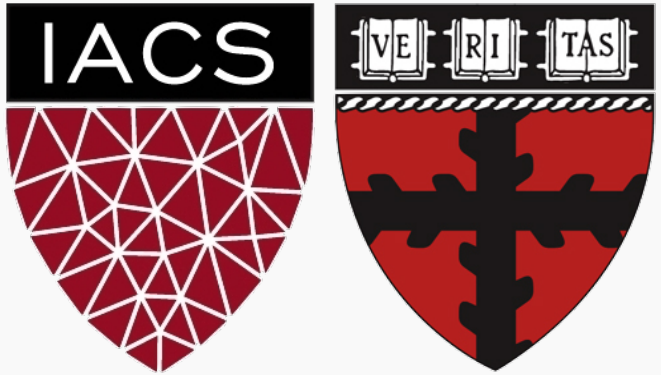
Lecture 25: More on Random Forests

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Tanner



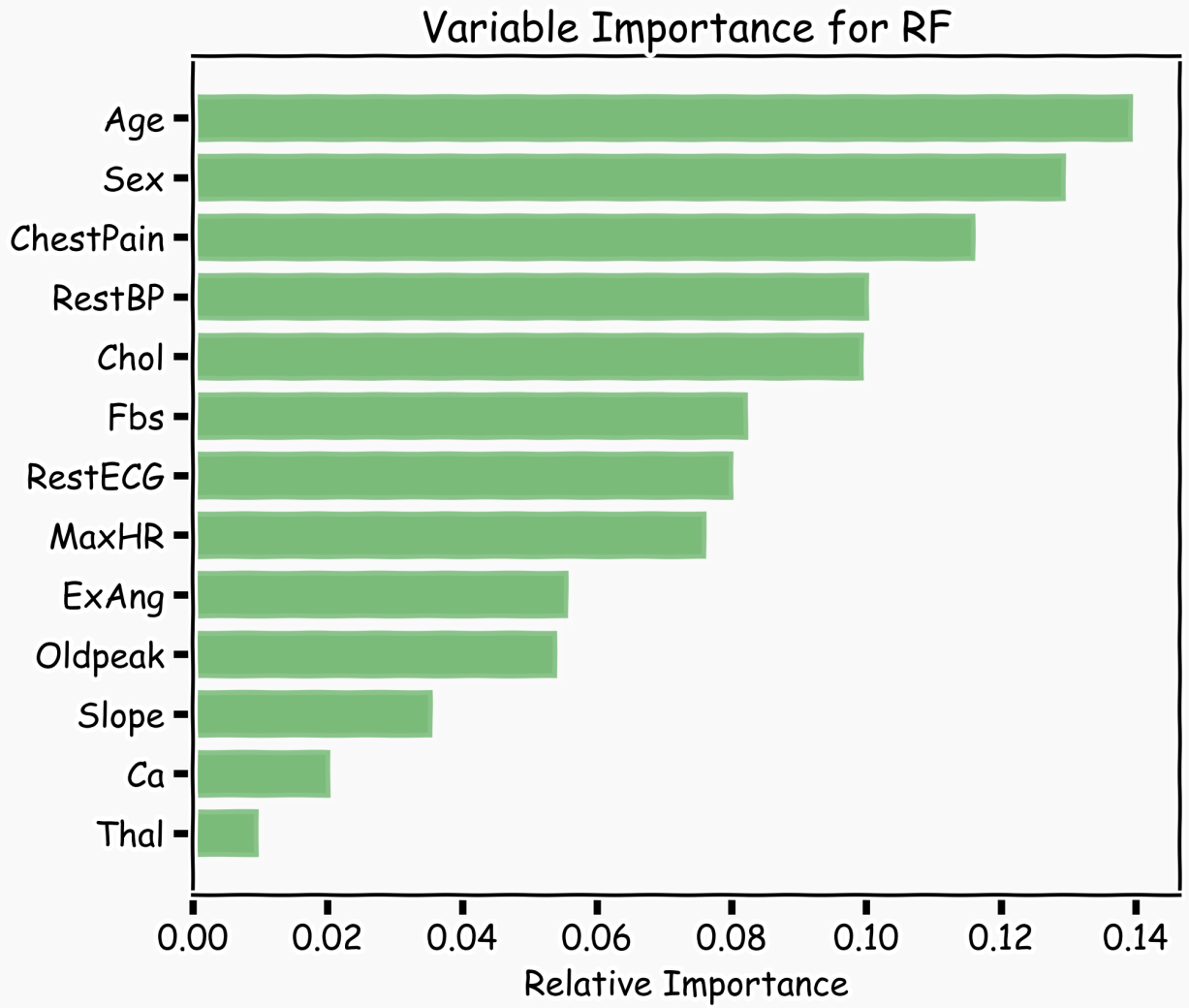
Outline

- Random Forest (RF)
- Tuning the hyperparameters of a RF
- **Feature interpretation in a RF**
- Categorical data
- Missing data
- Imbalanced dataset

Variable Importance for RF

Explaining predictions from tree models is always desired; the patterns uncovered by a model are, in some applications, more important than the model's prediction performance.

A drawback of RF, Bagging, and other ***ensemble methods,*** is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the *logic* of an output through a series of decisions based on predictor values!



Variable Importance for RF

1. Mean Decrease in Impurity (MDI)

- Same as Bagging.
- Calculate the total amount that the RSS (for regression) or Gini index (for classification) is decreased due to splits over a given predictor, averaged over all trees.
- In Scikit-learn you can use `model.feature_importances_`

Variable Importance for RF

2. Permutation Importance

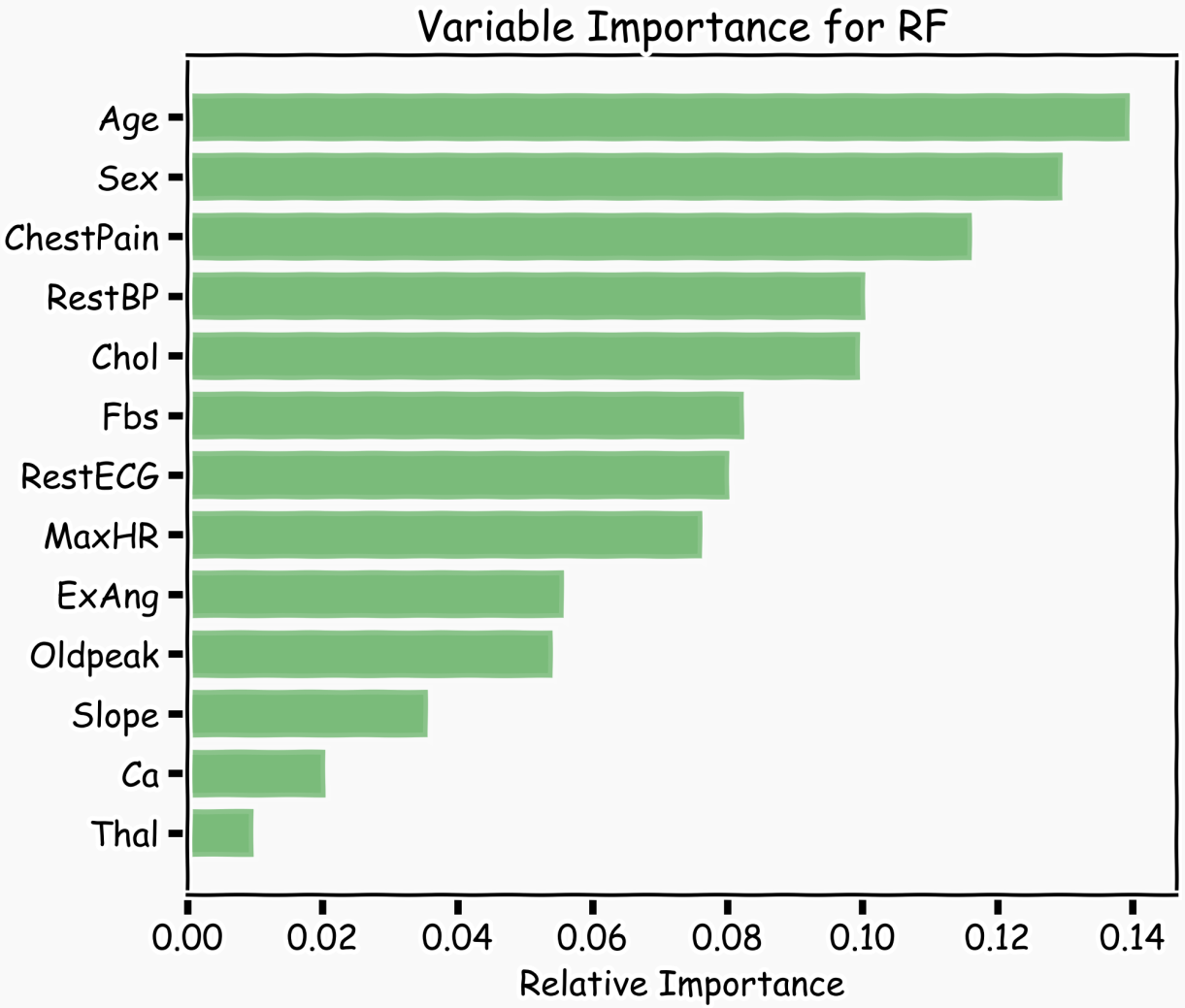
- Record the prediction accuracy on the *oob* samples for each tree.
- Randomly permute the data for column j in the *oob* samples the record the accuracy again.
- The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable j in the random forest.

