Data Wrangling and Data Analysis
Machine learning!
#3 mostly classification

Daniel Oberski, Ayoub Bagheri

Department of Methodology & Statistics

Utrecht University

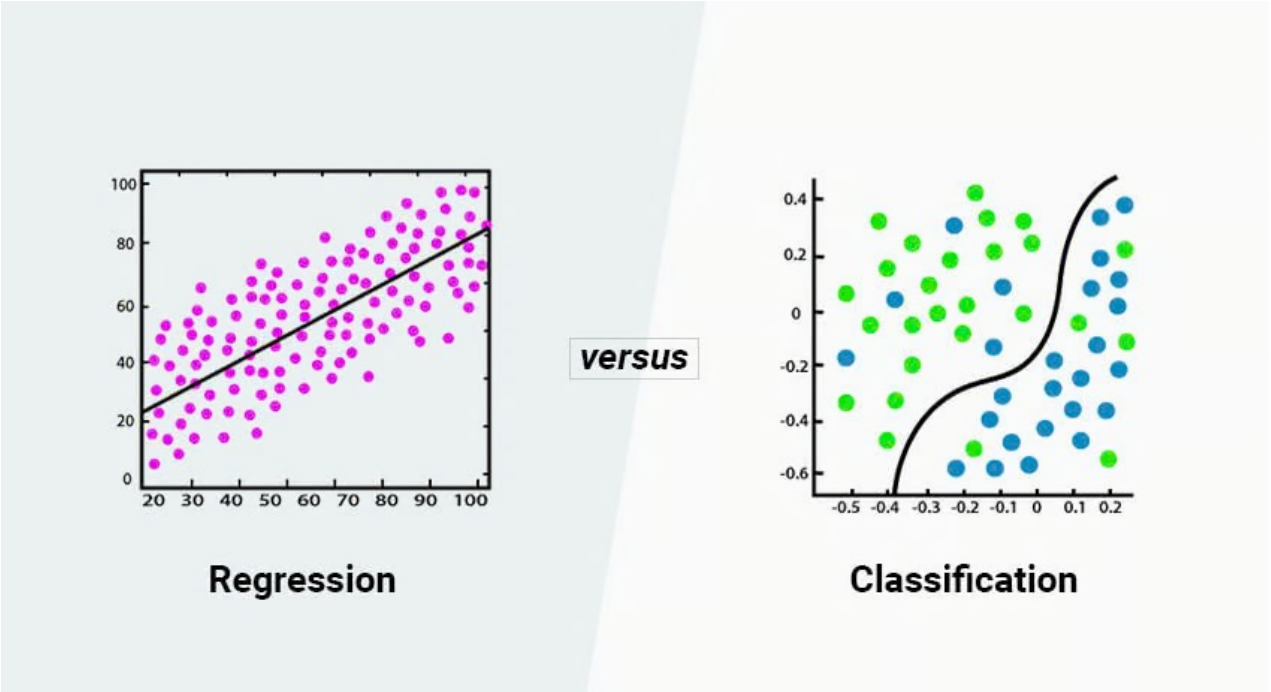
VERSION: 2025-10-08

Supervised machine learning

1. Regression (predicting continuous outcomes)
2. Regression evaluation
3. **Classification (predicting discrete outcomes)**

Classification

- **Supervised learning**: Learning a function that maps an input to an output based on example input-output pairs.
  - infer a function from labeled training data
  - use the inferred function to label new instances



