CQF FitchLearning

CORE LECTURE

Decision Trees and Ensemble Learning

Dr Richard Diamond

May 2022

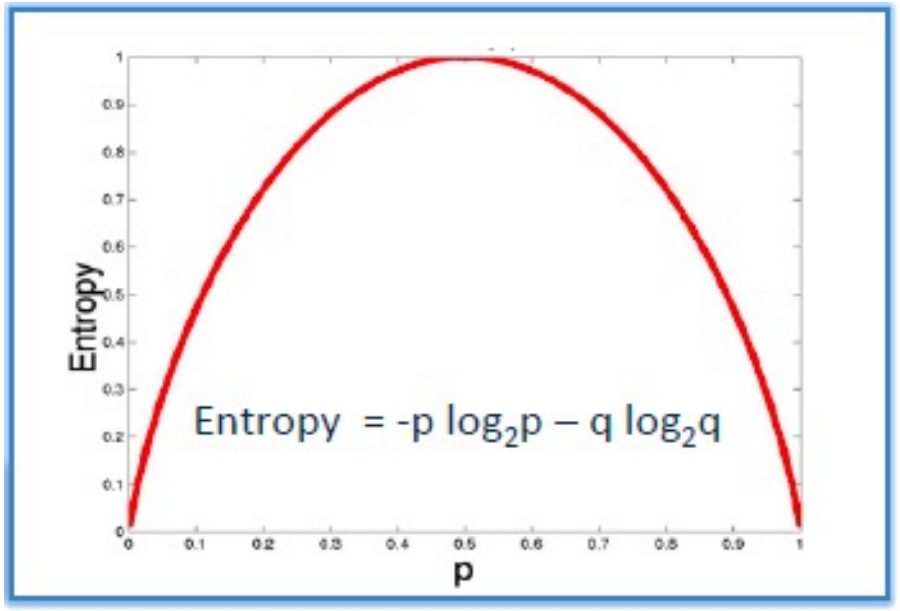
Knowledge | Skills | Conduct

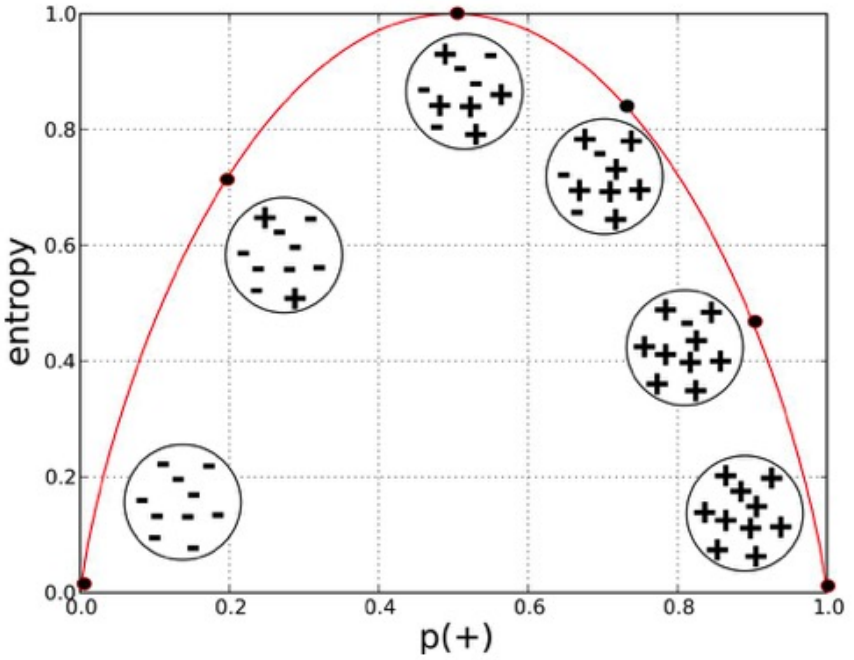
Learning Outcomes

By the end of this lecture you will…

- understand the maths and essentials of optimisation behind decision trees
- be able to operate with hyperparameter tuning
- understand and be able to use simple schemes for bagging and boosting (and get a glimpse of theory behind in-sample and extra-sample error)
- know that AdaBoost gives affine maths but trying to improve prediction of outliers is not necessarily good strategy

Entropy Minimization





Entropy = -0.5 $\log_2$0.5 – 0.5 $\log_2$0.5 = 1

$$Entropy = -\sum p(X) \log p(X)$$

here p(x) is a fraction of examples in a given class

Loss Functions

Where we had one loss function for Logistic Classifier, now we have **three**!

We define the proportion $p_i = \frac{N_i}{N}$ for the number of observations at leaf (or branch end) N_i

- **Entropy**

$$-\sum_{i=1}^{N} p_i \log_2(p_i)$$

Loss Functions (cont)

- **Gini impurity index**:

$$\sum_{i=1}^{N} p_i \left(1 - p_i\right) = 1 - \sum_{i=1}^{N} p_i^2$$

The minimum reached when for one class label $p_i = \frac{N_i}{N} = 1$ (other class labels' probability is zero.)

- **Misclassification error**

$$1 - \max_i p_i$$

Lack of differentiability means we can't look for solutions in more complex set up.

Main Optimisation 1

For feature k, the optimal split $c^{k,*}$ is one for which the total dispersion over the two subgroups is the smallest:

$$c^{k,*} = \underset{c^{(k)}}{\text{argmin}}\ V_I^{(k)}\left(c^{(k)}\right)$$

for example, take Altman Z-score feature RE/TA and by iteration $0, -0.1, -0.2, \ldots, -2$ arrive at specific optimal value

$$c^k = -1.9$$

Our empirical Tree Regressor had a split at RE/TA $<= -1.846$ but this is close enough.

Main Optimisation 1

DT checks across all variables (features) for the most optimal split inside each,

$$k^* = \underset{k}{\text{argmin}}\ V_I^{(k)}\left(c^{k,*}\right)$$

It is useful to impose a condition of minimum gain expected for each split.

Main Optimisation 2

We are ready to write expression for total variation (homogeneity), $V_I^{(k)}$

$$V_I^{(k)}\left(c^{(k)}\right) = \underbrace{\sum_{x_i^{(k)} < c^{(k)}} \left(y_i - m_I^{k,-}(c^{(k)})\right)^2}_{\text{Total dispersion of first cluster}} + \underbrace{\sum_{x_i^{(k)} > c^{(k)}} \left(y_i - m_I^{k,+}(c^{(k)})\right)^2}_{\text{Total dispersion of second cluster}}$$

For feature k, the optimal split $c^{k,*}$ is one, for which the total variation over the two subgroups is the smallest.

Main Optimisation 3

$$
\begin{aligned}
m_I^{k,-}(c^{(k)}) &= \frac{1}{\#\{i, x_i^{(k)} < c^{(k)}\}} \sum_{\{x_i^{(k)} < c^{(k)}\}} y_i \\
&= \\
m_I^{k,+}(c^{(k)}) &= \frac{1}{\#\{i, x_i^{(k)} > c^{(k)}\}} \sum_{\{x_i^{(k)} > c^{(k)}\}} y_i
\end{aligned}
$$

$\sum y_i$ are the average values of Y, conditional on $X^{(k)}$ being smaller or larger than c.

$\#\{\cdot\}$ is a cardinal function, it counts the number of instances of its argument.

Main Optimization – notes

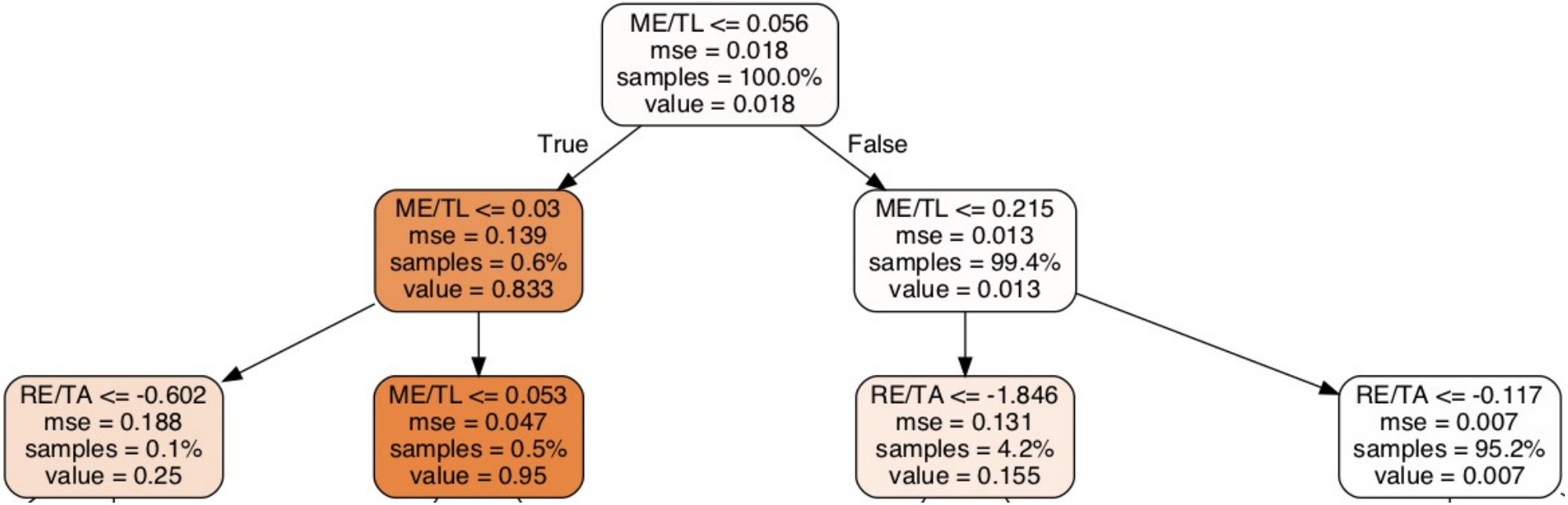
Corporate PD: an imbalanced dataset

The key aspect of our Corporate_PD.xlsm dataset is that among 4,000 annual observations (830 companies, 1999-2004), only > 41 defaulted within this five-year period.

Dependent variables (ratios) such as WC/TA are known as Altman Z-scores. The common model here is logistic regression (logit), however we will use decision trees.

	WC/TA	RE/TA	EBIT/TA	ME/TL	S/TA
0	0.500799	0.306846	0.043373	0.956271	0.334774
1	0.547780	0.322214	0.051843	1.064545	0.334591
2	0.451001	0.225150	0.026813	0.804096	0.245585
3	0.306887	0.191936	0.030058	0.387010	0.253438
4	0.447246	0.217368	0.032458	0.791639	0.275531

Decision Tree Regressor Output



- Market Cap/Total Liabilities ratio <= 0.056.

  Our first split immediately separates ‘small companies’. Our second split further separates micro caps below 0.03.

- Next we explain the meaning of “value” and “samples” outputs.

Regressor Output

- Regressor output gives **the average value of the class label**.