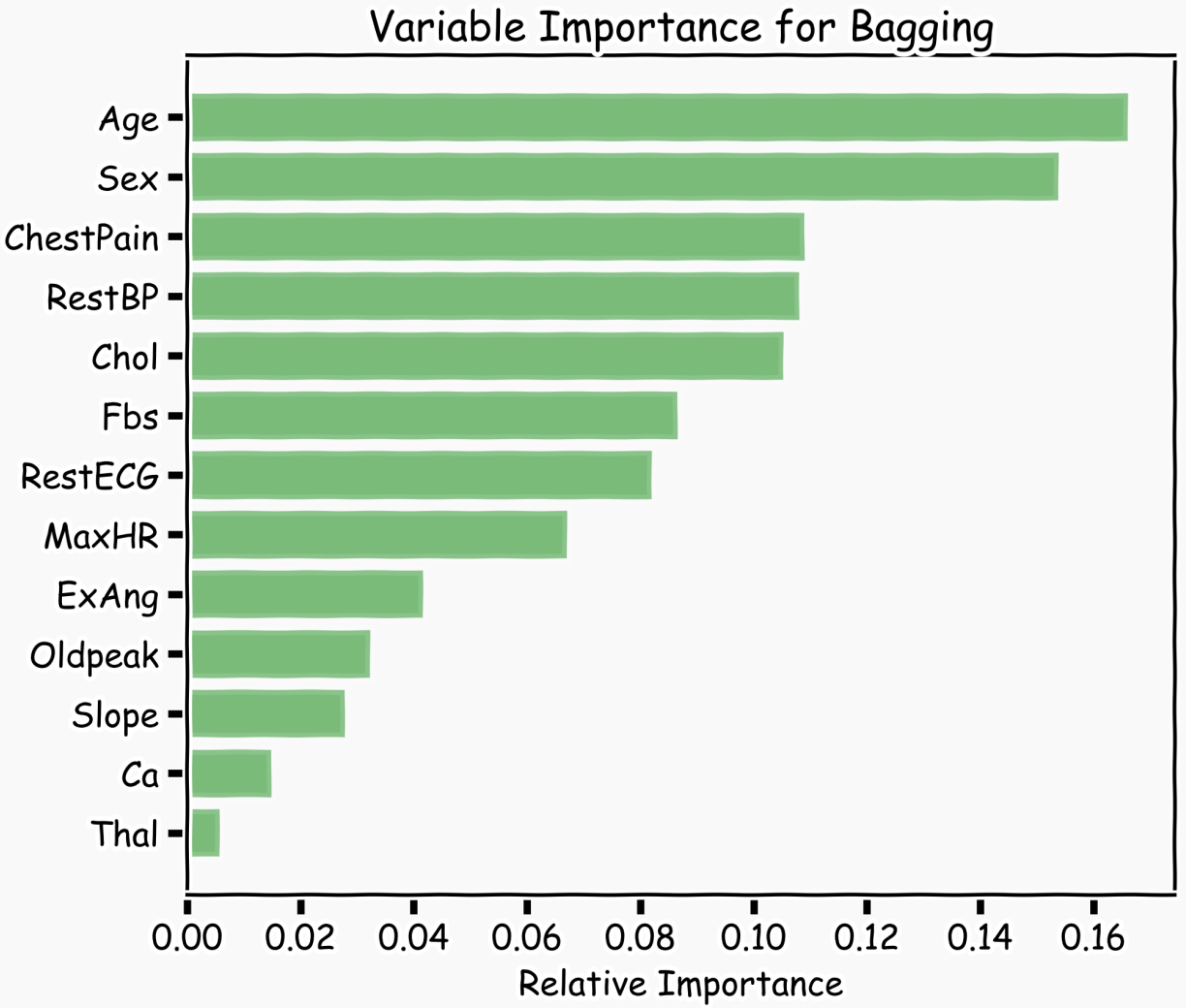
In Scikit-learn provides a library: `sklearn.inspection .permutation_importance`

3. One step further (SHAP values, LIME)

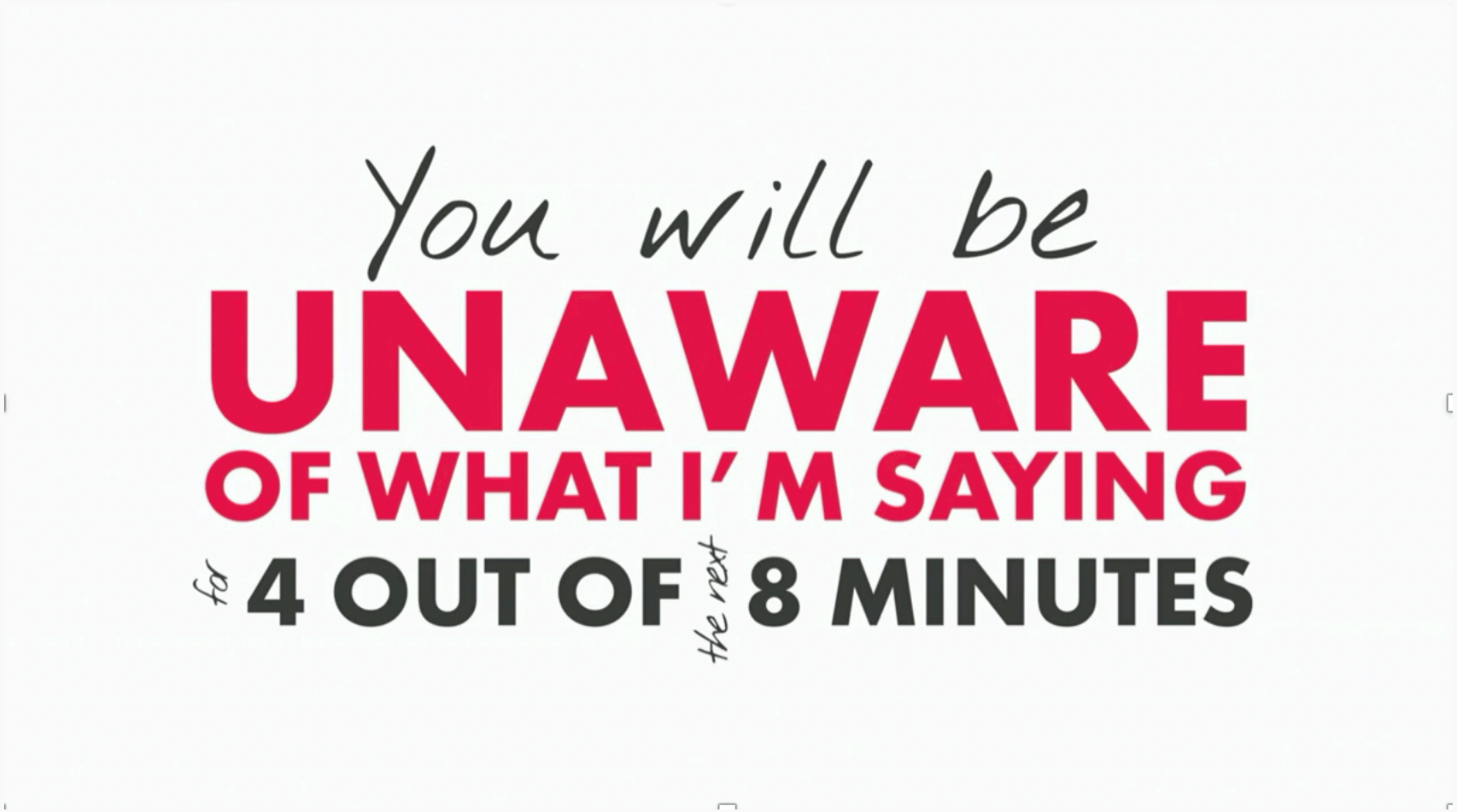
- We will see these methods in later lectures.

