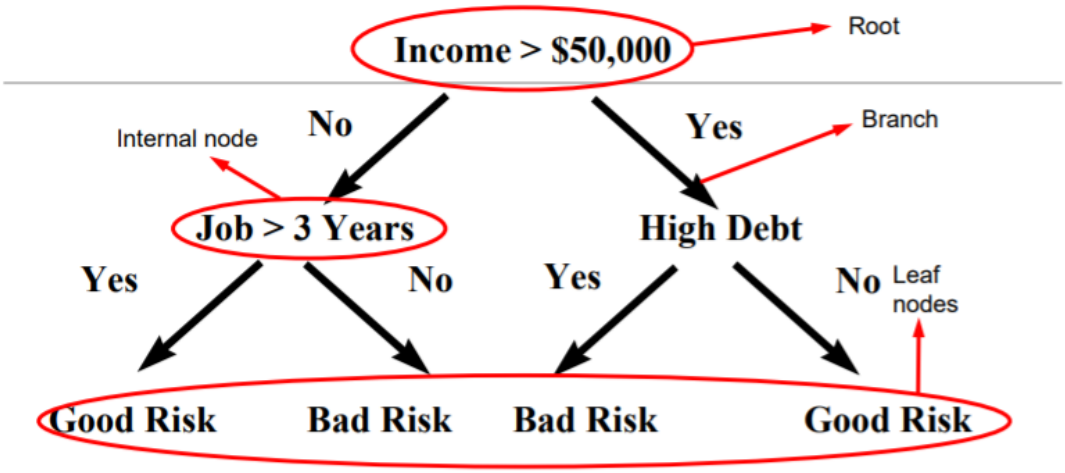
Classification and Regression Trees

- This is a very old model with limited usefulness, but the underlying methodology became the foundation of the most powerful model we currently have for structured data (XG Boosting and Random Forest) in terms of raw prediction power
  - The properties of the trees make them very bad to use on their own.
  - But it turns them into very useful model when used as an ensemble
- The good thing about the card models is that requires only minimal data cleaning
  - It only requires you to remove null values. The ones that are completely random
    - Ones that usually be replaced with median
  - And treat the outliers that you don't want in the model
- The WoE operator, in the background, is running one of these models
- The cuts are created using this model

Logic of the decision trees



- A decision tree is just help you make a decision
  - With many variables, this tree could be very big

How to build a tree

- Need to make three decisions
  - Start with a set of variables, and a cleaned data
    - Recall that valid null values and outliers should be treated
    - You need to do a statistical significance check: every category should be statistically significant

- The first thing that needs to be decided is how to create splits
- When to stop creating splits
- Finally, how to assign a certain leave to a certain class

Assignment decision

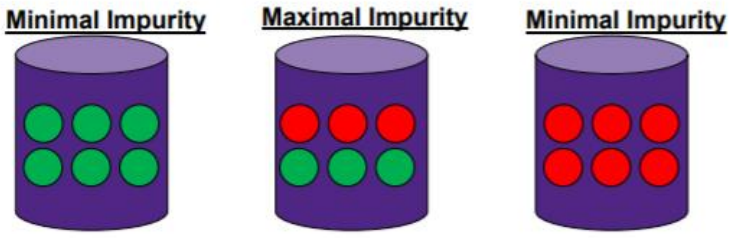
- The assignment decision
  - Simply use the majority class. Usually, this is normalized, so you calculate the percentage of goods and the percentage of bads and the one that's higher is the one that gets assigned
    - E.g. If I have 70% of goods and 30% of bads in the category, assign good to this leaf
    - Alternatively, some trees return the probability, so they will return 0.3 (30% chance of being a 1, or being a defaulter)
  - If you have a regression tree where you have a continuous variable, then the output is simply the average of the elements in this leaf