Classification

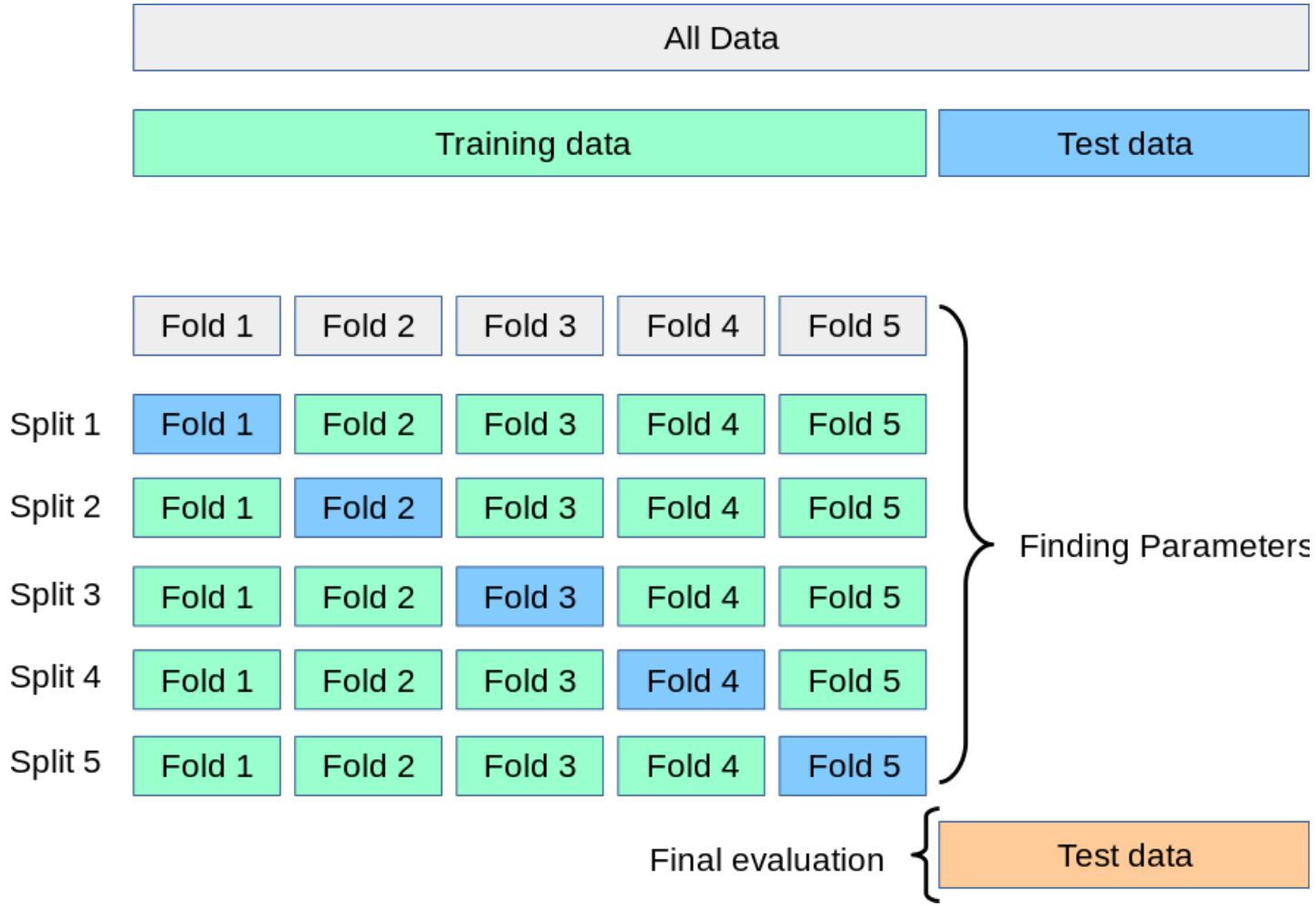
The thing you're trying to predict is **discrete**:

- *Titanic*: Survival/Nonsurvival
- Banking data: Default on/payment of debt
- GPS/Accelerometer data: Work/Home/Friend/Parking/Other
- Imagenet: gazelle/tank/pirate/sea lion/tandem bicycle/: : :
- Etc.