In our case that equals exactly the frequentist probability of default – because this is AVERAGE() among zeroes and ones {0,1}.

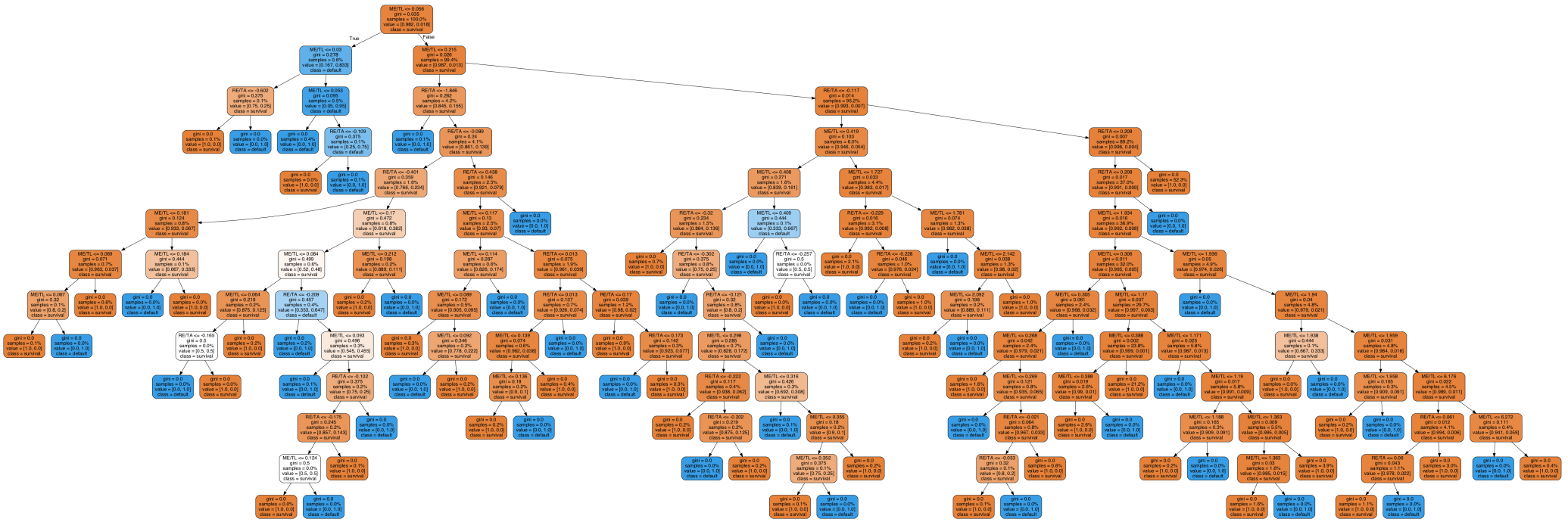
1.8% is exactly the proportion of defaults in the entire dataset.

Two orange leaves have 0.83 and 0.95 proportion of defaulted entities.

- Also gives **the number of samples (observations) in the leaf**, in our case as a percentage.

The top leaf contains 100% observations. Then the number is relative to the cluster before: 0.6% of observations are split into 0.5% ‘guaranteed’ defaults + 0.1% of entities with bad feature values but not defaulted.

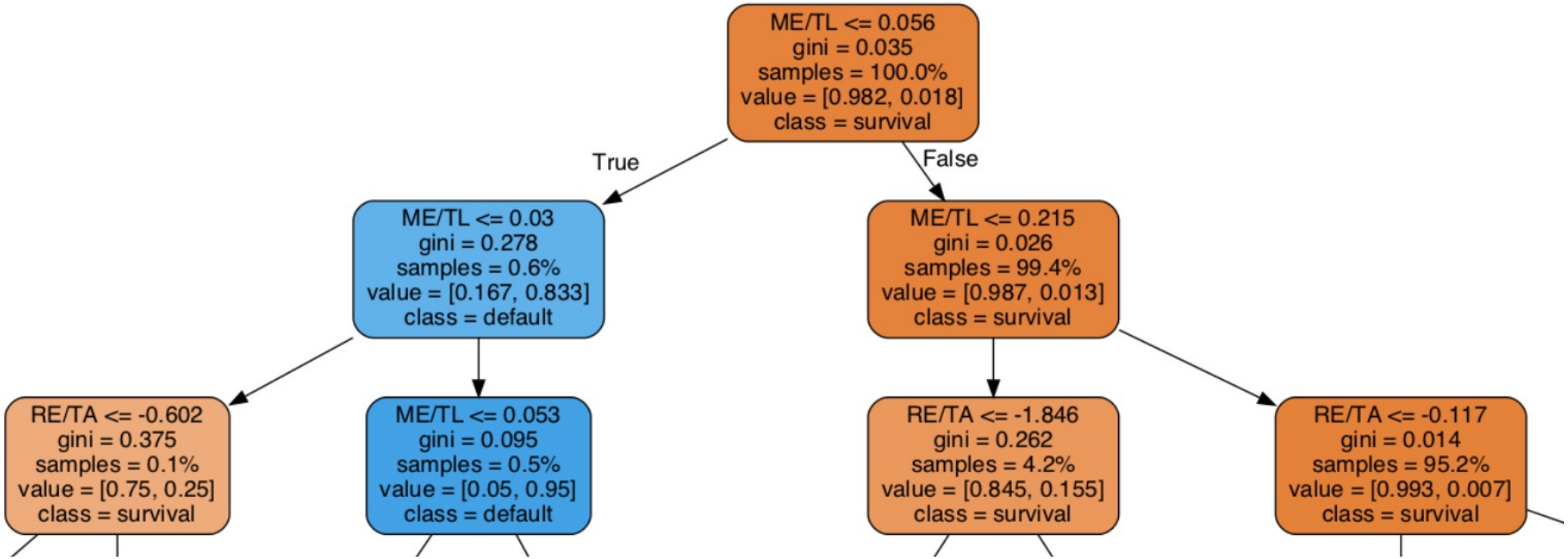
Decision Tree Classifier (no pruning)



WHOA! What is the problem with this Tree Classifier output?

Corrected by either running **Tree Regressor** or setting max_depth parameter.

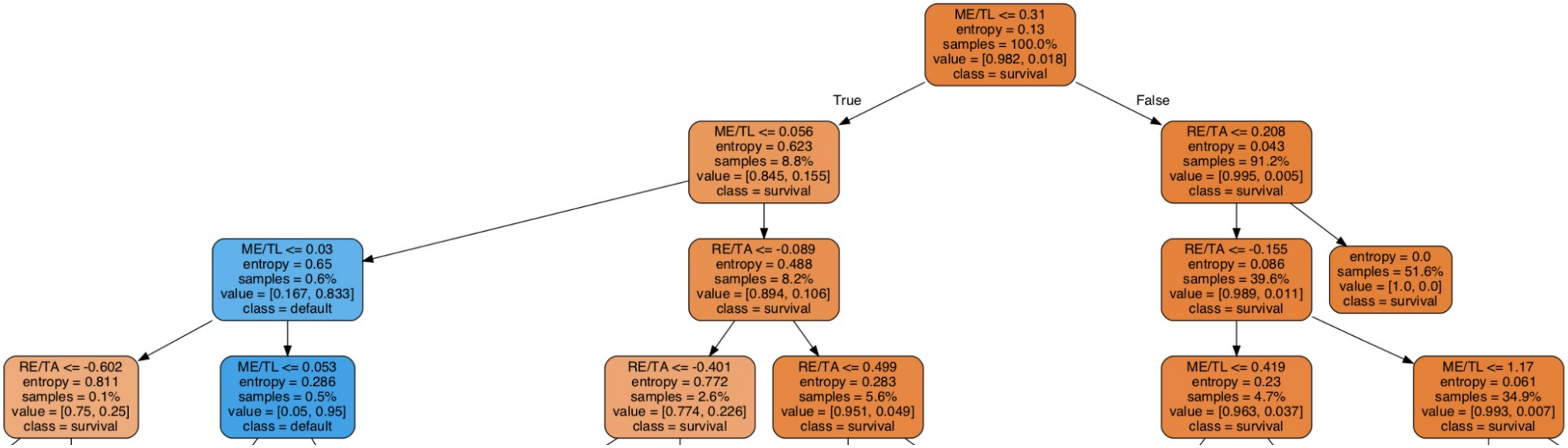
Decision Tree Classifier (Gini)



- Market Cap/Total Liabilities ratio <= 0.056.

  Same initial and consecutive spits – this looks almost like a mistake but the Gini impurity criterion is indicated for the classifier output.

Decision Tree Classifier (Entropy)



- Market Cap/Total Liabilities ratio <= 0.31.

  Some differentiation is made among survived companies (right side)

  Similar spits can be found for defaults (left side) as with Gini index :

  RE/TA <= -0.60, ME/TL <= 0.053 – visible in Python on the full tree.

Hyperparameters

- **Maximum Depth**

A decision tree can reach N_{obs}-1 depth. Whoa!

- **Min Number To Split** (default=2)

Minimum number (sample count) required before splitting an internal node

It is more computationally expensive, and **less intuitive** to change this.

- **Minimum Number In Leaf** (default=1)

By definition, a leaf is an external node. It cannot have any further splits.

A data scientist will run GridSearchCV to find an optimal value, **but** a smart analyst will impose a value (appropriate to the dataset) such, that small clusters in leaf not likely to be flukes.

Regressor vs. Classifier

Below are optimal results for hyperparameter tuning for tree regressor and classifier.