Variable Importance for RF

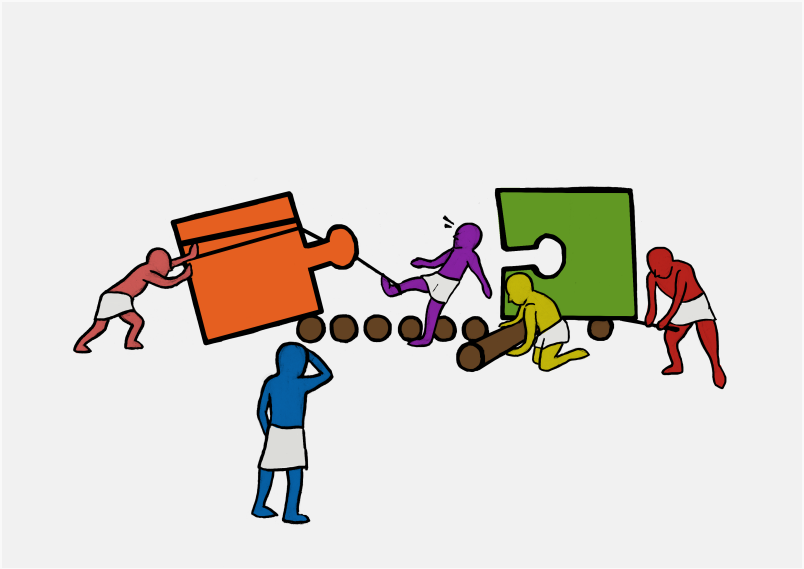




100 trees, max_depth=10



Zeenat Potia



Today's lucky student: Anyone

Exercise: Feature Importance

The goal of this exercise is to compare two feature importance methods; MDI, and Permutation Importance.

Next

- Imbalanced dataset
  - Weighted samples
- Categorical data
- Missing data

Tree-building algorithms

Any tree-bulding algorithm needs to consider many things:

- what metric to decide splitting?
- where to look for splits?
- how to handle categorical predictors?
- how to handle missing features?
- how to handle imbalanced classes?

Categorical Predictors

As it stands, sklearn decision trees do not handle categorical data.

From sklearn documentation:
scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now.

The recommended approach of using Label Encoding converts to integers which the DecisionTreeClassifier() will treat **as numeric**. If your categorical data is not ordinal, this is not good - you'll end up with splits that do not make sense.

Instead use sklearn's `OneHotEncoder` or pandas' `get_dummies`.

Missing Values: Surrogate splits

- When an observation is missing a value for predictor X, it cannot get past a node that splits based on this predictor. What can we do instead?
- We need surrogate splits: Mimic of actual split in a node but using another predictor. It is used in replacement of the original split in case a datapoint has missing data.
- To build them, we search for a feature-threshold pair that most closely matches the original split.
- Association measure is used to select surrogate splits. Depends on the probabilities of sending cases to a particular node + how the new split is separating observations of each class.

Surrogate splits

- The main function is to split when the primary splitter is missing, which *may* never happen in the training data, but being ready for future test data increases robustness.
- No guarantee that useful surrogates can be found.
- Sklearn's decision trees attempt to find five surrogates per split.
- Number of surrogates often varies from split to split.

Surrogate splits - example

- Imagine situation with multiple features, two of them being phone_bill (continuous) and marital_status (categorical).
- Node 1 splits based on phone_bill. Surrogate search might find that marital_status = 1 generates a similar distribution of observations in left and right node.
- This condition is then chosen as top surrogate split (for prediction).

	Left child	Right child
Phone_bill > 100	550R, 99G	50R, 301G
Marital_status = 1	510R, 128G	51R, 311G

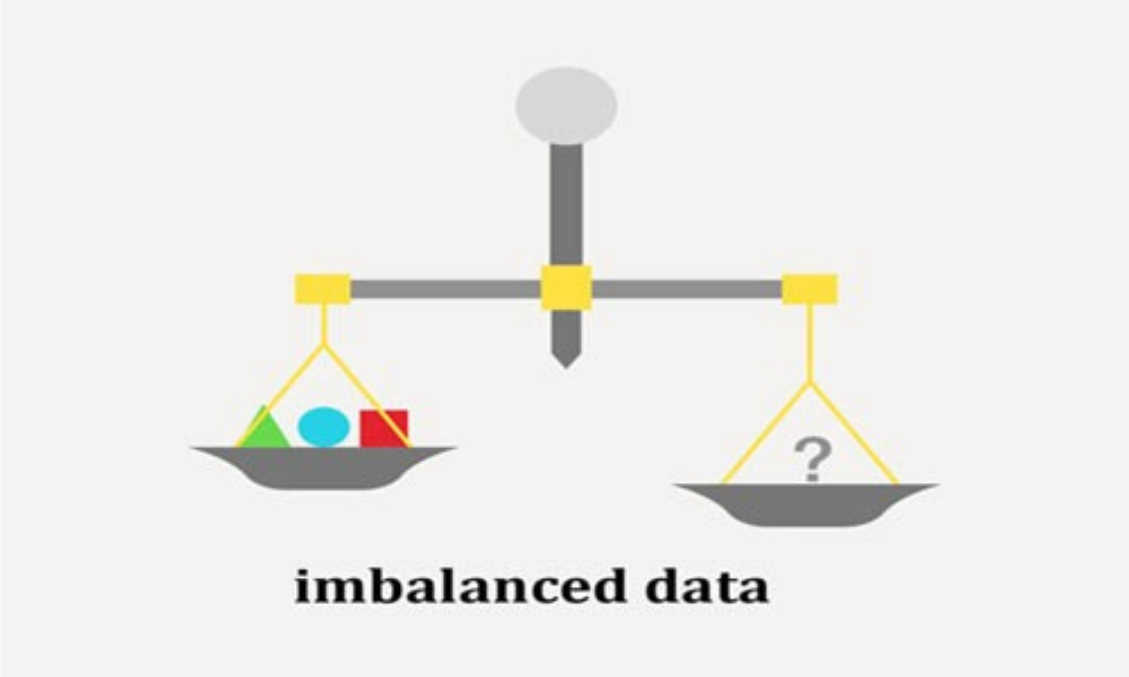
Surrogate splits - example

- In our example, primary splitter = phone_bill
- We might find that surrogate splits include marital status, commute time, age, city of residence.
    - Commute time associated with more time on the phone
    - Older individuals might be more likely to call vs text
    - City variable hard to interpret because we don't know identity of cities
- Surrogates can help us understand primary splitter.
- Surrogate splits perform better when there is multicollinearity (just like imputation!)

Imbalanced classes

Training a RF (or any machine learning model) on an imbalanced dataset can introduce unique challenges to the learning problem.

We saw in a previous lecture that using AUC score (or F1 score) is a better measure of the performance of a model when the classes are not balanced.