Split decision

- The splitting decision
  - Entropy: the level of disorder in a system
  - To data science: this is a measure of knowledge that you have about a certain phenomenon
    - E.g. if I know very little about the person in terms of default, so it's 50/50; if I know for sure the person is a defaulter, then I know a lot; if I know for sure the person is not a defaulter, then I also know a lot
  - Entropy is a numerical measure of that knowledge and is calculated as

$$E(S) = -p_G \log_2(p_G) - p_B \log_2(p_B)$$

    - Assumption: $0 * \log_2(0) = 0$
  - Calculate the entropy for the following



  - The first minimal impurity case

$$-1 * \log_2(1) - 0 * \log_2(0) = 0$$

    - No disorder
  - The maximal impurity case

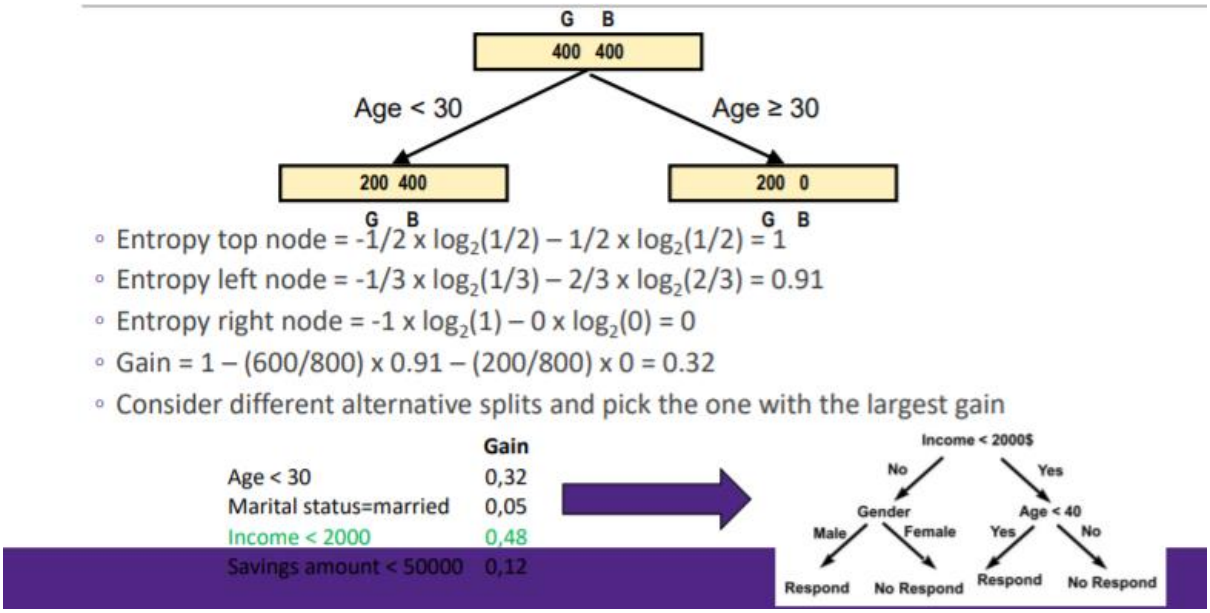
$$-0.5 * \log_2(0.5) - 0.5 * \log_2(0.5) = 1$$

    - Maximum entropy, maximum disorder
  - Calculating the logarithms are very hard before, so they use the Gini index for approximation

$$Gini(S) = 2p_G p_B$$

- ✧ Almost all packages include Gini as a measure, there would be not reason to use this instead of entropy because entropy is way more superior, but if you have billions of trees, it makes sense to use something like Gini even today
- ✧ Running time of logarithm and multiplication is in nano seconds, but with billions of data, this do make a difference

- Now, how do I build the model: calculate the entropy gain
  - ➢ The computer will be running the same thing as the WoE
    - ✧ It is going to create a segmentation of the variables, take all the categories in a categorical variable, and calculate how much entropy gain there is.
  - ➢ When a split is created, and it is going to simulate dozens of cuts across all variables that you have
    - ✧ The number of simulations is usually 100 cuts per variable * # of variables if use a mean percentage being of 1%
    - ✧ And for each one of those simulations, the gain of entropy is calculated
  - ➢ Example:



- So we want to see whether 30 is a good cut for age
  - ➢ Whether age less than 30 and age more than 30 produce good entropy compared to other cuts
  - ➢ At the top node before the split, the entropy is 1
  - ➢ After splitting, the entropy on the right has an entropy of 0, the entropy on the left has an entropy is 0.91
  - ➢ Then the gain is the original entropy - weighted sum of new entropy

- To build a tree, we will start with this, 100 cuts per variable, and for each variable's cut, calculate the gain, then the model will pick the highest gain
  - ➢ So if in the previous example, the age < 30 has a gain of 0.32, marital status = marries is 0.05, income < 2000 is 0.48,…, then it's going to pick income, so the first cut is going to be income
  - ➢ Then split into two by applying income < 2000 criteria, then for each of the two, do the whole

testing process all over again calculating new gains for new cuts, and pick the variable and the cut with the highest gain
  - This is called the greedy algorithm because they always eat the largest amount
  - Problem is, the greedy algorithm is inefficient, so they do not reach the optimum

Stopping Decision

- The thing is, we can keep gaining, and making the variables bigger and deeper until we have one case in every leaf
  - Then the entropy is 0 for every leaf
  - The bias of the model is 0 but the variance of the model will be huge
  - Using a validation sample
    - So, from the dataset, I am going to take one part out, and leaving it as a test set
    - Then the remining is the training set
      - For logistic regression, this is enough, because it doesn't grow every iteration like a tree
    - A tree can keep growing until infinity until optimal is 1 case per leaf.
      - If I use errors in the test set as an indication of when to stop, this is cheating because I am using test set to learn patterns even though it is not actively used to optimize.
    - So I need to use a validation set
      - Take a little piece of the training set, that is not used for training
      - Validation set is only used to stop training.

Advantage and Disadvantage of the tree

- Disadvantage
  - What happens, just of the bad luck, the very first variable (in our tree was originally income), the first variable is age, the final tree will be completely different
    - Because of bad sampling, the gain on entropy from the sample for income was accidentally higher than the gain on entropy from the income when the inverse should have happened.
  - This is an unstable model

- Advantage
  - They are easy to interpret and they are transparent
    - You can see where the cuts are made
  - They are not parametric
    - Entropy rules them all
  - You don't need to transform variables, normalize variables.
    - They consider every variable independently
  - It's very robust to outliers
    - It is making cuts, and one case is not going to change the entropy very much
    - That's why you only need to consider outliers that are outside the parameters of the model

- They are great for preliminary data screen
  - A new dataset can be jut put in the tree and see how it looks like, just get an idea of what variables are more important
  - This could be the first model to use

- While using one tree is a very bad model, but a lot of trees build over subsegments of data are extremely powerful

Random Forest

- The tree algorithm is unstable, but we could use this to our advantage
  - Learning is looking for patterns in my data
  - But if I have a data set of 100 million people, thinking that the same patterns will apply to every sub-segment of the data doesn't make sense
    - Some groups will have certain unique patterns that are unique to them
    - This is legitimate, but they are not optimal for the whole group, they are only good for the sub-group
  - Also, since the trees are greedy, if one variable is dominant in the population, then it will always be dominant while it may not be correct to certain sub-segment
    - Greedy means trying to maximize a very narrow minded objective
- So there comes the random forest algorithm
  - It will try to look for sub patterns over samples of both data and variables at the same time
    - So if you have 100 million cases and 100 variables, it will look for patterns over 30 million cases and 20 variables
    - And because of this, it will conduct a much deeper search
    - This idea borrows from bootstrap
    - Note that random forest could take sample with replacement, but it is advised against it, and we have a better model for that.
      - When the sample is so huge, the probability of taking the same sample twice is negligible
      - So if the sample size is small, then should sample with replacement
  - Random forest is a big data model, it shouldn't be used on small samples, it won't work
  - Note that random forest is less sensitive to parameters compared to other algorithms
    - Logistic regression, neural network is very sensitive to the number of parameters that we have available
    - For random forest, there is only 1 key parameter, which is the number of trees you train
      - This number should be as big as you can make it
      - But this is inefficient
  - Random forest, in theory,
    - Cannot overfit, you we could train, train, and train
    - It is optimal