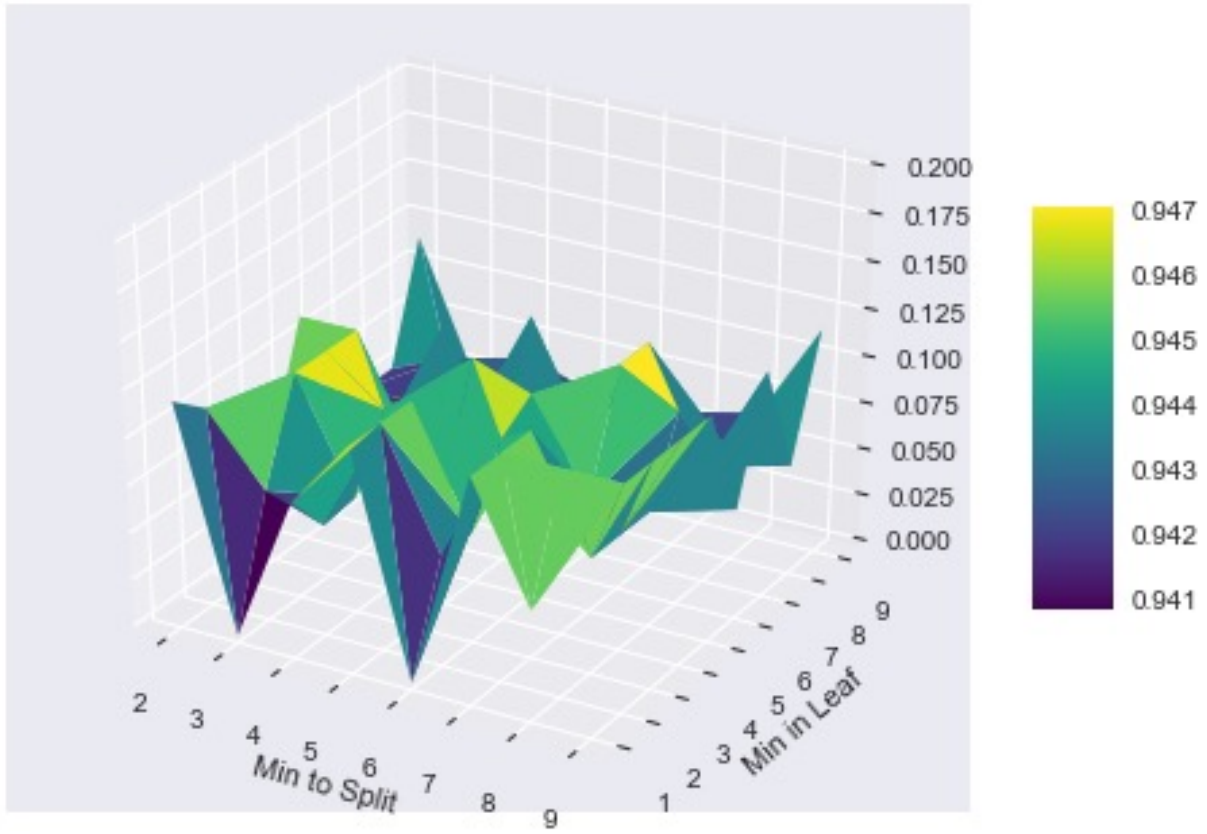
Without looking ahead, can you tell which one is which?

	MinNum2Split	MinNumInLeaf	MaxDepth	TrainAccuracy	TestAccuracy
629	9.0	7.0	9.0	0.260401	0.133929

	MinNum2Split	MinNumInLeaf	MaxDepth	TrainAccuracy	TestAccuracy
21	2.0	3.0	4.0	0.99375	0.95

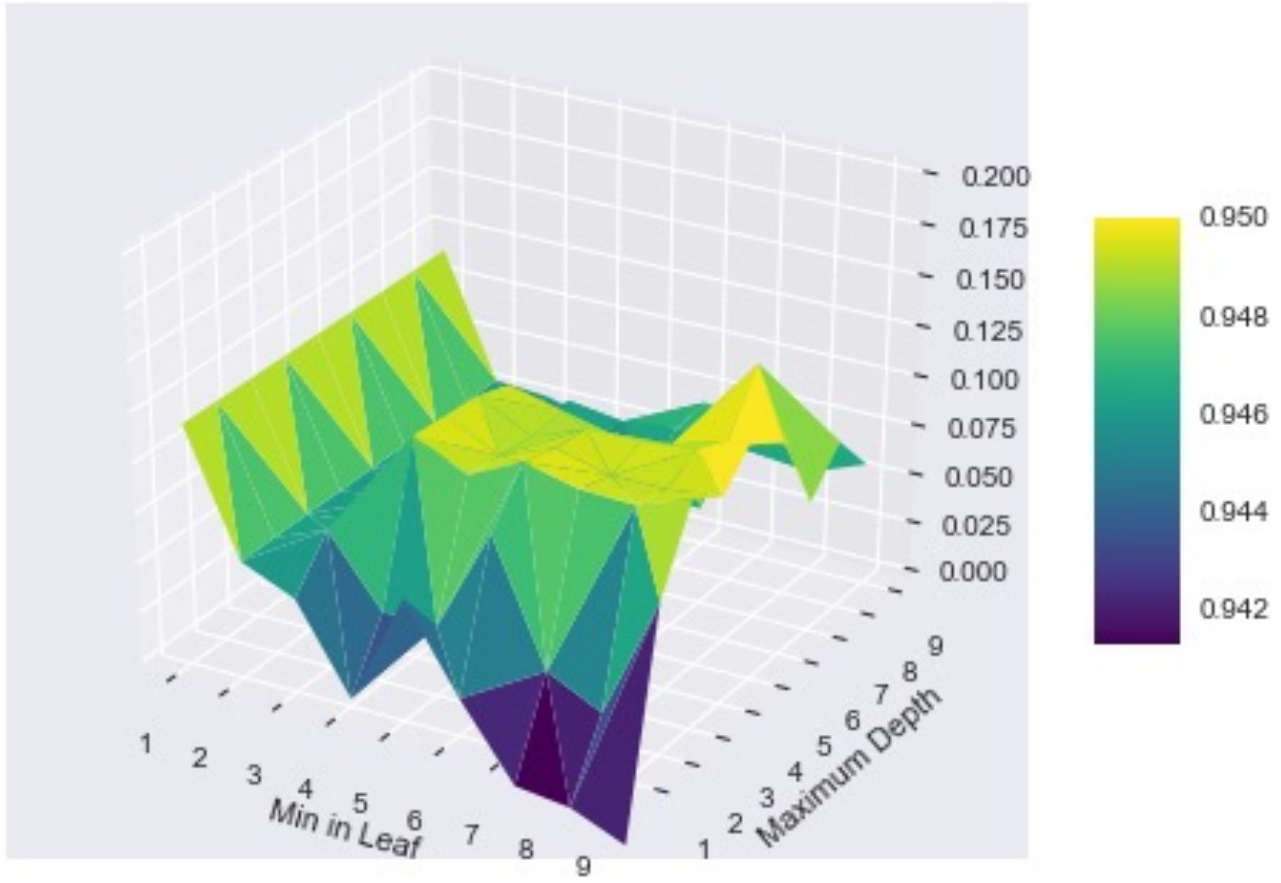
Min To Split > Minimum In Leaf healthy tuning condition

Tuning DT Regressor – 1



Here for a regressor, MinNum2Split, MinNumInLeaf are interlocked and deliver similar accuracy as long as we don't go into extremely low splits of leaves with 1-2 observations.

Tuning DT Regressor – 2

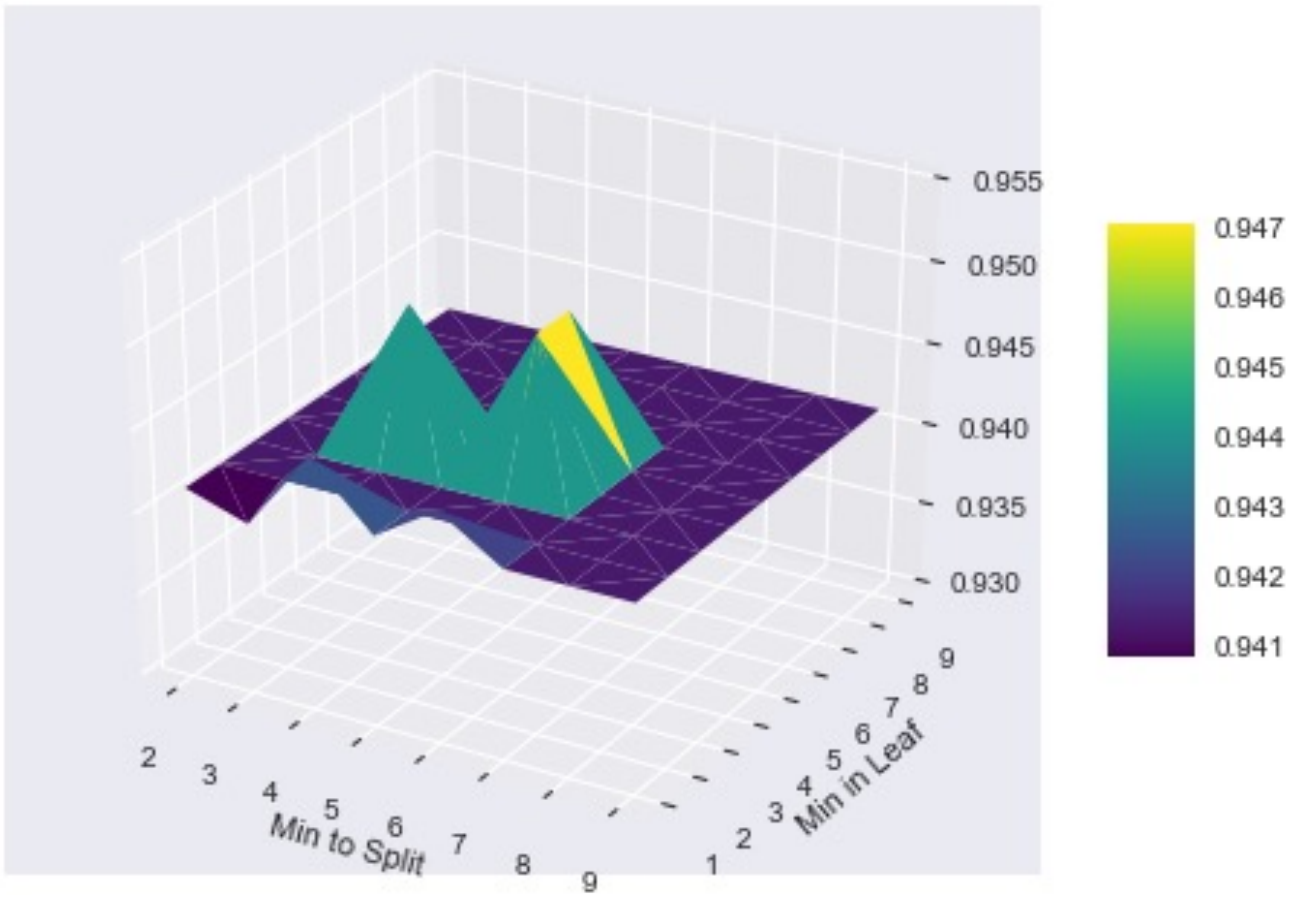


MinNumInLeaf vs MaxDepth is a surface with more characteristic.

Ignoring the extreme of 1 observation in leaf, the optimal value for MinNumInLeaf is 7-8.