Classification types

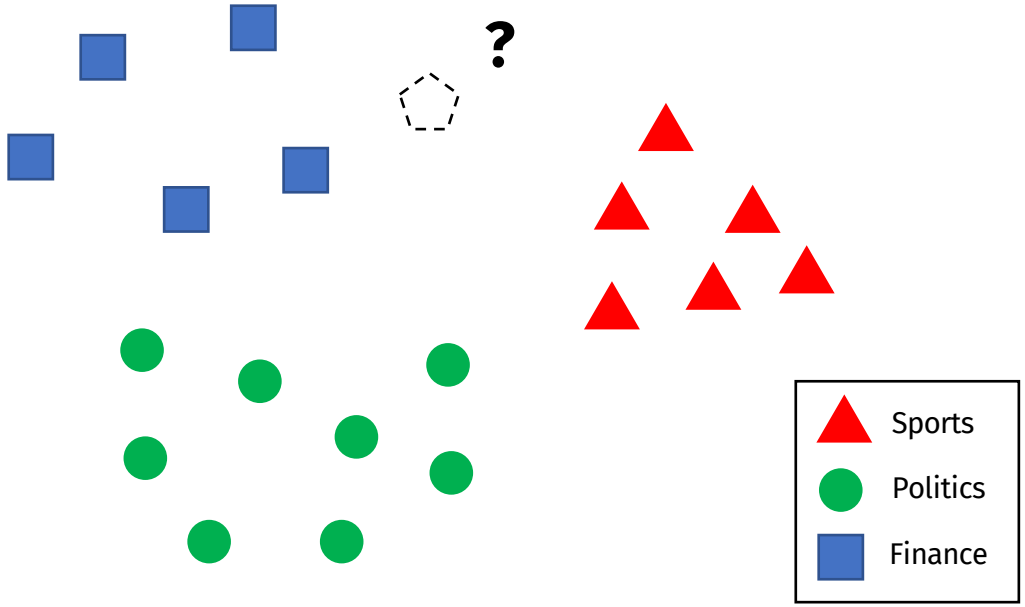
1. **Binary classification**:
   - **Definition**: Involves two categories or classes.
   - **Examples**: True/False, Yes/No, Spam/Not Spam, Positive/Negative sentiment.
2. **Multiclass classification**:
   - **Definition**: Involves more than two categories or classes.
   - **Examples**: Classifying animal images into Cats/Dogs/Birds, predicting the genre of a book (e.g., Fiction/Non-fiction/Science/History).
3. **Multilabel classification**:
   - **Definition**: Each instance can belong to multiple classes simultaneously.
   - **Examples**: Tagging a research article with multiple topics like "Machine Learning," "Natural Language Processing," and "Healthcare.", Assigning multiple genres to a book.

From Data collection to K-fold cross validation



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

How to classify?