source: https://www.nexsoftsys.com/articles/what-is-imbalanced-data.html

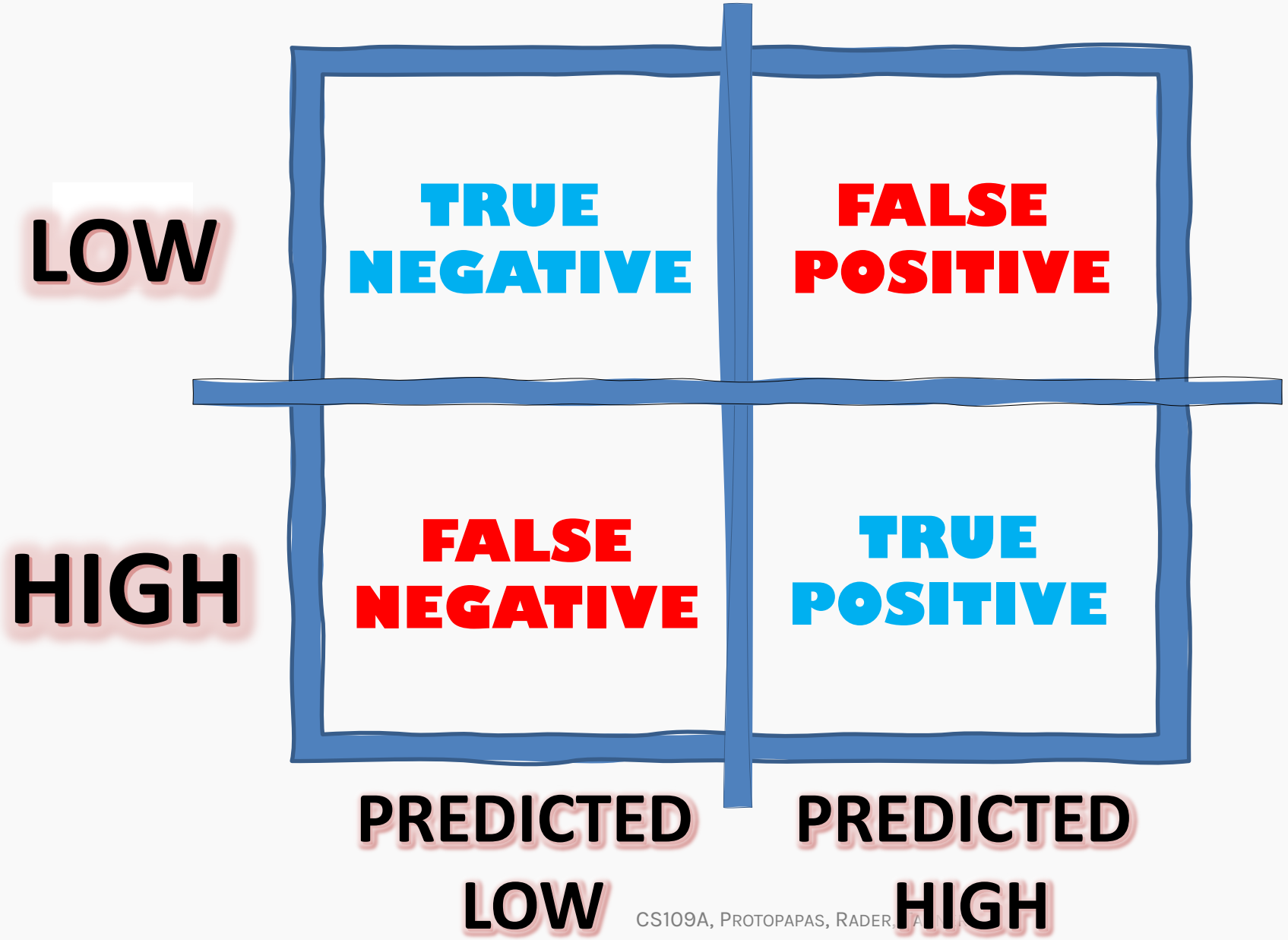
F-score

Accuracy is a great measure but only **when you have symmetric datasets** (false negatives & false positives counts are close), also, **false negatives & false positives have similar costs.**

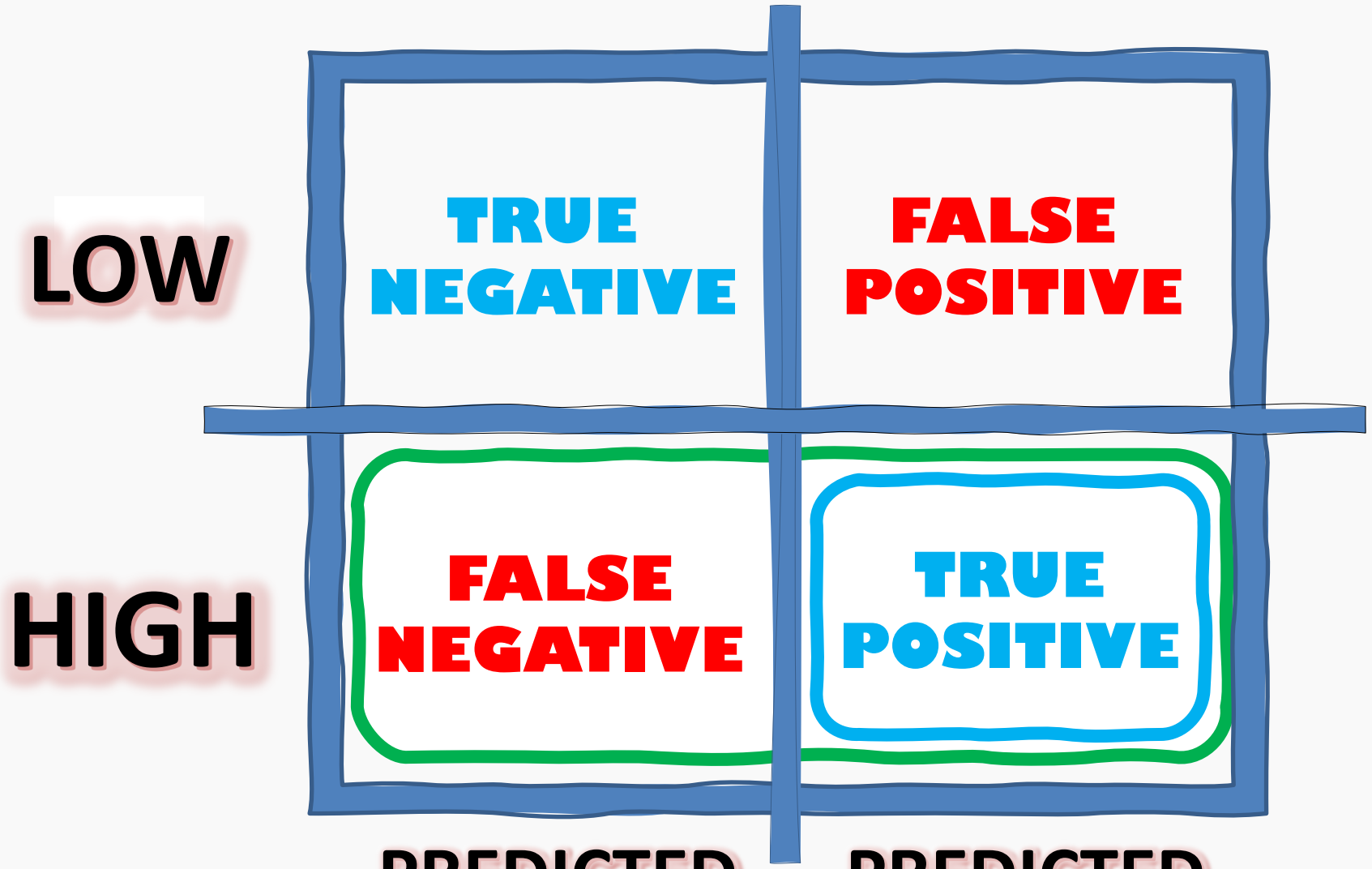
If the cost of false positives and false negatives are different then F1 is your savior. **F1 is best if you have an uneven class distribution**.