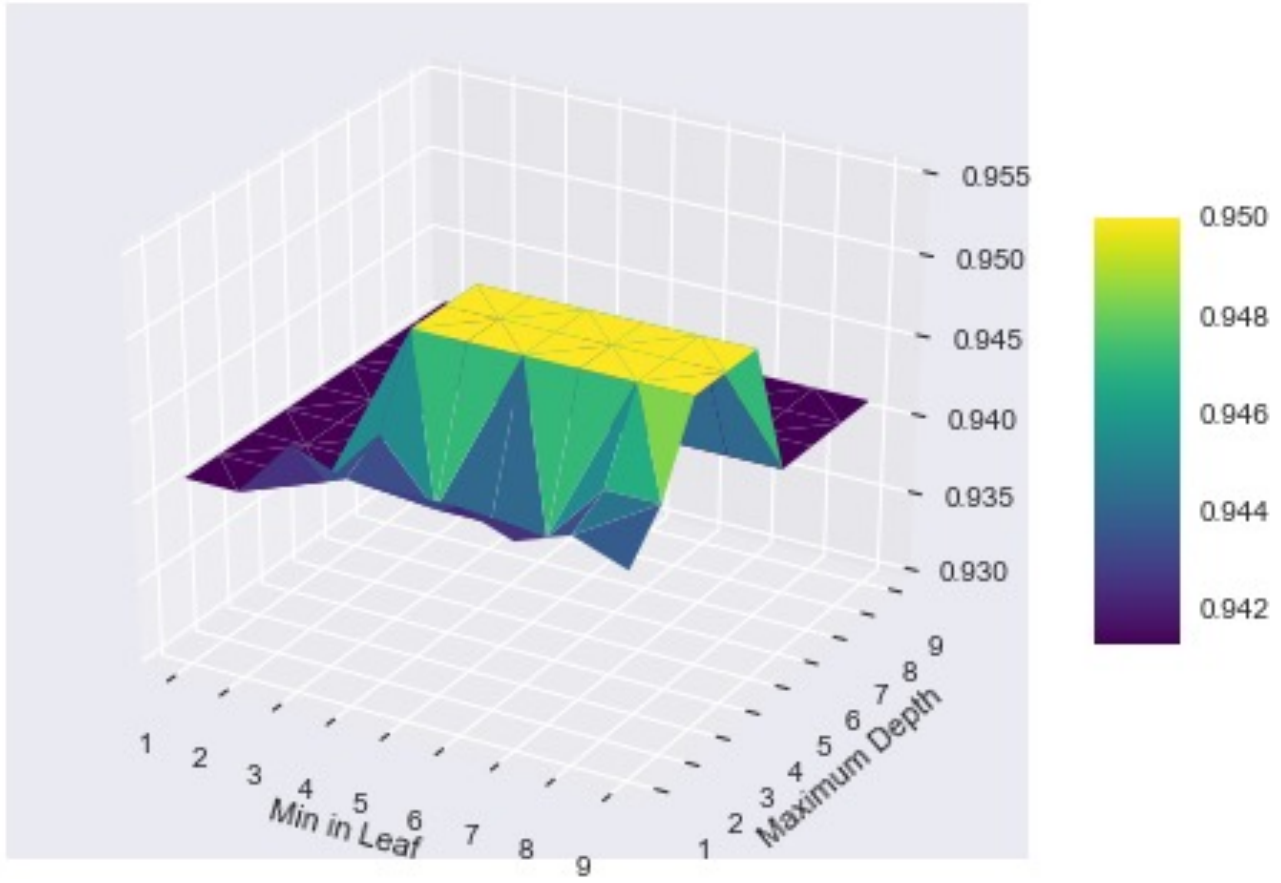
Tuning suggests to have deeper and richer trees.

Tuning DT Classifier – 1



Decision tree classifier ‘suffers’ from high accuracy and tuning does only adds a modest percentage of accuracy, e.g. from 94 to 95%.

Tuning DT Classifier – 2



At least, the tree classifier is not greedy in asking high MaxDepth: the parameter was varied from 1 to 10, the accuracy performance flattens for the tree of 4 levels. Likely specific to our Corporate PD dataset, even the tree of 1 level will separate all survivals effectively!

Hyperparameters declaration

Split criterion is *mean squared error*.

We have considered the three most important hyperparameters on the prior slide.

- **Max leaf nodes** can tinker with this and re-organise the tree structure but max_depth

```
DecisionTreeRegressor (
    criterion: str=str,
    splitter: str=str,
    max_depth: __class__=None,
    min_samples_split: int=2,
    min_samples_leaf: int=1,
    min_weight_fraction_leaf: float=0,
    max_features: __class__=None,
    random_state: __class__=None,
    max_leaf_nodes: __class__=None,
    min_impurity_decrease: float=0,
    min_impurity_split: __class__=None,
    presort: bool=False
)
```

Impurity refers to the default use of Gini index as Loss Function

CQF FitchLearning

CORE LECTURE

Onto Ensemble Methods – Bagging a Logistic Regression

Dr Richard Diamond

Ensemble Methods

A group of predictors form an **ensemble**. A predictor is a specifically fitted model, eg one fitted regression with specific *coefficients*, one fitted decision trees with known *splits*, or even one NN with fitted *weights and biases.*

- **Bagging** is a parallel ensemble. In quant finance we will look at the use of bagging to improve such tool as a logistic regression.

- **Boosting** is a sequential ensemble.

BAGGING ALGORITHM – 1,2

1. Sample multiple subsets from the training data.

This suddenly enlarges our dataset, which can be quite modest in financial applications (eg, from 2,000 to 10,000 observations).

2. Predictors estimated in parallel (independently).

Each predictor has *a likelihood* to accentuate different features.

Weak ‘negative feedback’ relations can be captured with stronger coefficients – more statistically significant by t-test (p-value).

`BaggingRegressor()` by default fits Decision Tree as an alternative model.

BAGGING ALGORITHM – 3

3. The predictions are aggregated to give a better estimate of the true value.

Aggregation is done by a weighted average. But a more involved scheme, **Soft Voting** is implemented in `VotingClassifier()`

classifier	class 1	class 2	class 3
classifier 1	w1 * 0.2	w1 * 0.5	w1 * 0.3
classifier 2	w2 * 0.6	w2 * 0.3	w2 * 0.1
classifier 3	w3 * 0.3	w3 * 0.4	w3 * 0.3
weighted average	0.37	0.4	0.23

3-class classification soft voting with equal weights w1=w2=w3. The predicted class probabilities presented in table.

Bagging sklearn code and tuning

Bagging Algorithm is a convenient **pipeline**.

- Number of subsamples

subsample size = N/n_samples

max_samples = 0.5 means to use 50% of observations for each predictor

- Number of predictors and type of predictor (logistic regressions, decision trees)

base_estimator, n_estimators

- Aggregation methods for ‘an average’ prediction

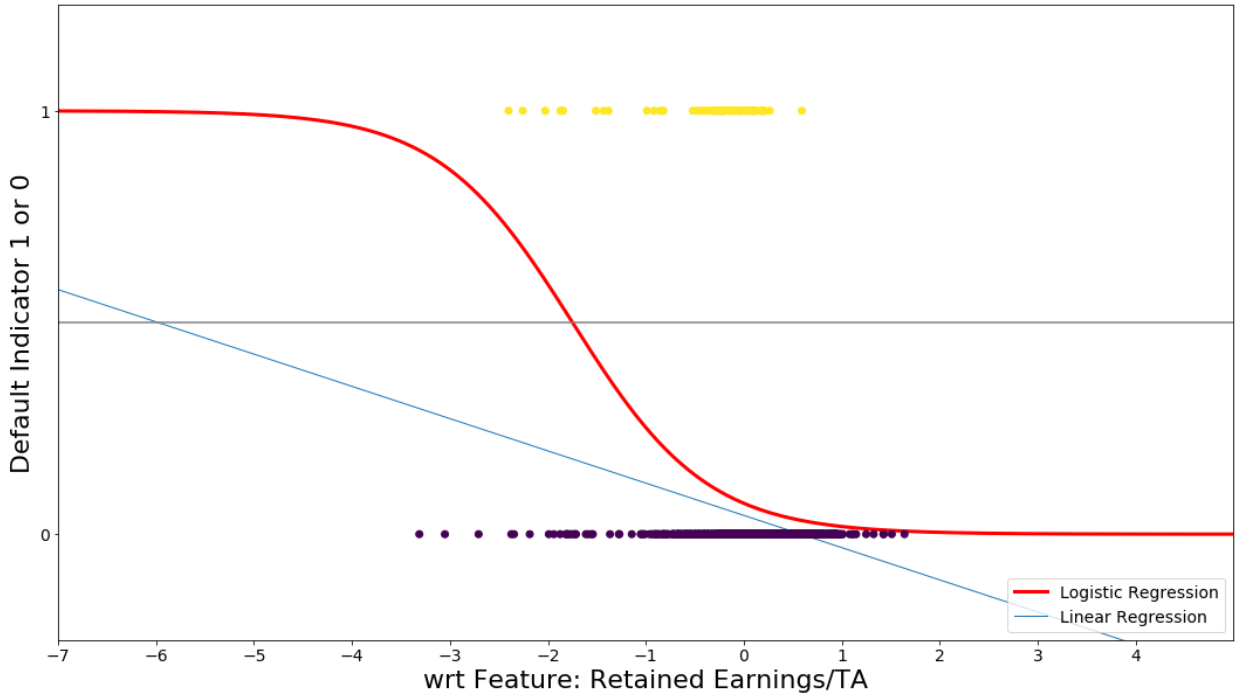
```
BaggingRegressor (
    base_estimator: __class__=N
    n_estimators: int=10,
    max_samples: float=1,
    max_features: float=1,
    bootstrap: bool=True,
    bootstrap_features: bool=Fals
    oob_score: bool=False,
    warm_start: bool=False,
    n_jobs: __class__=None,
    random_state: __class__=No
    verbose: int=0
)
```

Ensemble Learning can be used together with **any classifier or deep learning (neural net) model.**

Logistic Sigmoid

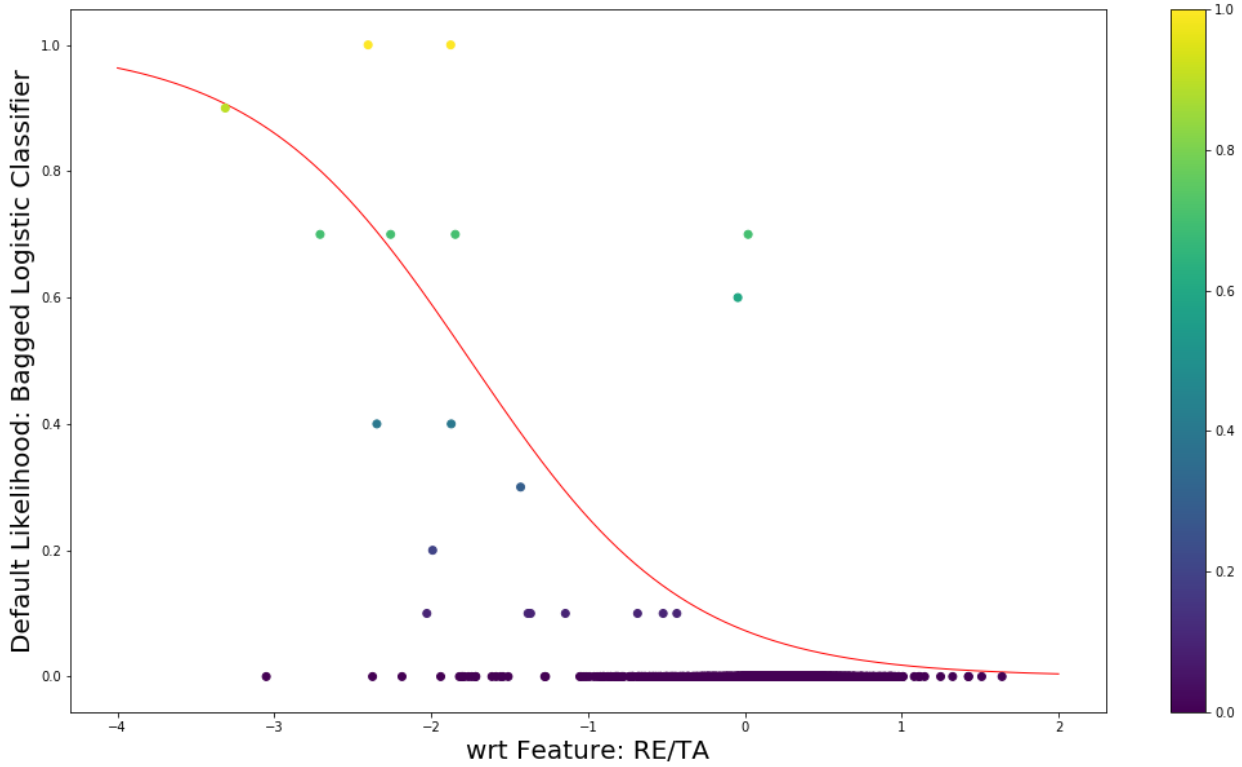
```
def logistic_sigmoid(xb):
    return (1 / (1 + np.exp(-xb)))
```