Classification: definition

- *Input*:
  - a sample d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

- *Output*: a predicted class $c \in C$

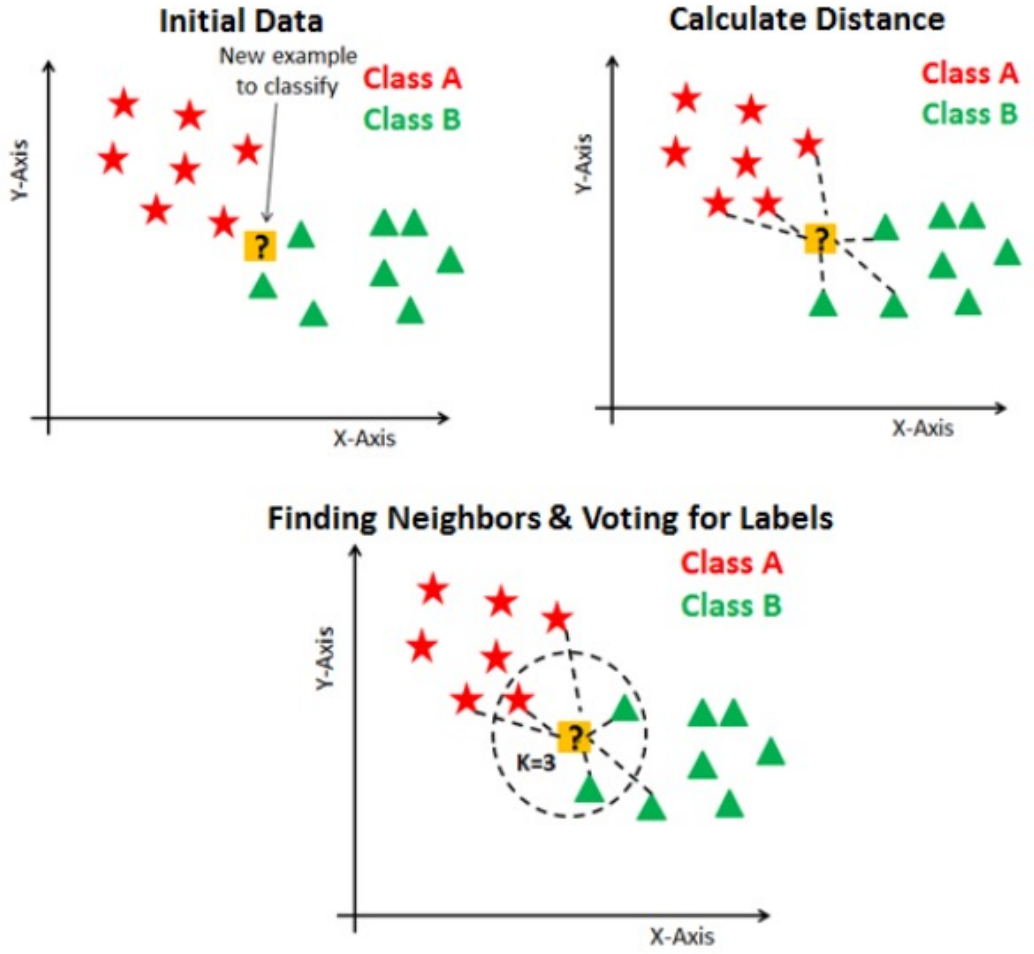
Common algorithms

- K-nearest neighbors
- Classification trees (decision trees)
- Logistic regression
- Support vector machines
- Neural networks
- Deep learning
- And …

K-nearest neighbors

- **K-Nearest Neighbors (KNN)** is a simple, non-parametric algorithm that classifies data points based on the majority class of their K nearest neighbors in feature space, using a distance metric such as Euclidean distance.

- Non-parametric methods: make no strong assumptions about the data's underlying distribution.