The Random Forest Algorithm

- So we take a dataset with N cases and V variables
  - V is usually at least 30-50 variables
  - N is usually 100,000 cases

- Then we will choose four parameters on our own with some pointers
  - 1. Percentage of cases, sub-sample size
    - Normally that percentage is around 66% of N
    - In reality, it's usually 63.2%, use that number because

$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} = 0.632 \ldots$$

      - If you sample with repetition and do it infinite number of times, the probability of a random element being chosen is 63.2%
  - 2. Number of variables
    - Usually, this number is 1/3 of V (for around 30-40 variables) or square root of the V (if you have a lot of variables)
  - 3. Number of trees
    - It should be as much as you need them, however, how do you find that number?
      - Rule of thumb

$$10^{floor(\log(N)-2)}$$

      - If you have 100,000 cases, use 1,000 trees, if you have a million cases, use 10,000 trees, if you have 10 million cases, use 100,000 trees
  - 4. A minimum leaf size
    - If you observe that the random forest is taking too long to train, you may want to restrict how much the tree grows
    - This comes at a predictive cost, so should not do it unless necessary (low memory or taking too long)

- Once the numbers are determined, take a sample of the sample size randomly

- Then create a tree
  - Only choose from a subset of variables for each level where it loos for cuts only from the selected variables
  - This takes on use of the unstable property of the tree, we know the root will be different most of the times, so it gives different patterns
  - At the next level of the tree, for each criteria, we again choose a subset of the variables
    - So for example, the first cut is income < 1000, now for income < 1000, we choose a subset of variables to build upon that, and for income >= 1000, we again choose a different subset of variables to build.

- Then we sample again, and we build a new tree

- After having all of the trees, we put them together by
  - ➢ Averaging the prediction of each tree
    - ✧ Every tree will boat on what class the element is and your average of the probability or the selections it will give you
  - ➢ For example, assume we have 2 classes, defaulters and non-defaulters
    - ✧ 900 trees tell you it's good and 100 trees tell you it's bad, so the output will give you is 0.1

- Note that if you have smaller number of cases and sample without replacement, I will not have patterns to learn, and it won't be very useful. Or if I have a small number of variables, I can't achieve randomness
  - ➢ What to do if small cases and large variables? Take samples with replacement
  - ➢ But you can't get around with few variable cases

Theory: Why does it work?

- Each tree can be considered a Random Variable, correlated with the other trees by a factor ρ. The final variance is given by:

$$\sigma_{RF} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

- As B grows, the fraction $\frac{1-\rho}{B}$ gets smaller, and by sampling p variables, the correlation between trees ρ also goes down!

- Reaches better variance-bias trade-off level.

- With subsamples of both variables and cases, the correlation between the trees will be low
- And as the number of trees (B) grow, the second coefficient would be small as well
- The variance of the random forest is lower than the variance of the combination of the trees built traditionally.
  - ➢ Give better variance-bias trade-off level so it is more efficient

Stochastic Gradient Boosting

- Also focusing on creating diversity in the data
  - ➢ This method will work perfectly over small data
  - ➢ But this is extremely sensitive to parameters, getting it right is difficult.
  - ➢ It requires parameter tunning
    - ✧ Need to find out and understand which parameters are working and which aren't
- Analogy
  - ➢ A random forest is getting a bunch of experts, and ask them a question
    - ✧ Each one independently gives the answer

- Alternative,

- Now, we cerate diversity by limiting how big the tree gets
  - We are going to make the next tree learn from the mistakes of the one before it.

- You will need a lot less data and trees for this to converge
  - Usually around 100 – 500 trees/ 50-500 trees
  - This is not determined
  - This can overfit
  - So you want to stop training when appropriate.