$$F1 = \frac{2\ Recall\ * Precision}{Recall + Precision}$$

The 'Confusion' matrix

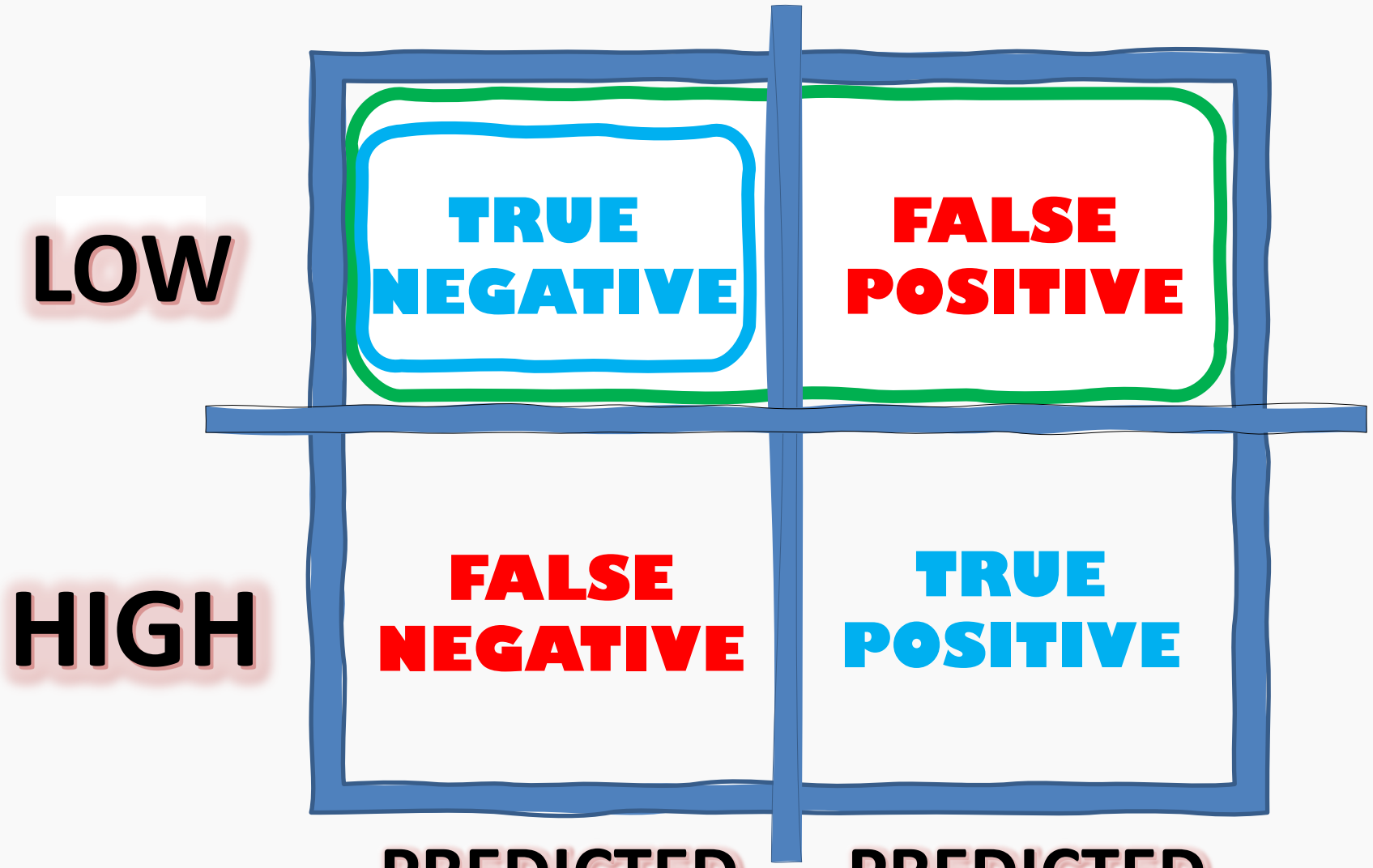


Sensitivity



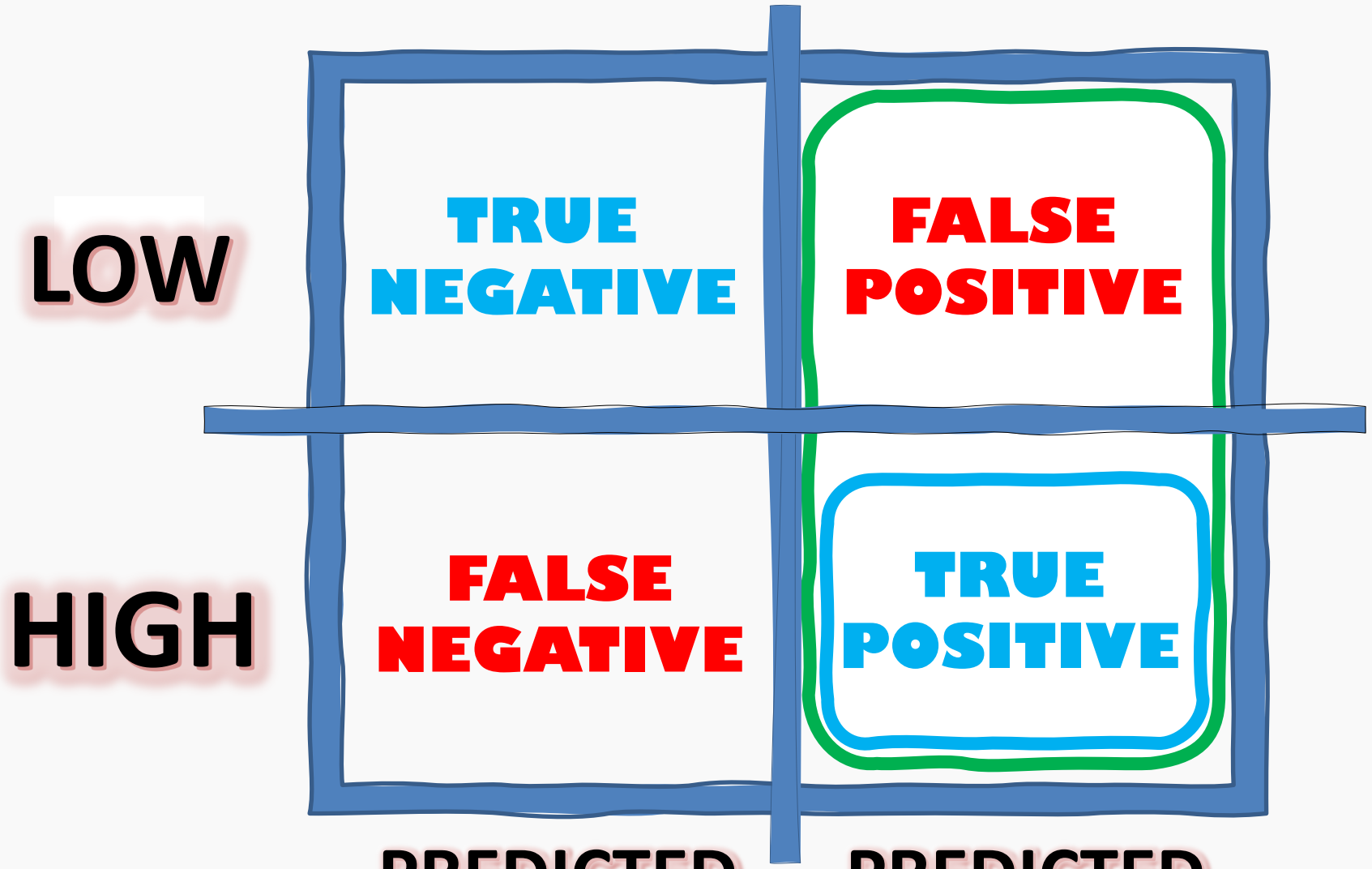
$$\frac{TP}{TP + FN}$$

Specificity



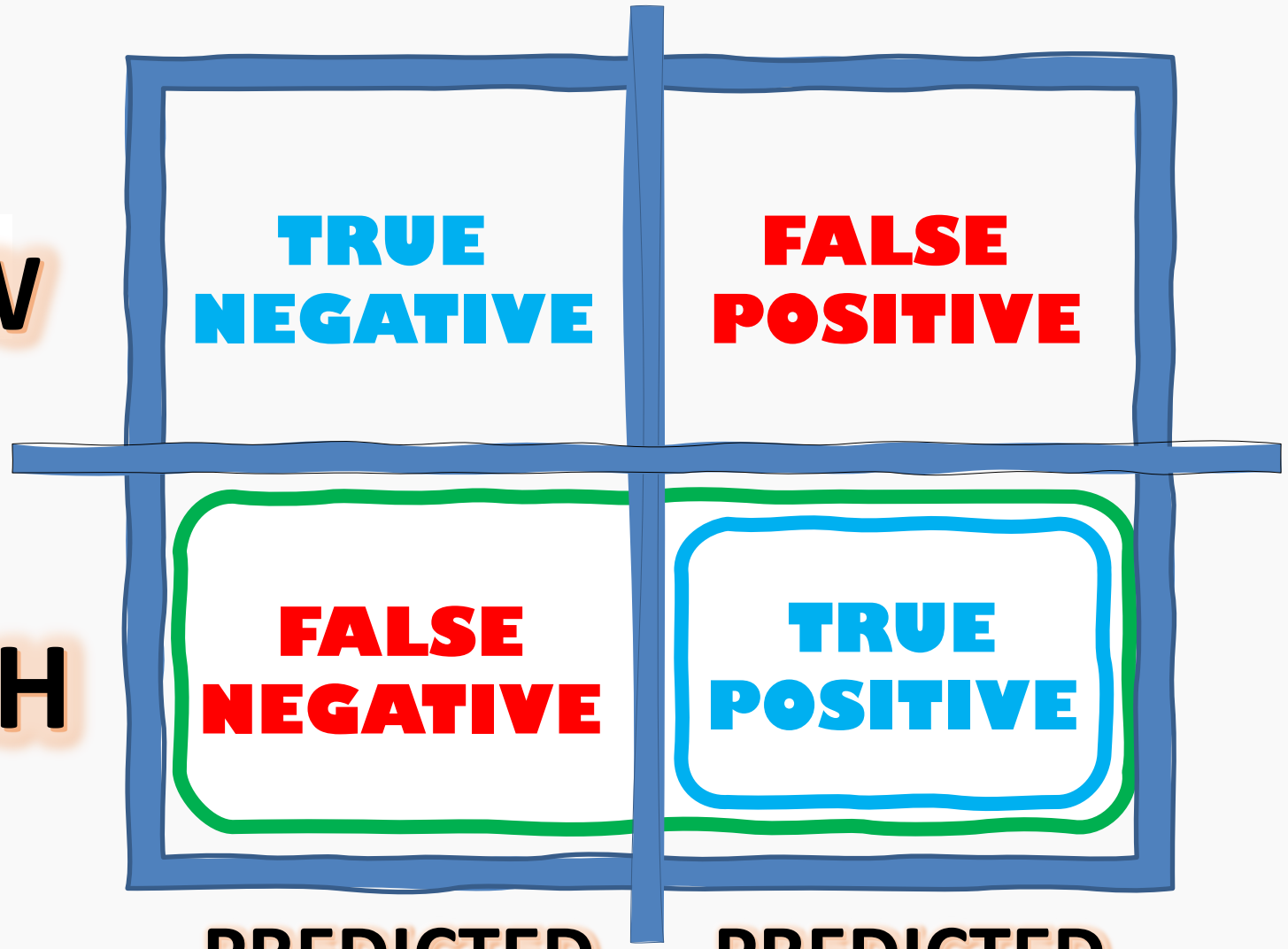
$$\frac{TN}{TN + FP}$$

Precision



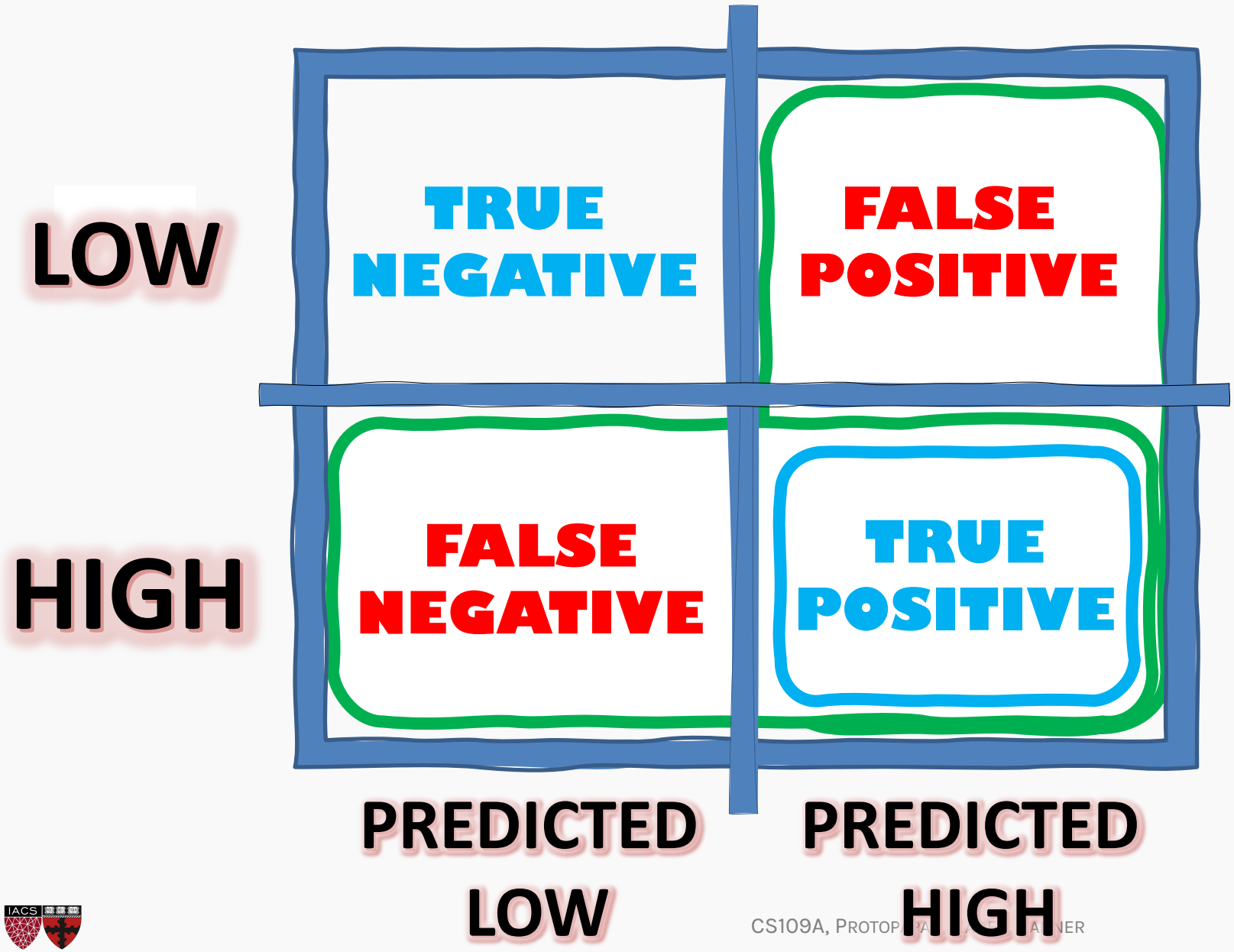
$$\frac{TP}{TP + FP}$$

Recall



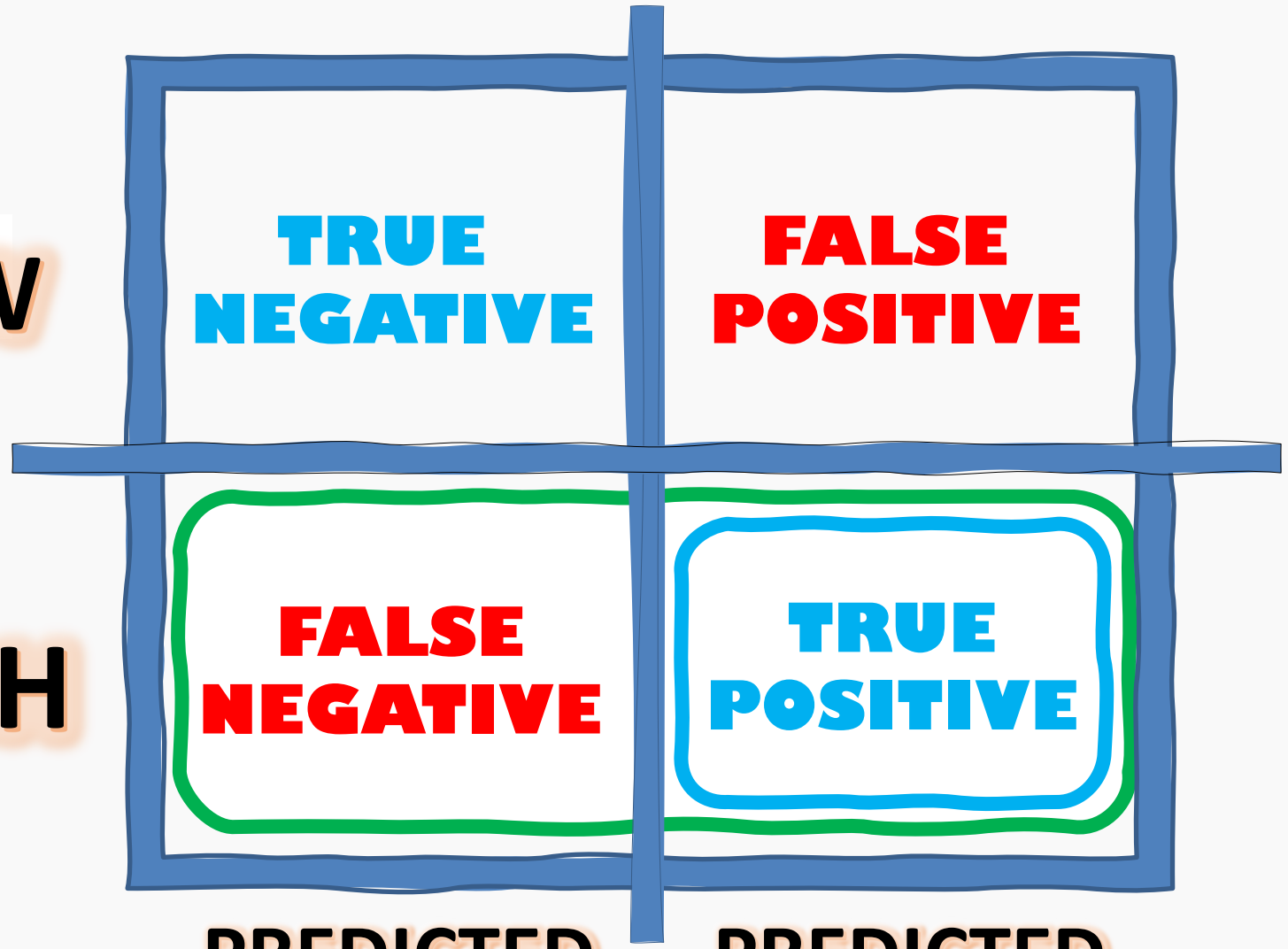