Example from the book: ISLR

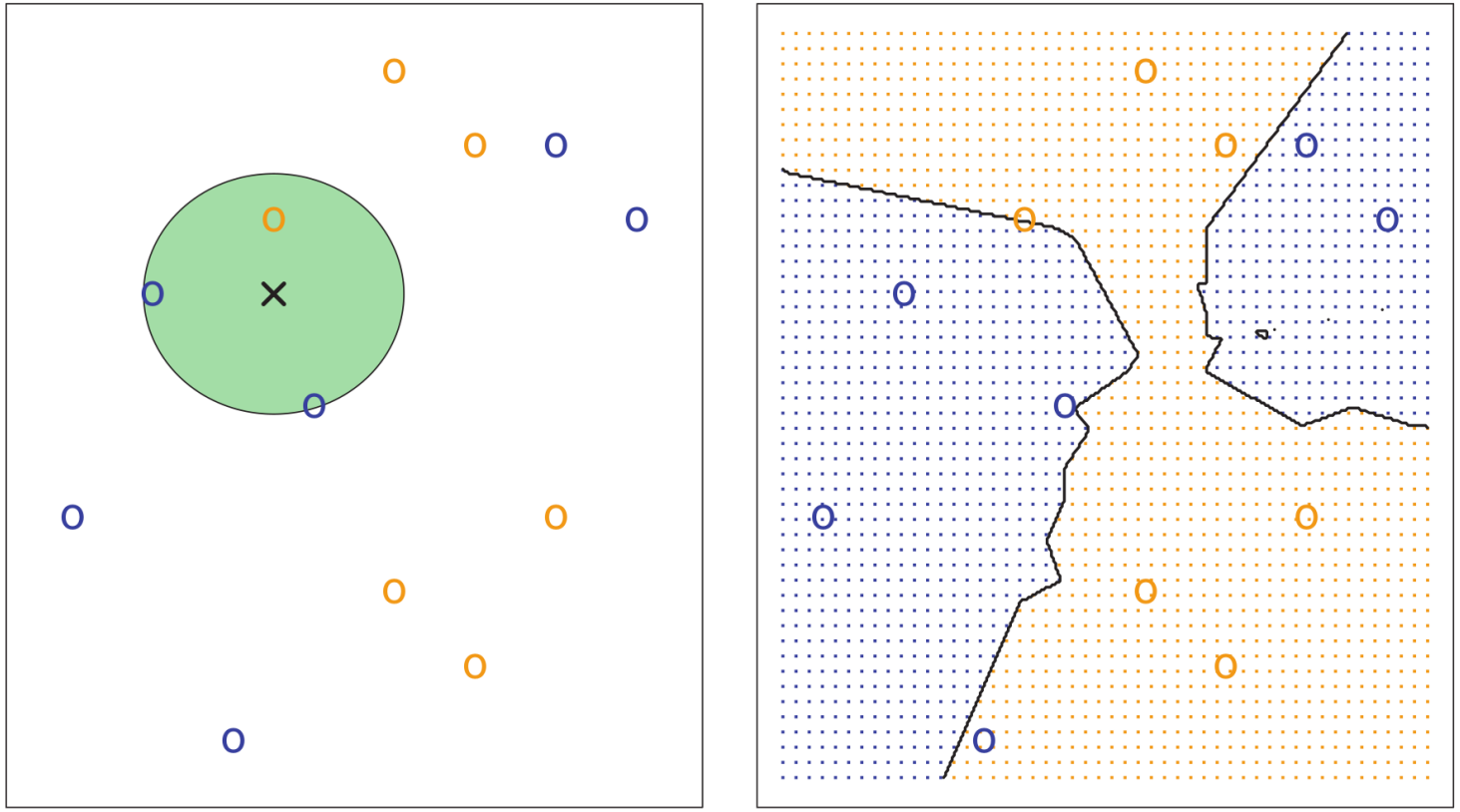


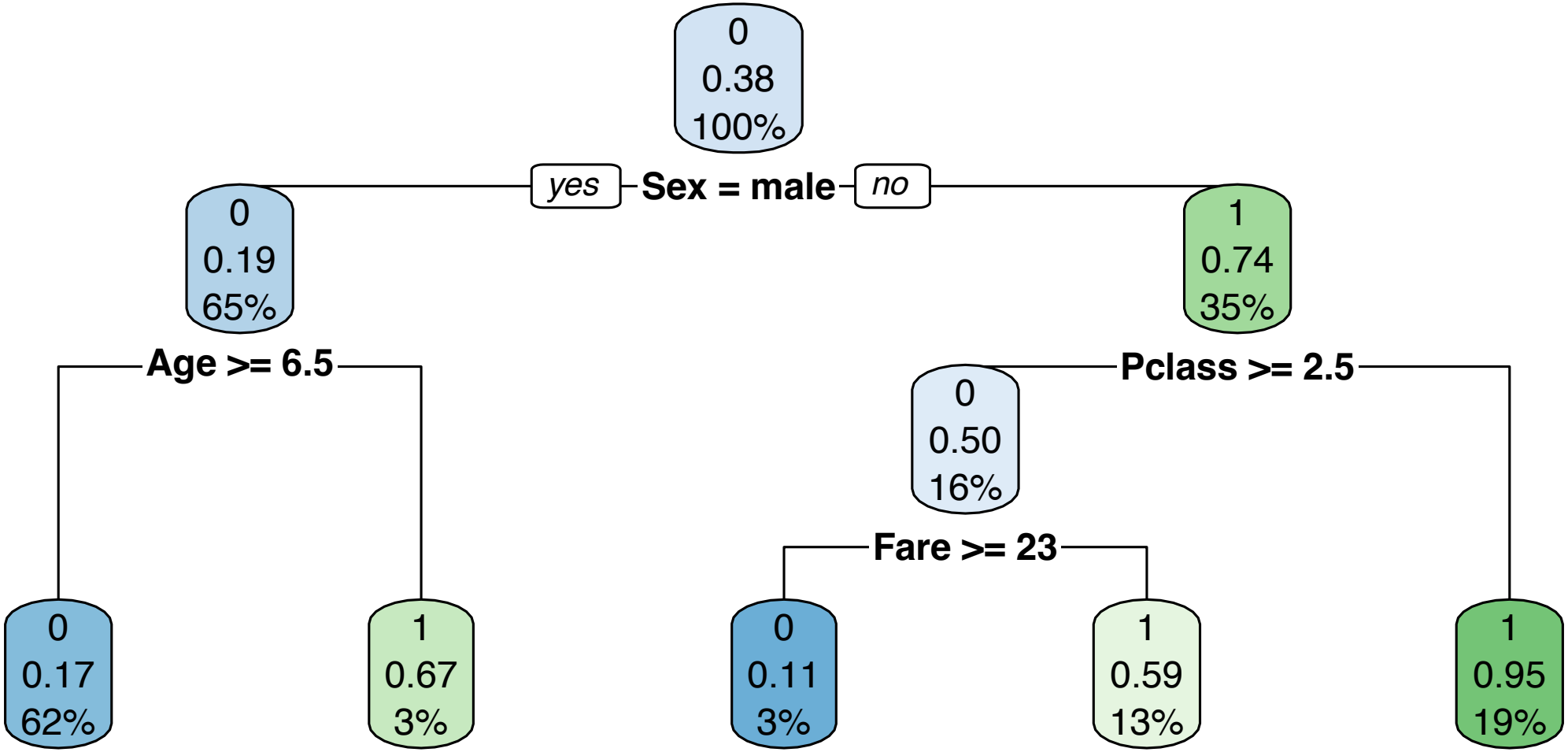
FIGURE 2.14. *The KNN approach, using* $K = 3$*, is illustrated in a s*