$$p(X) = \frac{1}{1 + e^{-\boldsymbol{X}\beta'}}$$



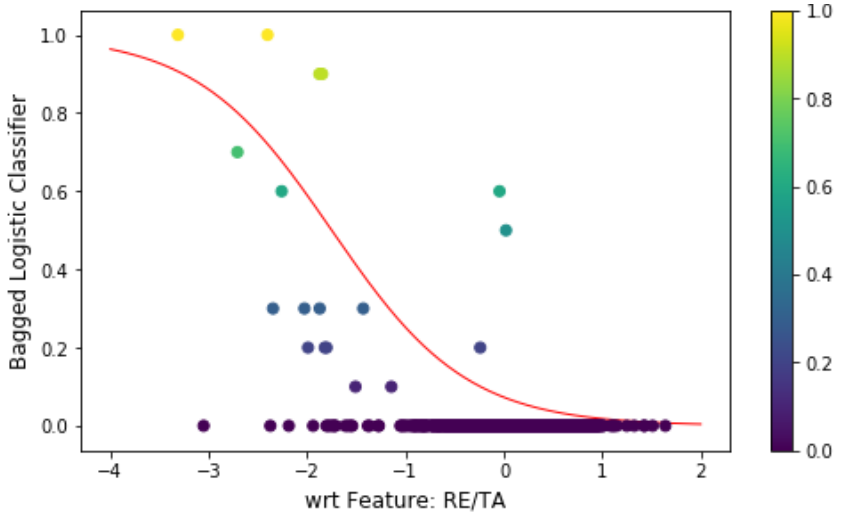
What does this tell us in terms of classification?

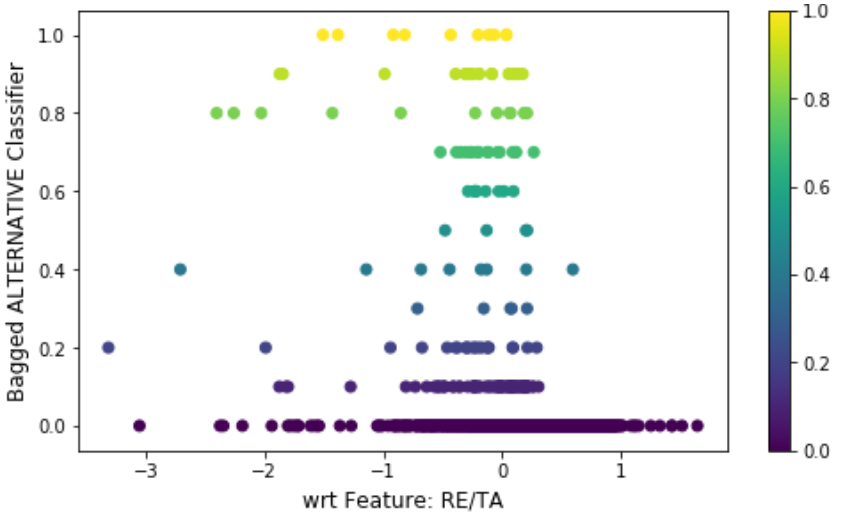
now predictor is Bagging Regressor

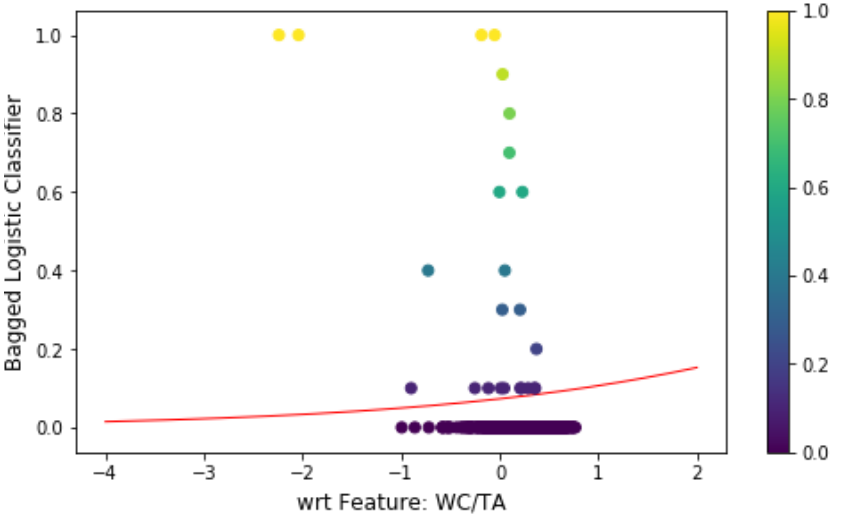


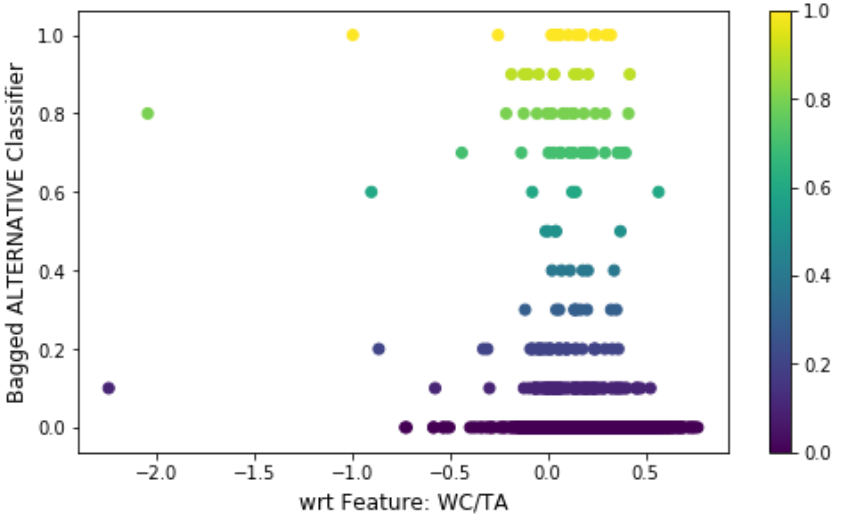
Results will differ on each run of bagged classifier – this is a **pipeline** of Ensemble Method learning on top of supervised classifiers.

Bagging Regressor (super-predictor)









our empirical use of Bagging on Corporate PD

The observations count on which Bagging Regressor operates is obviously above $> 4,000$ (Ncompanies in our dataset) -- notice the density of dots.

n_estimators = 10 by default. That means 10 alternative Decision Tree models were produced, each will have different companies predicted as defaulted (REF (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html)).

- Those 10 trees were more different than necessary, but each is accurate in its unique way. Combining their predictions — results in a better estimate of the true value.
- A clever bit: trees are created so that rather than selecting optimal split points, suboptimal splits made randomly. Done to improve out-of-sample predictions.
  Otherwise, given large enough tree it will fit the sample excessively perfect.
- **TAKE AWAY** If you get good classifying or clastring results with an algorithm that has inherently high variance (eg Decision Tree), you can often get better results by bagging that algorithm.

Origins of bagging (bootstrapping in statistics)

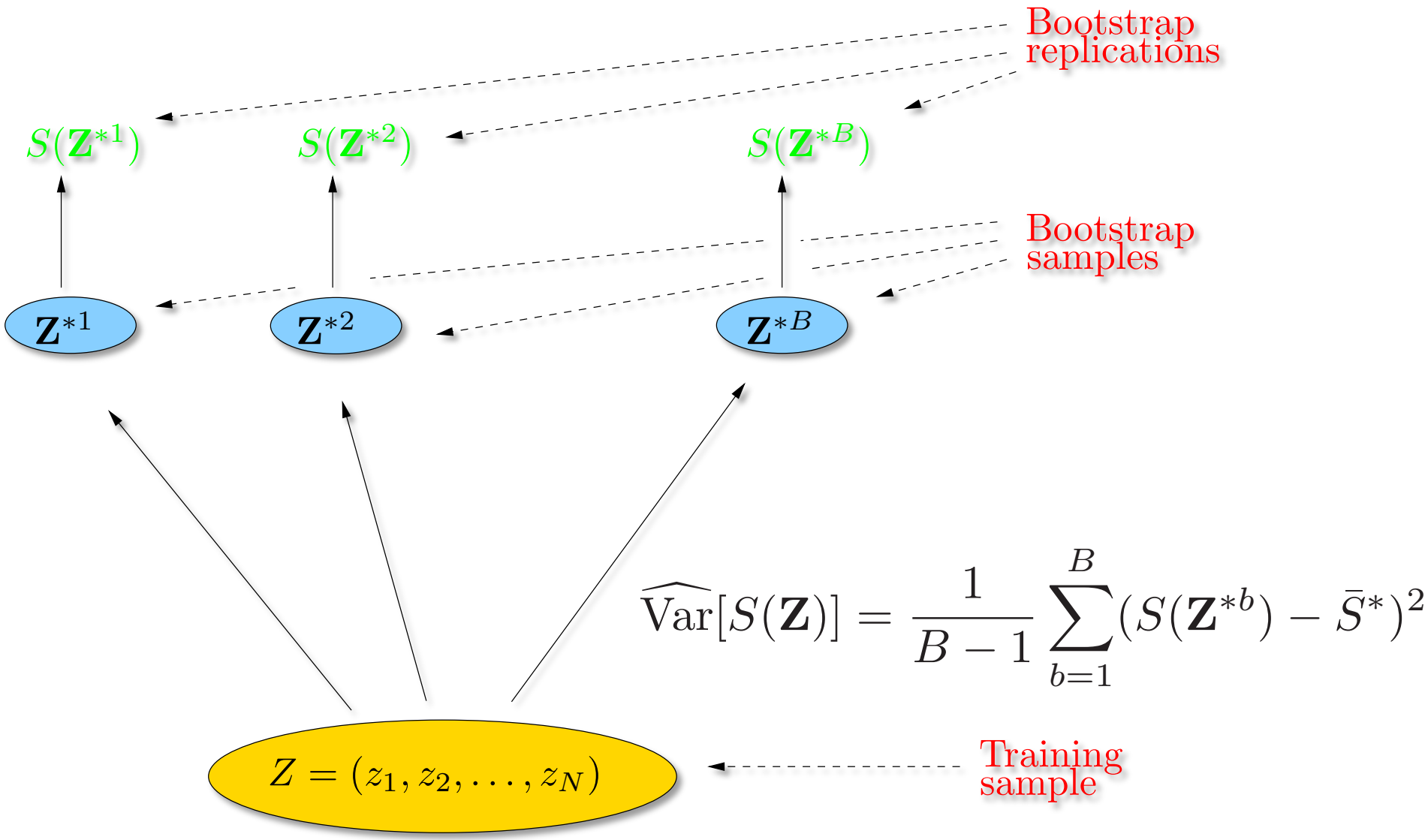
Formally, bagging relies on the following relationship:

- The bootstrapped sample mean quantity is approximately equal to a posterior average – as if a Bayesian formula were utilised to update the estimate of mean. This is painless, without having to formally specify the prior and posterior distributions.