Parameters and Process

- First, we select the number of trees
- Second, we select the maximum depth of each tree
  - The trees need to be small, so the depth will always be 2, 3, 4, and no more, 4 is pushing it.
- Then do the same process as random forest
  - Select around 63.66% of the cases with repetition (designed for small samples)
  - Select 1/3 of the variables
    - But now, you do not subsample the variables
  - You simply build the tree
    - Build the tree using the target variable as the target
    - So you try to create the best tree of pre-selected size (usually 3)
    - The best tree you can create using the original
      - The first model will be very crappy
    - Now go to the dataset, and calculate the errors in predicting, ε_{tree1}
    - Now, select the target variable as the error
    - And repeat
  - Now, the fourth tree will learn from the error of the first, the second, and the third tree, and so on.
    - Every tree will get some knowledge out of the errors of the ones that came before

- Call this gradient because of using the errors

- Now, it is the completely opposite logic, we will create a few small and correlated trees
  - But have this strategy of learning one step over the next or the next,….

- This has won every competition in structured data
  - Structured data: something fits in a table

- Theoretically, random forest and SG boosting perform equally because they follow the same principle
  - But in practice, SG boosting tends to converge faster to the right solution if you tune your parameter well

- Parameters need to be tuned
  - Number of trees, maximum depth, and sometimes the selection process of parameters
  - One more parameter, importantly, is the learning rate
    - The new target variables is

$$y = |y_{real} - \alpha * \varepsilon|$$

    - This is extremely important because it shrink the estimates so that we don't overlearn.
    - What kills the SG boosting model is learning too much in the first few shots
      - If you learn too much, you leave nothing left to do the deep space search
    - Slow and steady wins the race
    - The typically learning rate for SG boosting is between 0.1 and 0.3
      - Sometimes, even smaller
    - There is a tradeoff between the learning rate and the number of trees
      - The lower the learning rate, the higher the number of trees needed
    - There is a sweet spot, we need to do deep space searches in order to get the good number
      - Very complex dataset will require low learning rates and a lot of trees

- If you have structured data and you have no problem that your model is nonlinear, then this is the way to go
- There is a big problem
  - These are non-linear, this could be illegal (e.g. application scoring)
    - We can use it for LGD model, behavioral scoring model, generally internal models
- The U shape problems in the WoE, the model will deal with these just fine
  - These models are called universal approximators
  - This is the first model that we see that has the property of infinite approximation

- Difference in building a SG boosting and random forest
  - Random forest has more number of trees, deeper in the tree, and can be trained in parallel
  - Training a random forest is efficient, training a SG boosting is not, it requires sequencing.

XG Boosting vs. Random Forest

- In theory, they should perform the same
- Random forest requires less tuning
- SG boosting models can be a lot smaller
- SG boosting is robust to small sample size
  - Size below 100,000, do not use random forest

Error Measures

- Validating a model is working correctly is exhausting.
  - A lot of tests
    - Should test all of them, if one fails while others don't, should explore why
  - We focus on prediction measures

How to measure performance

- The question: how well does the estimated model performs in predicting unseen cases?
  - There are many strategies, all of which has reasons not to use them, so we should use them all.
  - Easiest way and absolute minimum: Split Sample Method; train vs test (maybe 30%)
    - Should aways set a side a test set unless sample is too tiny
    - Best guess of how good the model will perform
    - Test set should never be used until the final state of the report
    - Usually, stratify sampling
      - If not, then will have unbalanced data, and you will think that your model performs better from one side to the other just because of this unbalanced.
  - N-Fold cross validation: in order to get a confidence interval of the performance #1
    - This will directly produce an uncertainty band
    - This is a cheaper way to calculate than using bootstrap
    - This will give you a standard deviation, and an error measure
      - Calculate average, and the standard deviation
    - Normally, will do 10 by 10 (10-Fold CV)
      - Run CV of size 10, then you reshuffle, and you repeat the process 10 times, you get 100 values
    - But this is not the confidence interval for the test set
    - For smaller samples, this is the one to use over splitting (can't split)
  - Bootstrap
    - This will give you the error that you can use to shift the test set
      - So gives you a confidence interval over the test set
    - You still training it on the training set, but the dispersion you get applied to the test set
    - Same thing as the random forest, now we sample with repetition, and train models, the number of models depends on the target (financial world: 10,000, maybe too expensive)
    - Bootstrap requires a good number of repetitions for it to converge
    - You take a sample of size 63.2% from the train set
      - You train you model,
      - In the remaining 36.8% that you didn't select, calculate the performance
      - Then through everything back, to the total set, you sample again with repetition, so on, …

