-the model outputs importance of variable,
-balancing errors in data sets where classes are imbalanced.
-we are working with image, sound, brain signal, deep learning remains the way to go.
DisAdv of RF:
-good at classification but not for regression as it does not gives precise prediction (overfit)
-fails when there are rare outcomes or rare predictors,
WorkshopNote:
Bagging = 3
m=4
-bootstrap:m must be the same for B1, B2, B3
-pasting: m is depends
—
Workshop answers:
(1)Bootstrapping:with replacement,m=M
M = SamplesSize
Sampling: without replacement
m<=M
(2)Bagging Work: Bootstrapping the data plus performing some sort of aggregation (averaging or majority votes) is called bootstrap aggregation or bagging.
How reduces Overfitting: by constructing multiple trees
(3) -OOB use validation purpose: oob is a data that is not used in that particular tree (unseen data can use as validation set)
-primary difference between Bagging and Random Forests:
In bagging > n_estimator (trees) using any algorithm,
Using all features in each tree
RF > n_estimator using Decision Trees only, using random subset of features
-RF perform feature importance: by calculating the decrease in impurity involving that feature, weigthed by how many samples reach that node.
-Yes, possible.While individual trees may be weak or make errors, the collective strength of the ensemble allows Random Forests to maintain high predictive accuracy and generalization performance.
— — — — — — — — —

AdaBoost

$$\epsilon_s = \frac{\sum_{i=1}^{m} w_s^{(i)} I(h_s(\mathbf{x}^{(i)}) \neq y^{(i)})}{\sum_{i=1}^{m} w_s^{(i)}}$$

$$\alpha_s = \frac{1}{2} ln \frac{1-\epsilon_s}{\epsilon_s} \qquad w_{s+1}^{(i)} = w_s^{(i)} e^{-\alpha_s h_s(\mathbf{x}^{(i)}) y^{(i)}}$$

$$w_{s+1}^{(i)} = \frac{w_{s+1}^{(i)}}{\sum_{i=1}^{m} w_{s+1}^{(i)}}$$

we use the hypothesis function:

$$h(\mathbf{x}) = \text{sign}\Big(\sum_{s=1}^{S} \alpha_s h_s(\mathbf{x})\Big)$$

take a weak classifier($h_s(x)$), boosting the overall performance. Weak classifier: Decision Tree with max_depth=1 and max_leaf_nodes=2 are often used (stump)
each classifier, define α as *voting power* of that.(-∞,∞)
Higher the alpha, the more we trust that classifier.
hypothesis function is based on a linear combination of the weak classifier .
good classifier should simply has the minimum weighted errors,Ɛ.[0,1]

Initial weight: ∑w = 1 $\quad w_s^{(i)} = \frac{1}{m}$
higher the error, lower is alpha, which means we don't trust that classifier
e=0, α=good
e=0.5, α=negative
e=1, ignore that classifier
by assigning negative alpha values, the algorithm effectively flips the predictions of the weak classifier, so their combined effect is closer to random guessing, reducing their influence on the final strong classifier.(don't use -α)
Workshop
(1) s: number of tree: number of weak classifier
W:determines how importance of that sample in that tree
(2)sumW=1,{0,1}, Ɛ is error
(3)Ɛ:to update the weight, to decide α the of that tree
(4) α decide which tree is more important, confidence for a classifier.Higher α > lower Ɛ.Higher α is better.
(5) $e^{-\alpha_s h_s(\mathbf{x}^{(i)}) y^{(i)}}$ higher means wrongly classified sample, should get bigger weight next time.
(6) new weight=>
(7) early stopping, reached number of classifiers, or certain error threshold is reached. If α is getting negative, delete α.
—————————————————

Gradient Boosting Regression

Similar to AdaBoost,
Gradient Boosting works by adding sequential predictors
instead of adding **weights**, this method tries to fit the new predictor to the **residual errors** made by the previous predictor.
Stop creteria:
When we reached desired iterations
When the residual does not decrease further using some validation set
Advantages:Extremely powerful-for heterogeneous data (e.g., house price, number of bedrooms).
Disadvantages:
-cannot be parallelize: since sequential data
-easily overfit: (carefully choose n_estimators, regularizaiton(e.g.max_depth)
homogeneous data such as images, videos, audio, text, or huge amount of data, deep learning works better.
—
Workshop
(1) differences between AdaBoost and <u>Gradient Boosting:</u>
-XG:no alpha is applied to the first predictor, because learning is "sequential"
-XG:all alpha share the same number|Ada:alpha diff across all predictors
-Ada:fixed err of prev tree.
XG-fixed err of all trees

① AdaBoost	Gradient Boosting
① learn α	- α is hard code. (α is same)
② learn from previous tree.	- learn from all tree.
③ h_0 has α	- h_0 has no α

(2)main idea: predict the residual of previous model to be closer to y
(3)not set α, (0,1).α same across the model
(4)α:learningrate-reduce overfitting.