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

- Each predictor has a likelihood to accentuate different features, which will be more relevant to a particular sub-sample.
- Bagging averages the prediction over a collection of bootstrap samples (aka bootstrap aggregation), thereby reducing its variance.

EXTRA Bootstrap Scheme



1) Overlapping of samples. “0.632 Estimator”

Impact of repetitive observation in N samples

$$\begin{aligned}\Pr\{\text{observation } i \in \text{bootstrap sample } b\} &= 1 - \left(1 - \frac{1}{N}\right)^N \\ &\approx 1 - e^{-1}\end{aligned}$$

the average number of distinct observations in each bootstrap is 0.632 * N.

$$\widehat{\text{Err}}^{(.632)} = .368 \cdot \overline{\text{err}} + .632 \cdot \widehat{\text{Err}}^{(1)}$$

The idea of prediction error being computed from the leave-one-out bootstraps Err^(1) -- samples not containing repetition(s) of the observation.

Crossvalidation (CV) / Bagging at best work with the expected prediction error that is still an in-sample error (Err_{in}). Hence, 0.368 of training error err added to 0.632 of in-sample error in order to obtain an extra-sample error estimate.

2) In-sample error

$$\mathrm{E}_{\mathbf{y}}(\mathrm{Err}_{\mathrm{in}}) = \mathrm{E}_{\mathbf{y}}(\overline{\mathrm{err}}) + 2 \cdot \frac{d}{N}\sigma_{\varepsilon}^2$$

$$\mathrm{Err}_{\mathrm{in}} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{E}_{Y^0}(Y_i^0 - \hat{f}(x_i))^2$$

$$\overline{\mathrm{err}} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{f}(x_i))^2$$

- err is training error from a specific sample, which is typically below a fair in-sample error $\mathrm{Err}_{\mathrm{in}}$, let alone generalization error of a model.
- in-sample error **allows for** new responses Y_0 at the training points x_i (x_1, x_2, …).

Generalisation Error can be thought as *an extra-sample* error, as testing inputs do not need to coincide with training inputs. Think of generating function values $f(x_i)$ for a random but statistically sufficient set x_i.

2) Sources of out-of-sample error for Corporate PD data

While it might not be possible to quantity the error, it is useful to apply the idea of coinciding inputs (or not coinciding).

Corporate PD data **would not** be coinciding for:

- companies defaulted, new companies survived at the same historic times;
- the same companies but evaluated over the next 5-year period.

CV/bootstrap are meant to be direct estimates of the extra-sample error Err – but unlikely to be successful if training samples are homogenous. Also, regime changes are common to financial data: you are not learning to recognise the same target...

3) Number of features

The right side of our in-sample error formula is **Akaike / Bayesian criteria** for the number of params/features/variables **d**.

For the logistic regression model,

$$\mathrm{AIC} = -\frac{2}{N} \cdot \mathrm{loglik} + 2 \cdot \frac{d}{N}.$$

- We choose the model giving the smallest AIC for the number of features *d.*
- The other popular use of AIC/BIC criteria is: deciding the number of past (lagged) returns to be included into a vector autoregression.
  - r_{t-1}, r_{t-2}, r_{t-3} etc

CQF FitchLearning

CORE LECTURE

Decision Trees Boosting: from AdaBoost scheme to XGB

Dr Richard Diamond

BOOSTED Decision Trees

The model (a pipeline of predictors) is built sequentially as each additional tree aims to correct / address the inaccuracy of the previous.

- **Gradient Boosting** relies on an *additive strategy*, which fixes (records) what was learned as a state variable h(x).

- **XGBoost** each tree focuses on minimisation of loss in the training sample (eg, sum of differences y – ytrue). The method is flexible and gets you to win a Kaggle competition. Great computational scalability.

- **AdaBoost** focuses *only* on observations that give the largest inaccuracies (loss, squared error). Improving outlier prediction…

BOOSTED Decision Trees notes

In context of boosting, separate decision tree regressors are referred to as *'weak learners'.*

Consensus of independent weak learners: random forests can improve on bagging if the correlation between the sampled trees is reduced.

BOOSTING ALGORITHM

1. A learner l_m minimises weighted Loss across all observation-prediction pairs $(l_m(x_i), y_i) = (ypred_i, y_i)$

For a regressor, the loss function is Mean Squared Error.

2. Incorrect predictions are `assigned' to the next boosted learner l_{m+1} via a changed weight.

Correct predictions will have zero error which means zero weight

$$w_i \leftarrow w_i e^{f_w(l_m(\mathbf{x}_i),\mathbf{y}_i)}$$

Weights are normalised.

3. Step 2 is repeated until a correct prediction

AdaBoost: maths summary

We consider our Python Notebook, *DT Stock Selection - T Guida.ipynb*, please refer to Section 3 Boosted Trees: AdaBoost.

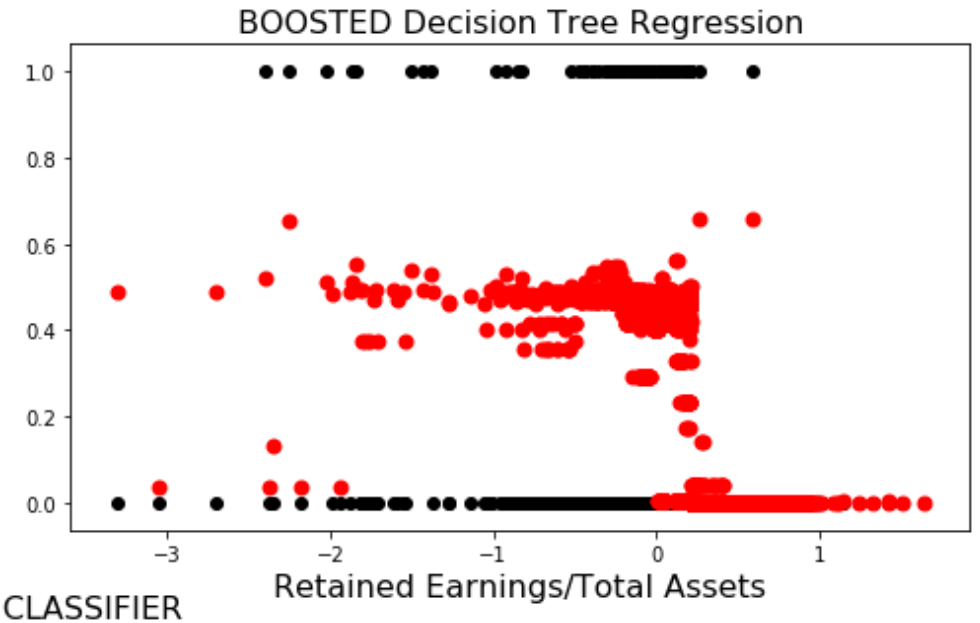
The core of AdaBoost is Step 3 updating the observations weights to incentivize the next learner (decision tree) to address them.

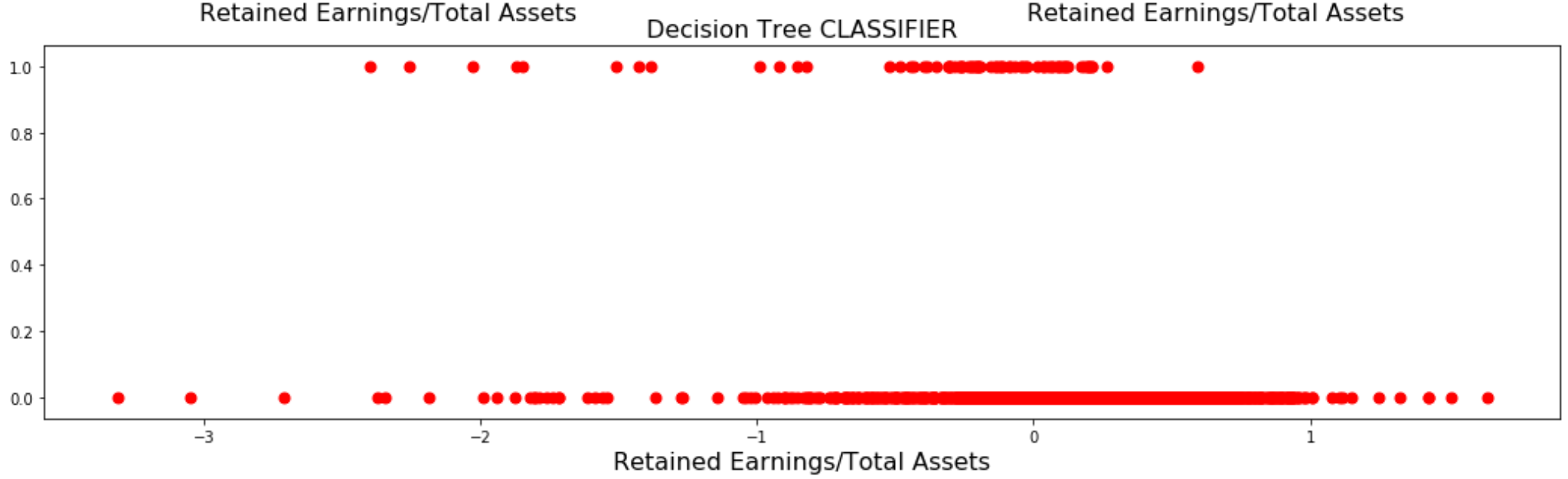
	Bin. classif. (orig. Adaboost)	Regression (Drucker (1997))
Individual error	$\epsilon_i = \mathbf{1}_{\{y_i \neq l_m(\mathbf{x}_i)\}}$	$\epsilon_i = \frac{\lvert y_i - l_m(\mathbf{x}_i)\rvert}{\max_i \lvert y_i - l_m(\mathbf{x}_i)\rvert}$
Weight of learner via f_a	$f_a = \log\left(\frac{1-\epsilon}{\epsilon}\right)$,with $\epsilon = I^{-1}\sum_{i=1}^{I} w_i \epsilon_i$	$f_a = \log\left(\frac{1-\epsilon}{\epsilon}\right)$,with $\epsilon = I^{-1}\sum_{i=1}^{I} w_i \epsilon_i$
Weight of instances via $f_w(i)$	$f_w = f_a \epsilon_i$	$f_w = f_a \epsilon_i$
Output function via f_y	$f_y(x) = \text{sign}(x)$	weighted median of predictions

AdaBoost on our Corporate PD data

Our case: three datasets on one plot (actual, DT Regressor, AdaBoost).