- ✧ Theory: the standard deviation of the value from above process can be applied directly to the test set to get the confidence interval over the test set
  - ▫ You don't calculate any average

- There could be a tradeoff
  - ➢ You may want a worse mode which produce worse average, but more stable and doesn't have much dispersion
  - ➢ Banks usually prefer to use simpler models with lower dispersion
    - ✧ Losses can be controlled.

Performance Measures for Classification Models

- Confusion matrix
  - ➢ How much the model is getting confused in selecting cases.

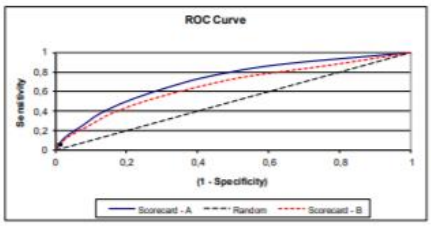
- Depends on the cutoff
  - ➢

- The sensitivity is how sensitive the model is to the class labeled as 1
- Specificity is how sensitive the model is toward the negative class (0)

Receiver Operating Characteristic (ROC) Curve

- Make a table with sensitivity and specificity for each possible cutoff.
- ROC curve plots sensitivity versus 1-specificity for each possible cutoff.

Cut-off	sensitivity	specificity	1-specificity
0	1	0	1
0,01			
0,02			
....			
0,99			
1	0	1	0



- In a credit scoring context, the sensitivity is the percentage of goods predicted to be good, and 1-specificity is the percentage of bads predicted to be good.
- A perfect model has sensitivity of 1 and specificity of 1 (i.e., upper left corner).
- Scorecard A is better than B in above figure.

- 1 – specificity is error made of negative prediction
  - ➢ Error made by predicting that the customer is good

- How do you compare intersecting ROC curves?
  - No stochastical dominance! One model better than the other in different segments of curve.
- ROC curve can be summarized by the area underneath (AUC); the bigger the better!
- The AUC provides a simple figure-of-merit for the performance of the constructed classifier.
  - **Intuitive interpretation:** estimate of the probability that a randomly chosen instance of class 1 (good payer) is correctly ranked higher than a randomly chosen instance of class 0 (bad payer) (Hanley and McNeil 1983).
- A straight line through (0,0) and (1,1) represents a classifier found by randomly guessing the class and serves as a benchmark. Hence, a good scorecard should have an AUC larger than 0.5.

- The curve measure that correct operating, characteristics of the model
- The curve is a simulation
  - We can see what model is better
  - The diagonal means no prediction, random
    - You are hitting the market the same percentage that you're making mistakes.
    - Anything above that is learning more
    - If the cure is under the random model, that means the target variable is flipped
    - It gives you a way to compare cutoffs
- The blue line is above the red line
  - The blue model is always better than red model
  - This is called the stochastic dominance
- What if they intersect?
  - There is no stochastic dominance
  - AUC gives an average