-DummyRegressor(strategy='mean')
DummyClassifier(strategy='most_frequent')
Extra:
loss function for regression:

$$J = \frac{1}{2}(y^{(i)} - h(\mathbf{x}^{(i)}))^2$$

Find h that minimize J.
—
Heterogeneous|Homogeneous Features.
XGBoost is the best for tabular data
DeepLearning:audio,images
—------------------------------------
Unsupervised Learning
Clustering(unsupervisedL)
KMeansClustering:cendroid based
clustering problem, we are given a training set,and we are tasked to find y. (unsupervised)
-Not all unsupervised find y.
-Dimensionality reduction don't find y
-samples: customer segmentation or image segmentation or for visualization purposes
-E-step" or Expectation step:updating our expectation of which cluster each point belongs to $(xi - ci)^2$ >>
$(x1 - c1)^2 + (x2 - c2)^2 + (x3 - c3)^2$
The "M-step" or Maximization step: defines the location of the cluster centers which accomplished by taking a simple mean of the data in each cluster.
$(x1 + c1)/2$,$(x2 + c2)^2$,$(x3 + c3)^2$ = center=[1,1,1]
-stop(1) max iterations, (2) samples no longer move to another class.
Workshop:
(1)choose-k:
-principleBased:need domain knowledge, *used when want high explainability.*
-dataBased:min(TotalwithinClusterVariation) or elbow method which compute within clusters distances.
(2)fail when data are not spherical distribution means kmeans need equal size of samples
—-------------------------
GaussianMixtureModelClustering
-EM algorithm:Notice that r actually depends on mean, covariance and pi. But then, mean, covariance, and pi also depends on r. Based on this, we can use EM algorithm, where we can (1) create a random mean, covariance, and pi, (2) calculate r and then repeat 1 and 2 until certain stopping criteria.
-Pros:Address the limitations of K-means - Can be used to generate data, since we know p(x|y)
-Cons: - Just like K-mean, this algorithm can sometimes miss the globally optimal solution, and thus in practice multiple random initializations are used.
Workshop
(1) Diff GMM & KMeans
-multi-shape| only circle
-find μ,N,r,π| fin d μ
-work with various size | same size
(2) μ:initialize randomly
(3) shape μ:for1cluster=(n,1) and all cluster=(n,k),n:feature
(4)1μ for (1feature,1cluster),
Since k=3, each sample has 2feature.[row:n, col:k]
(5)random initialize covariance matrix,shape=1/k
(6)1cluster=4covm, 3cluster=12covm
(7) π=p(xbelongstoclusters)| prior|p(y), shape=(k,)
(8)yes, π sum up 1
(9)r=p(y|x), likelihood for each sample for each cluster,(m,k)
(10)primaryObjectiveFunc:toMax

$$\prod_{i=1}^{m}\sum_{c=1}^{k}\pi^{(c)}\mathcal{N}(\mathbf{x}^{(i)}|\mu^{(c)},\Sigma^{(c)})$$

(11)stop:(1)maxIteration(2)Center do not move(3)not changing μ,N,r,π

InClustering:wecan'tchooseFeture base on correlation since we don't have y. We choose based on domain knowledge.
—
DimensionalityReduction
PCA:best line is the line that can maximize the variance.
–Workshop–
(1) why do DR?
-to visualize data
-to do modelling
(2)shapeofProjectionMatrixB
(n,n')
(3)n' = randomly
(4)PCA-maximize the variance of reduced data.In layman terms, we want to find the **direction** of the projection vector that maximizes the variances, thus the **projection vector must be a unit vector**.
(5) $\frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{z}_1^{(i)} - \mu_1\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{z}_1^{(i)}\right)^2$ assume **our data mean=zero**, which will makes thing easy.
(6)Lagrange method: (optimization)
According to our high variance equation: Var(z1) = $\mathbf{b}_1^\top \mathbf{S}\mathbf{b}_1$, we
We need tp restrict all solutions to
$\|\mathbf{b}_1\|^2 = 1.$ in order to change the dir not the magnitude. So, we use Lagr.
What are h(w) and f(w)?