$$\frac{TP}{TP + FN}$$

F1-score



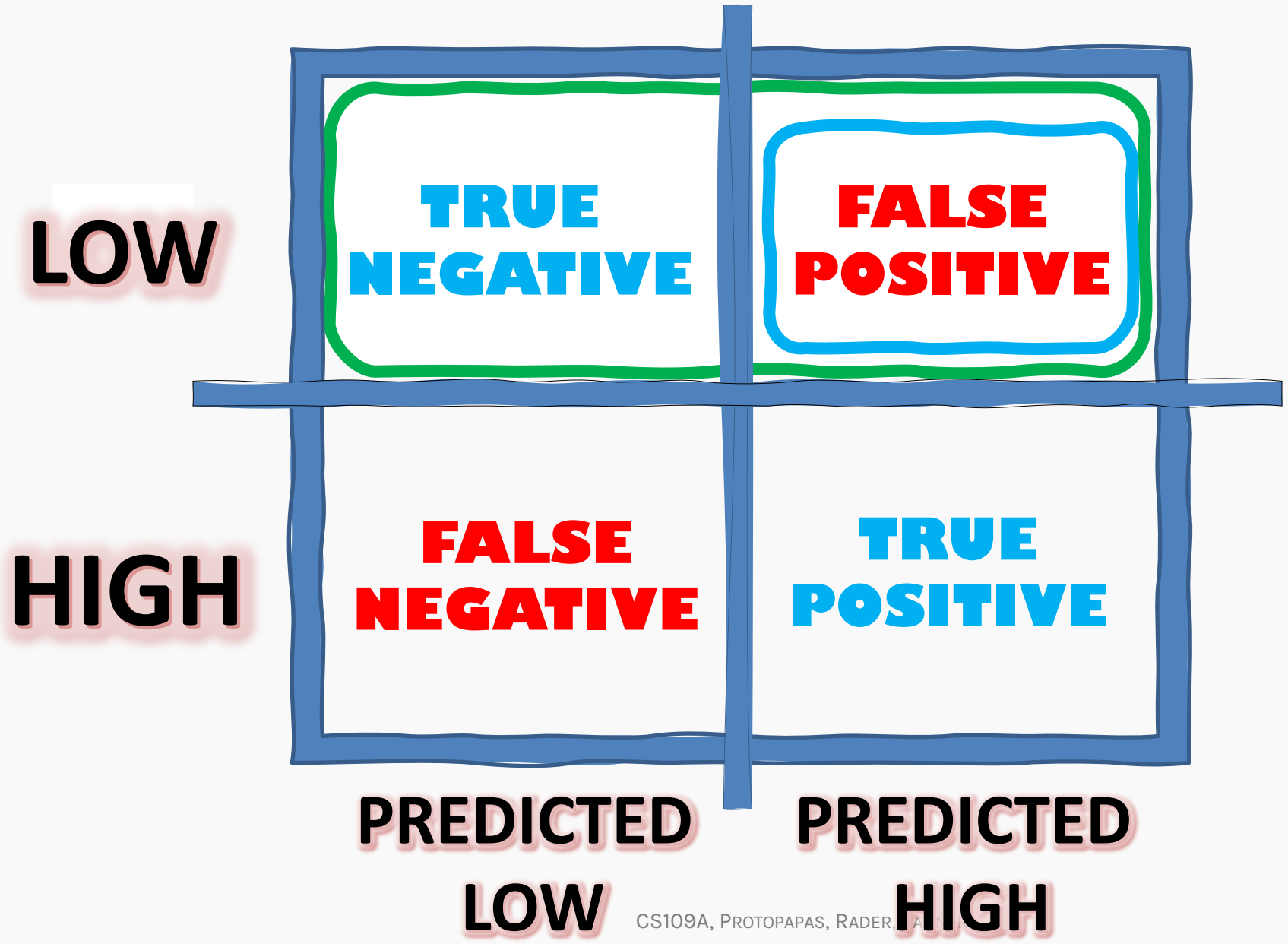
$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

True Positive Rate



$$\frac{TP}{TP + FN}$$

False Positive Rate



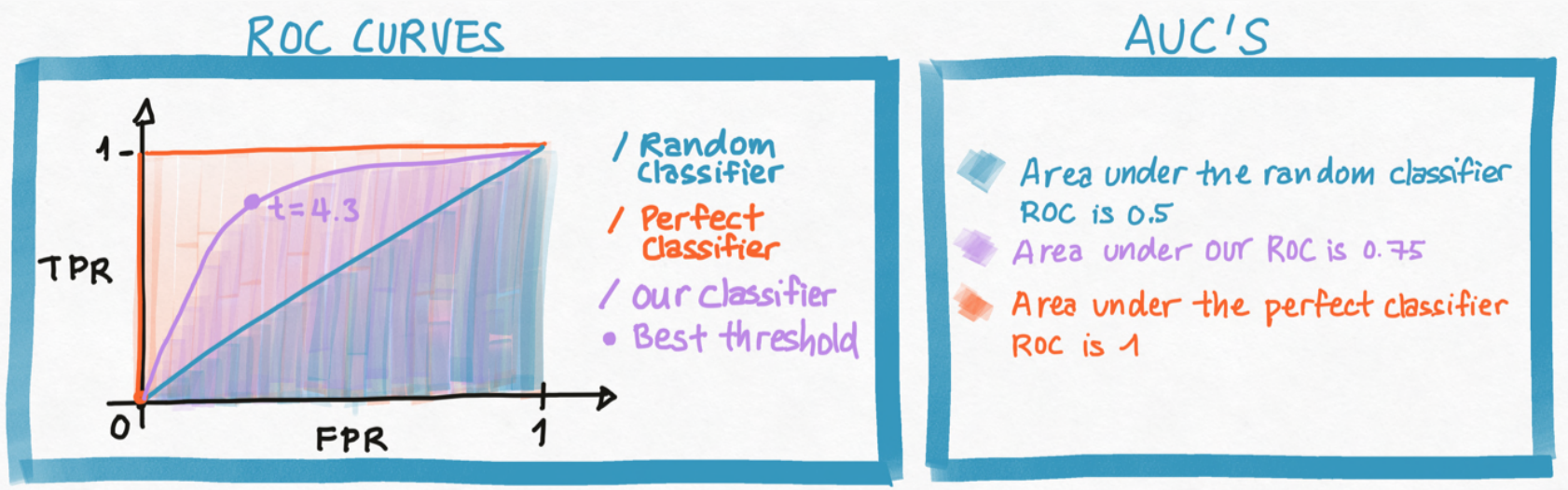
$$\frac{FP}{TN + FP}$$

Area Under the ROC curve (review)

The ROC curve allows us to find the classification threshold that gives the best FPR and TPR trade-off.

If all we are interested in is comparing our classifier against a perfect classifier and a random classifier, then we want to summarize the information in the ROC curve.

We summarize the ROC by computing the Area Under the ROC curve. For the perfect classifier the AUC is 1, the random classifier has AUC of 0.5. We want our classifier to have AUC as close to 1 as possible.