Output pattern tells us about the method





Breiman (2001) result

Generalisation Error for an ensemble of learners (such as Random Forest) is bounded by the inaccuracy of aggregation:

$$\bar{\rho} \ \frac{1 - s^2}{s^2}$$

s is the strength (average quality) of the individual classifiers, and

$\bar{\rho}$ is the average correlation between the learners.

Implication is that random forests are not prone to overfitting, but we saw in both, AdaBoost over regressor/Boosted tree classifier – the predictive utility of random forest is very average.

Conclusions on Decision Trees/Forests

- Overfitting. One does not need a super-large or boosted Random forest to achieve 100% accuracy in-sample.

  Trees are adaptable to the sample, not many branches are required to zero-in on MinNumInLeaf of 1-2 observations.

- Complexity of each individual tree is a decreasing function of hyperparameters, eg MinNum2Split. Discussion of pruning criteria overlaps with hyperparameter tuning, which we considered.

  See Section 1.3 in *DT Theory and Stock Selection – T Guida.ipynb*

- While its possible to extract individual trees, the visualisation and synthesis across Random Forest is out of reach. xGBoost case studies up with features importance (see Python Lab) and that is it.

Useful Links to Cases

Decision Trees

- Framework of Ensemble Methods
  https://www.mathworks.com/help/stats/ensemble-algorithms.html

  of particular interest is Adaptive Boosting for Binary Classification `AdaBoostM1`

- Ensemble Methods in sklearn

  https://scikit-learn.org/stable/modules/ensemble.html

- Tree-based methods, Chapter 7 of *Machine Learning for Factor Investing (Coqueret & Guida 2020)*

CQF FitchLearning

THEORY EXTRA SLIDES

Bias and Error in Model Space

Dr Richard Diamond

May 2022

Model Error and Bias

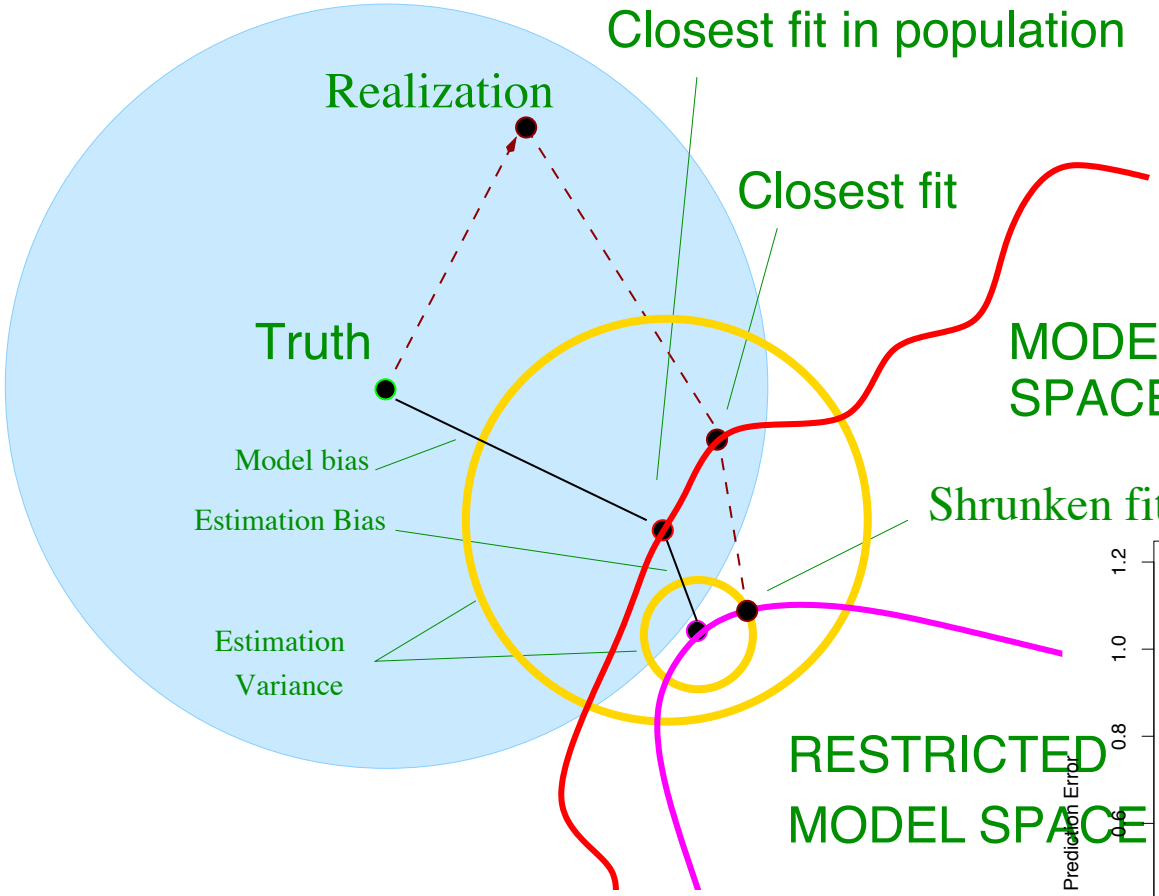
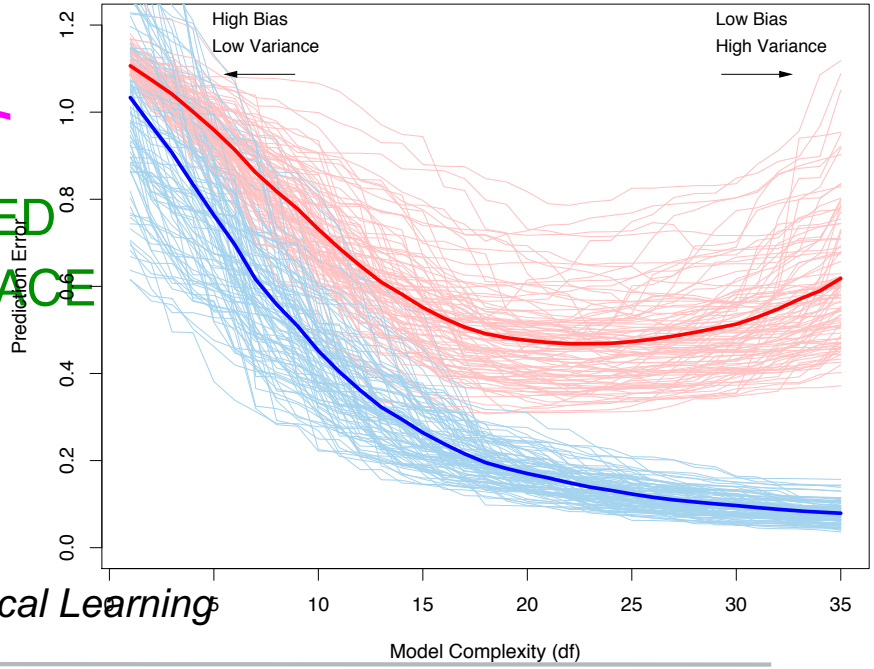


FIGURE 7.2 page 225

FIGURE 7.1 page 220



From: Hastie, Tibshirani, Friedman, The Elements of Statistical Learning

Bias-Variance Decomposition and Bagging

Assume our training observations $(x_i, y_i),\ i = 1, \ldots, N$ are independently drawn from a distribution $\mathcal{P}$, and consider the ideal aggregate estimator $f_{\text{ag}}(x) = \text{E}_{\mathcal{P}} \hat{f}^*(x)$. Here x is fixed and the bootstrap dataset $\mathbf{Z}^*$

$$\begin{aligned} \text{E}_{\mathcal{P}}[Y - \hat{f}^*(x)]^2 &= \text{E}_{\mathcal{P}}[Y - f_{\text{ag}}(x) + f_{\text{ag}}(x) - \hat{f}^*(x)]^2 \\ &= \text{E}_{\mathcal{P}}[Y - f_{\text{ag}}(x)]^2 + \text{E}_{\mathcal{P}}[\hat{f}^*(x) - f_{\text{ag}}(x)]^2 \\ &\geq \text{E}_{\mathcal{P}}[Y - f_{\text{ag}}(x)]^2. \end{aligned} \tag{8.52}$$

$f_{ag}(x)$ estimator is computed from the actual population ***P*** rather than data. Last line suggests that true population aggregation (bagging) would not increase the mean squared error (it is ideal estimator's error plus extra).

However, for binary/multinomial classification bagging a good classifier can make it better, while bagging a bad classifier can make it perform worse.

From: Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, section 8.7. p 285-286

Textbook original text (*Hastie et al.*)

Assume our training observations (x_i, y_i), $i = 1, \ldots, N$ are independently drawn from a distribution $\mathcal{P}$, and consider the ideal aggregate estimator $f_{\text{ag}}(x) = \text{E}_{\mathcal{P}} \hat{f}^*(x)$. Here x is fixed and the bootstrap dataset $\mathbf{Z}^*$ consists of observations x_i^*, y_i^*, $i = 1, 2, \ldots, N$ sampled from $\mathcal{P}$. Note that $f_{\text{ag}}(x)$ is a bagging estimate, drawing bootstrap samples from the actual population $\mathcal{P}$ rather than the data. It is not an estimate that we can use in practice, but is convenient for analysis. We can write

$$\begin{aligned} \text{E}_{\mathcal{P}}[Y - \hat{f}^*(x)]^2 &= \text{E}_{\mathcal{P}}[Y - f_{\text{ag}}(x) + f_{\text{ag}}(x) - \hat{f}^*(x)]^2 \\ &= \text{E}_{\mathcal{P}}[Y - f_{\text{ag}}(x)]^2 + \text{E}_{\mathcal{P}}[\hat{f}^*(x) - f_{\text{ag}}(x)]^2 \\ &\geq \text{E}_{\mathcal{P}}[Y - f_{\text{ag}}(x)]^2. \end{aligned} \tag{8.52}$$

The extra error on the right-hand side comes from the variance of $\hat{f}^*(x)$ around its mean $f_{\text{ag}}(x)$. Therefore true population aggregation never increases mean squared error. This suggests that bagging—drawing samples from the training data— will often decrease mean-squared error.