$$\mathcal{L} = f(w) + \sum \alpha \cdot g(w) + \sum \beta \cdot h(w)$$

f(w)= $\mathbf{b}_1^\top \mathbf{S}\mathbf{b}_1$ and
h(w)=(1- $\mathbf{b}_1^\top \mathbf{S}\mathbf{b}_1$). β=w
(7)b=eignvector(the **primary** directions in which your data varies the most)
The variance is informed by its corresponding **eigenvalues**,β.
(8) There are two ways to find eigenvectors: 1) Eigen decomposition and 2) Singular Value Decomposition.
(9)shape of eignvector:(n,n)
Shape of eigenvalues:(n,1)
Note:3features, 3eigenvector, 3eigenvalues. If 2D-1D projection, choose 1 the largest eigenvalues. If 3D-2D, choose 2 largest eigenvalues.
(10)we take n' columns of Q depending on how many columns we want to take.

Projection $\mathbf{z} = \mathbf{X}\mathbf{Q} \in \mathbb{R}^{m \times n'}$
Projecting back: to know how much information have loss.
Loss=var(x|orig)-var(x'|project)
—

Deep Learning

ANN (for tabular)

-InputLayer|Hidden,intermediate(weight/parameter)|OutputLayer
Code:
-Pytorch-tensor: do gradient descent for every steps
-Cal gradient with loss: loss.backward()
-if most of the thing are in 1000, lr should be around 0.0001.

```
-model = nn.Sequential (
   nn.Linear (10, 24),
   nn.Linear (24, 12),
   nn.Linear (12,1)
) # any number of matrix multiplication can be approximated into one matrix multiplication.
```

-So we use Relu btw nn.Linear as an Activation func.(relu>stable gradient descent)
-Max amount of gradient Sigmoid = 0.25

-In [10x24]matrix,Totalparams= 240, 24feature>24 bias
—
CNN (for images)
-Input can be image pixels
4 x 4 pixels = 16 features
-0 to 255>normalized > 0 to 1.
-near pixels have relationship
-Linear combination of all the **localized matrix**->called featuresMap= OriginImage@convolutionalLayer (3x3, 5x5, 7x7)
-convolutionalLayer|weight|kernal
-I**nputShapeofimage=(batchsize,Inchannels, height,width):** gray (1,14,14), RGB(3,14,14), CMYK(4,14,14)
-1 featuremap>1weight
-if have k filters>k featuremap
-In a convolution operation, there are 3 main hyperparameters to fine tune (1) filter size (2) padding (3) stride.
-**ShapeofFilter=(inChannel,outChannel,filter width,filter height)**3x64=192
in3out64filter3x3=(3,64,3,3)
-afterConvolution,someinfoloss.To prevent, use Padding=
$\frac{K-1}{2}$,k=filter size.
-Output(batchSize,out channel,outHeight,outWidth)

$$O = \frac{I - F + 2P}{S} + 1$$

Stride: big number, more skip, will drop resolution
Code:
torch.manual_seed(20) to replace cross_validation as we cannot do cv in deeplearning
-batchsize bigger better:but CUDA out of memory (reduce batch size)
-for Conv matrix: Conv1d for text, timeseries,signal; Conv2d for image; Conv3d for fmri, mri
-relu>not chang to pos num, set 0 to neg num

```
# 5. calculate gradient
     optimizer.zero_grad()
     loss.backward()
# 6. update weight
     optimizer.step()
```

—
RNN (specialized for signal, text, timeseries) where previous affects the next. (LSTM)
-all weights are the same

$$\mathbf{h_t} = f(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b})$$

h is the hidden size, and
$\mathbf{h}_t \in (h,)$ is the hidden state at time t,
$\mathbf{x}_t \in (n,)$ is the input at time
$\mathbf{W}_x \in (h,n)$ is the weight matrix for inputs to hidden layer,
$\mathbf{W}_h \in (h,h)$ is the weight matrix for hidden layer to hidden layer,
$\mathbf{b} \in (h,)$ is the hidden bias ,
f is the activation function.
When working with LSTM models, we start by dividing the training sequence into a series of overlapping "windows".
-shape of xt=(n,1)
ht = (h,)
Wh=(h,h)
Wx=(h,n)
b=(h,)
F:activation function