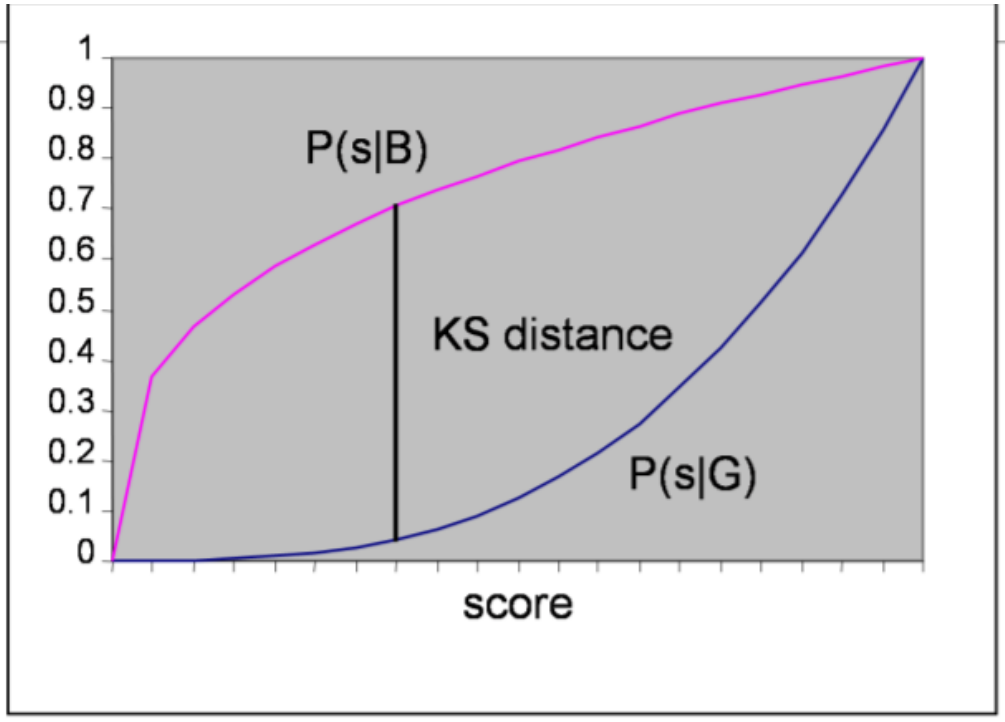
- To summarize the ROC curve, we calculate the area under the curve
  - Called the AUC
  - Between 0 and 1
  - Tells you how good the model is
    - Usually > 0.65 indicates good for application scorecard, > 0.8 good for behavior scorecard

- If there is a box with goods and a box with bads, if you close you eyes, and pick one from the each,, the AUC is the probability that 1 has a higher probability than the 0, so they are ordered correctly
  - If you take two randomly, it's the probability that they are order correctly

The Kolmogorov-Smirnov Distance

- Separation measure
- Measures the distance between the cumulative score distributions P(s|B) and P(s|G)
- KS = $\max_s$ | P(s|G)-P(s|B) |, where
  - P(s|G) = $\sum_{x \le s}$ p(x|G) (equals 1- sensitivity)
  - P(s|B) = $\sum_{x \le s}$ p(x|B) (equals the specificity)
- KS distance metric is the maximum vertical distance between both curves.
- KS distance can also be measured on the ROC graph:
  - Maximum vertical distance between ROC curve and diagonal

- Calculate whether two distributions are independent



- If you have a probability of being one, and you have a model
  - ➢ You would expect the ones to be closer to the right

- In general, application scorecard will have between 25% and 40% chaos
- Behavioral scorecard will have

So with 70% goods and 30% bads in the leaf, I am taking this as good, that means, in this category, I am predicting 30% bads as goods, problem!!

If I have a huge tree with only 1 case in every leaf, this implies that I perfectly categorized all the customers considering all the possible variables (income, debt, …). This suggests that I am able to assign a correct and unique rating (relative probability of default) for everyone. Doesn't this imply that my prediction would be much more accurate? Because I break down the variables so detailed that I make a 100% accuracy in predicting the training set. From my point of view, if a new customer has the exactly same behavior of the existing customer in my training set, the outcome for default will be very likely the same. Maybe a minor difference in behavior will change the outcome, but if my tree is so detailed, in theory, it should capture every difference in behaviors and assign a correct rating based on that using the training set.

This means I can categorize the next new customer perfectly in the tree in predicting his default behavior, I am thinking that this could actually make the prediction even better because all the information is being considered. Unless the information about the new customer is incomplete, in which, we can't make a decision

For the bad luck part of selecting a different first variable and cut, how can we choose age over income if the income is producing a higher gain on the entropy? Is it because of the sample is bad and accidentally causing the gain on entropy of age to be higher than income so that I would choose age over income?

```
min_perc_fine_bin=0.01,
```
in the bins cutting

error from CV vs error from splitting
different model

bootstrap vs. CV: both calculate average and sd of errors from performance over test sets