Bias-Variance Decomposition in detail

The *true model* is $Y = f(X) + \epsilon$ with $E(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma_\epsilon^2$.

$\text{Err}(x_0)$ as the expected test error (or generalization error)

$$\begin{aligned}
\text{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] \\
&= E[(f(x_0) + \epsilon - \hat{f}(x_0))^2] \\
&= E[(f(x_0) - \hat{f}(x_0))^2 + 2\epsilon(f(x_0) - \hat{f}(x_0)) + \epsilon^2] \\
&= E[(f(x_0) - \hat{f}(x_0))^2] + 2E[\epsilon(f(x_0) - \hat{f}(x_0))] + E[\epsilon^2]
\end{aligned}$$

$$\text{Err}(x_0) = E[(f(x_0) - \hat{f}(x_0))^2] + \sigma_\epsilon^2$$

$$E[(f(x_0) - \hat{f}(x_0))^2] = E[(f(x_0) - E\hat{f}(x_0) + E\hat{f}(x_0) - \hat{f}(x_0))^2]$$

$$\begin{aligned}
\text{Err}(x_0) &= (f(x_0) - E\hat{f}(x_0))^2 + E[(E\hat{f}(x_0) - \hat{f}(x_0))^2] + \sigma_\epsilon^2 \\
&= \text{Model Bias}^2 + \text{Model Variance} + \sigma_\epsilon^2 \,.
\end{aligned}$$

Practical implicaitons

A very simple explanation about bias:

It means a model started away from the true initial position x_0, $f(x_0)$.

E.g., we never know what our true initial estimates to normalise the data to them (like we do in cointegration analysis by normalising all series with division by the first observed price P_t / P_0).

The similar problem is that we don't know the true mean of the evolving time series data (and that is after assuming stationarity in the data!).

Consequence 1. Can't de-trend the data and properly compute std dev and correlations! (x- xbar)^2 present in all formuales.

Theory Slide 1. Extra-sample error definition

common and convenient is *squared error loss*: $L(Y, f(X)) = (Y - f(X))^2$.

$$\begin{aligned} \mathrm{EPE}(f) &= \mathrm{E}(Y - f(X))^2 && (2.9) \\ &= \int [y - f(x)]^2 \Pr(dx, dy), && (2.10) \end{aligned}$$

Figure 2.7 illustrates the setup. We have broken down the MSE into two components that will become familiar as we proceed: variance and squared bias. Such a decomposition is always possible and often useful, and is known as the *bias–variance decomposition*. Unless the nearest neighbor is at 0,

$$\begin{aligned} \mathrm{MSE}(x_0) &= \mathrm{E}_{\mathcal{T}}[f(x_0) - \hat{y}_0]^2 \\ &= \mathrm{E}_{\mathcal{T}}[\hat{y}_0 - \mathrm{E}_{\mathcal{T}}(\hat{y}_0)]^2 + [\mathrm{E}_{\mathcal{T}}(\hat{y}_0) - f(x_0)]^2 \\ &= \mathrm{Var}_{\mathcal{T}}(\hat{y}_0) + \mathrm{Bias}^2(\hat{y}_0). && (2.25) \end{aligned}$$

estimate of test error is the *training error* $\frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$. Unfortunately training error is not a good estimate of test error, as it does not properly account for model complexity.

true prediction error $\mathrm{Err}_{\mathcal{T}}(\hat{\alpha})$

True prediction error can be thought of as extra-sample error

Section 7.4 Training err < generalization error

tion 7.2. Given a training set $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)\}$ the generalization error of a model $\hat{f}$ is

$$\text{Err}_{\mathcal{T}} = \text{E}_{X^0, Y^0}[L(Y^0, \hat{f}(X^0))|\mathcal{T}]; \tag{7.15}$$

Averaging over training sets $\mathcal{T}$ yields the expected error

$$\text{Err} = \text{E}_{\mathcal{T}}\text{E}_{X^0, Y^0}[L(Y^0, \hat{f}(X^0))|\mathcal{T}], \tag{7.16}$$

which is more amenable to statistical analysis. As mentioned earlier, it turns out that most methods effectively estimate the expected error rather than $\text{E}_{\mathcal{T}}$; see Section 7.12 for more on this point.

Now typically, the training error

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{7.17}$$

will be less than the true error $\text{Err}_{\mathcal{T}}$, because the same data is being used to fit the method and assess its error (see Exercise 2.9). A fitting method

Optimism

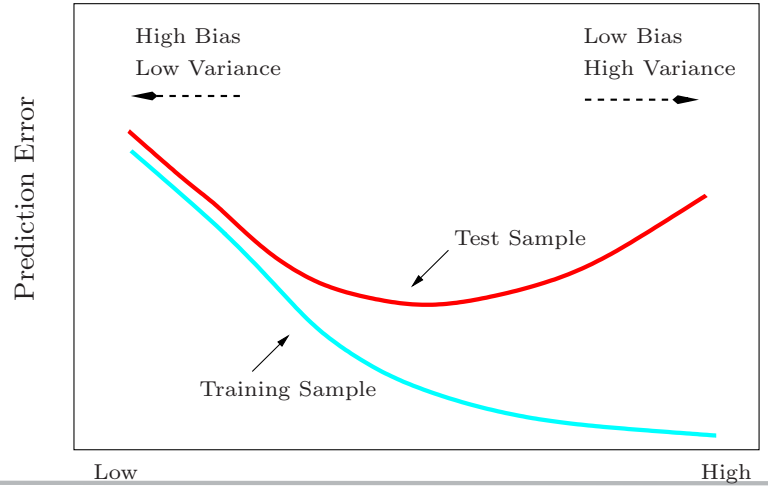
$$\text{op} \equiv \text{Err}_{\text{in}} - \overline{\text{err}}. \tag{7.19}$$

$$\omega \equiv \text{E}_{\mathbf{y}}(\text{op}). \qquad \omega = \frac{2}{N}\sum_{i=1}^{N}\text{Cov}(\hat{y}_i, y_i), \qquad \text{E}_{\mathbf{y}}(\text{Err}_{\text{in}}) = \text{E}_{\mathbf{y}}(\overline{\text{err}}) + \frac{2}{N}\sum_{i=1}^{N}\text{Cov}(\hat{y}_i, y_i). \tag{7.22}$$

Theory Slide 2. Impact of Bias/Variance

The k-nearest-neighbor regression fit $\hat{f}_k(x_0)$ usefully illustrates the competing forces that affect the predictive ability of such approximations. Suppose the data arise from a model $Y = f(X) + \varepsilon$, with $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma^2$. For simplicity here we assume that the values of x_i in the sample are fixed in advance (nonrandom). The expected prediction error at x_0, also known as *test* or *generalization* error, can be decomposed:

$$\begin{aligned} \mathrm{EPE}_k(x_0) &= \mathrm{E}[(Y - \hat{f}_k(x_0))^2 | X = x_0] \\ &= \sigma^2 + [\mathrm{Bias}^2(\hat{f}_k(x_0)) + \mathrm{Var}_{\mathcal{T}}(\hat{f}_k(x_0))] \qquad (2.46) \\ &= \sigma^2 + \left[f(x_0) - \frac{1}{k} \sum_{\ell=1}^{k} f(x_{(\ell)}) \right]^2 + \frac{\sigma^2}{k}. \qquad (2.47) \end{aligned}$$



Theory Slide 3. Curse of Dimensionality

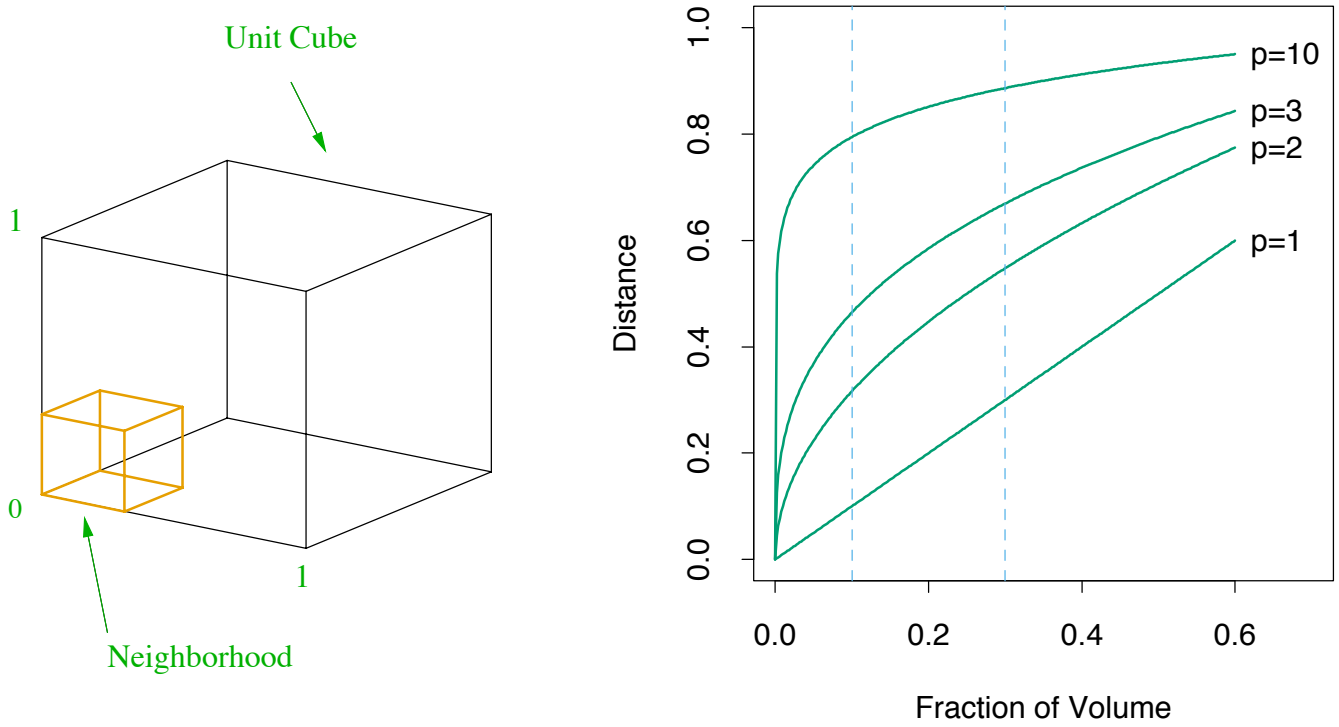


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover* 80% *of the range of each coordinate to capture* 10% *of the data.*

distance from the origin to the closest data point is given by the expression

$$d(p, N) = \left(1 - \frac{1}{2}^{1/N}\right)^{1/p} \qquad (2.24)$$