Imbalanced classes in sklearn

Sklearn we can provide the class_weight as a dictionary or use

class_weight = balanced

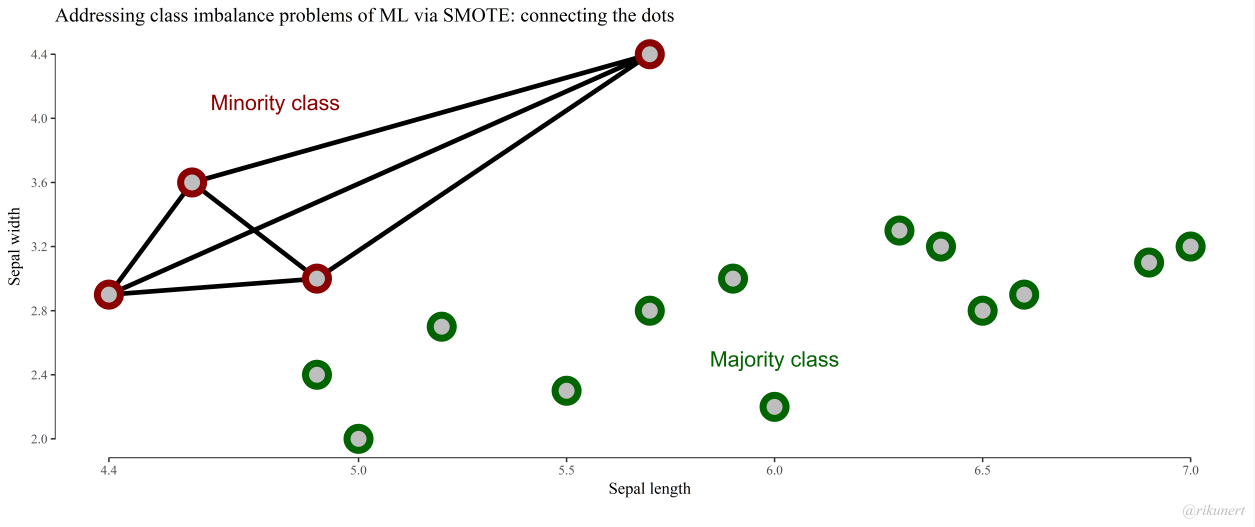
Then it uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as

$$W_k = \frac{N}{K \times N_K}$$

Where N is the total number of samples, N_k is the number of samples in class K and K is the total number of classes.

One step further

SMOTE: A library that does upsampling by "interpolation". See exercise.



Tree-building algorithms

ID3: Iterative Dichotomiser 3. Developed in the 80s by Ross Quinlan.

- Uses the top-down induction approach described previously.
- Works with the **Information Gain** (IG) metric.
- At each step, algorithm chooses feature to split on and calculates IG for each possible split along that feature.
- Greedy algorithm.

Tree-building algorithms

C4.5: Successor of ID3, also developed by Quinlan ('93). Main improvements over ID3:

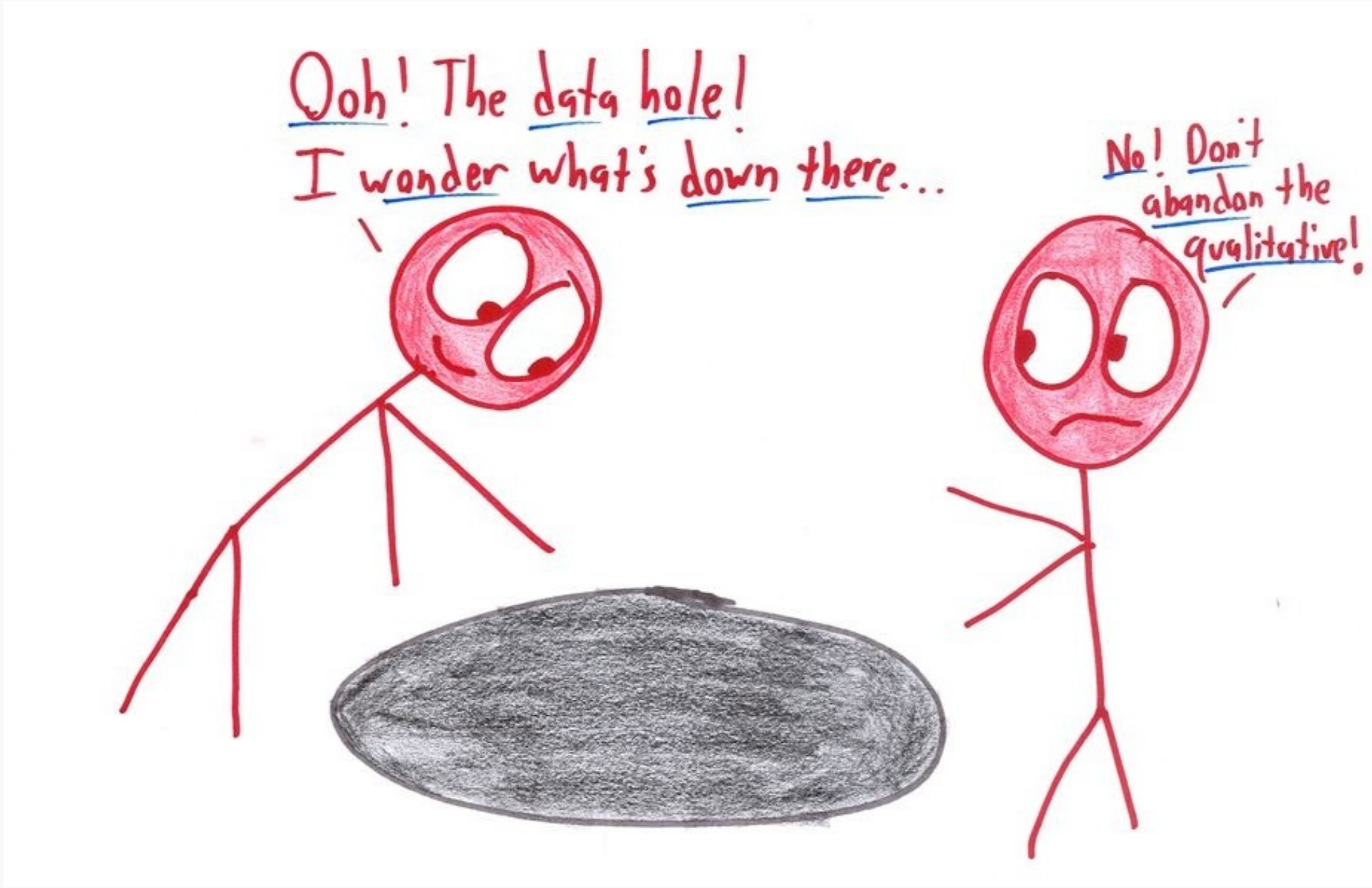
- Works with both continuous and discrete features, while ID3 only works with discrete values.
- Handles missing values by using **fractional cases** (penalizes splits that have multiple missing values during training, fractionally assigns the datapoint to all possible outcomes).
- Reduces overfitting by **pruning**, a bottom-up tree reduction technique.
- Accepts weighting of input data.
- Works with multiclass response variables.

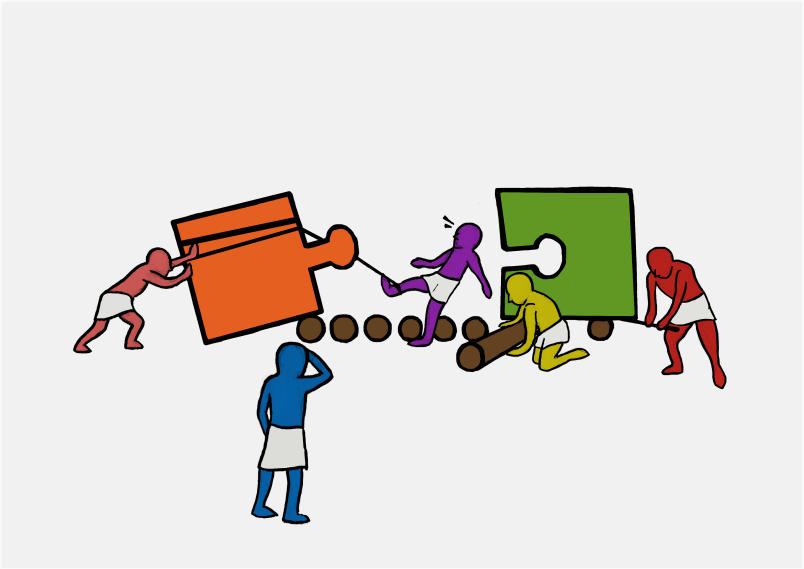
Tree-building algorithms

CART: Most popular tree-builder. Introduced by Breiman et al. in 1984. Usually used with Gini purity metric.

- Main characteristic: **builds binary trees.**
- Can work with discrete, continuous and categorical values in features.
- Handles missing values by using **surrogate splits**.
- Uses **cost-complexity pruning**.
- Sklearn uses CART for its trees.

Note: "scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now."


Ooh! The data hole!
I wonder what's down there...
No! Don't abandon the qualitative!



Today's lucky student: Anyone

Exercise goal

Investigate various methods dealing with class imbalances