Classification trees

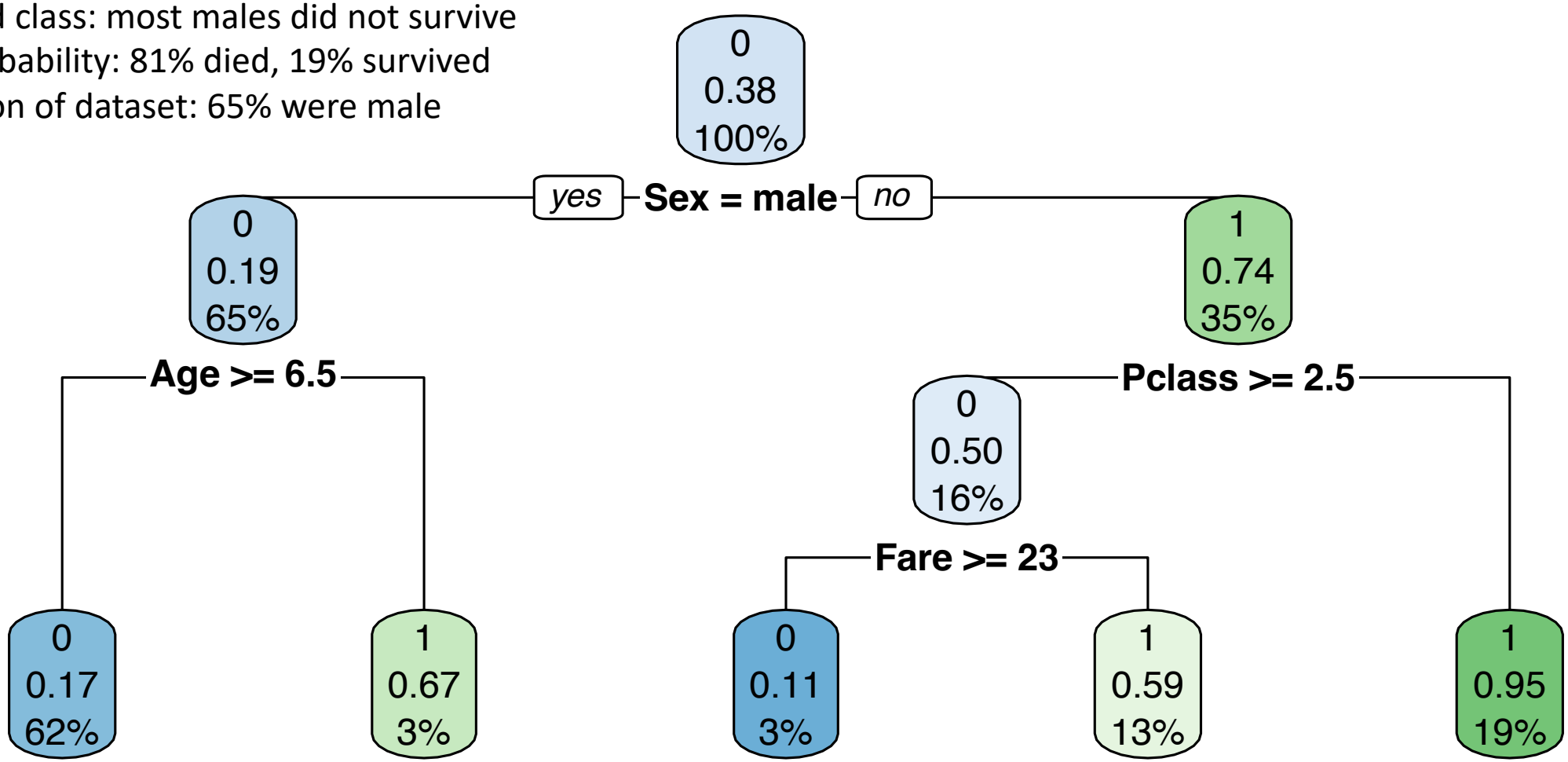
Titanic data

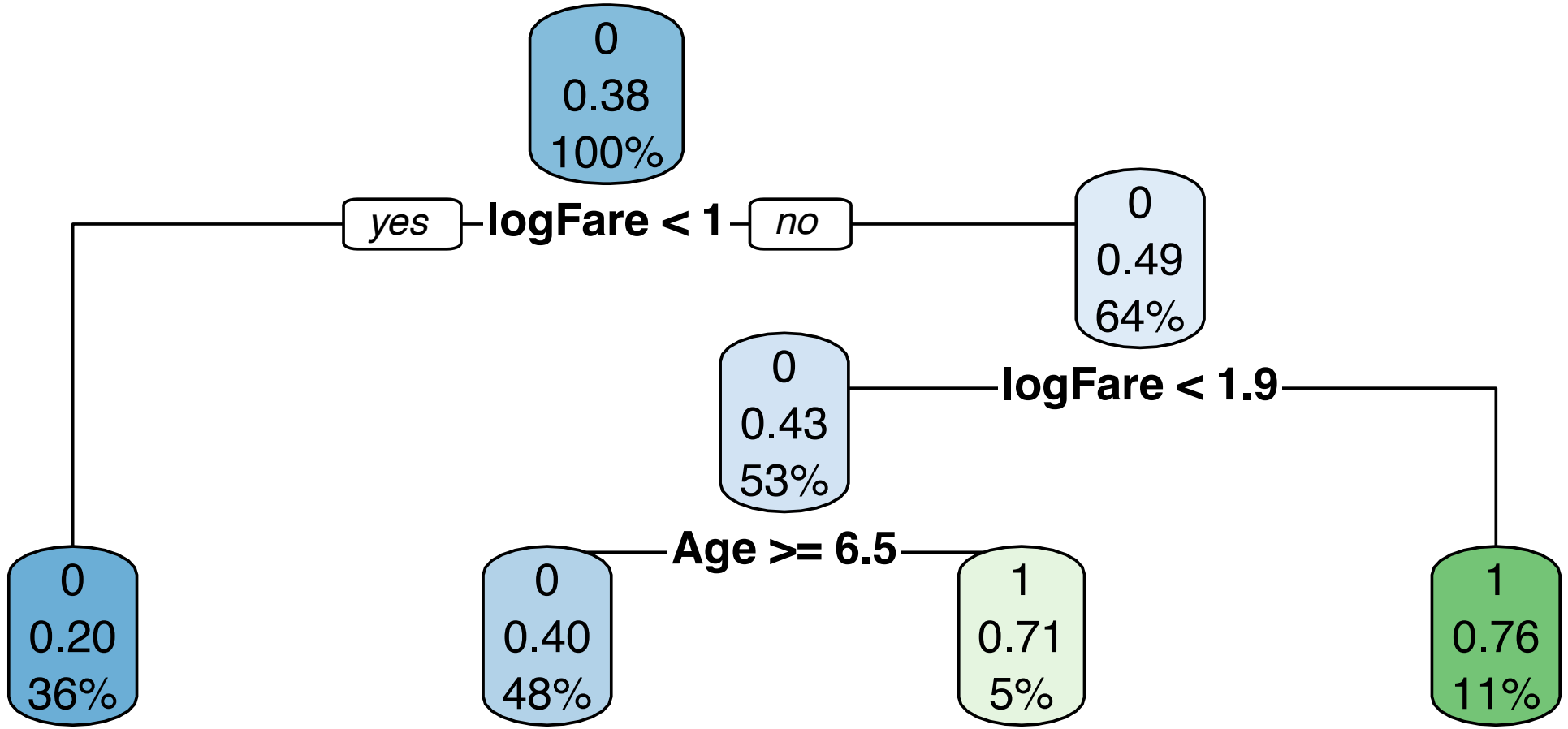
```
df = pd.read_csv('assets/train.csv')
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S


0
0.38
100%
yes
Sex = male
no
0
0.19
65%
1
0.74
35%
Age >= 6.5
Pclass >= 2.5
0
0.17
62%
1
0.67
3%
0
0.50
16%
Fare >= 23
0
0.11
3%
1
0.59
13%
1
0.95
19%

Predicted class: most males did not survive
Class probability: 81% died, 19% survived
Proportion of dataset: 65% were male






```
set.seed(73) # To make your tree reproducible
```

Learning algorithm

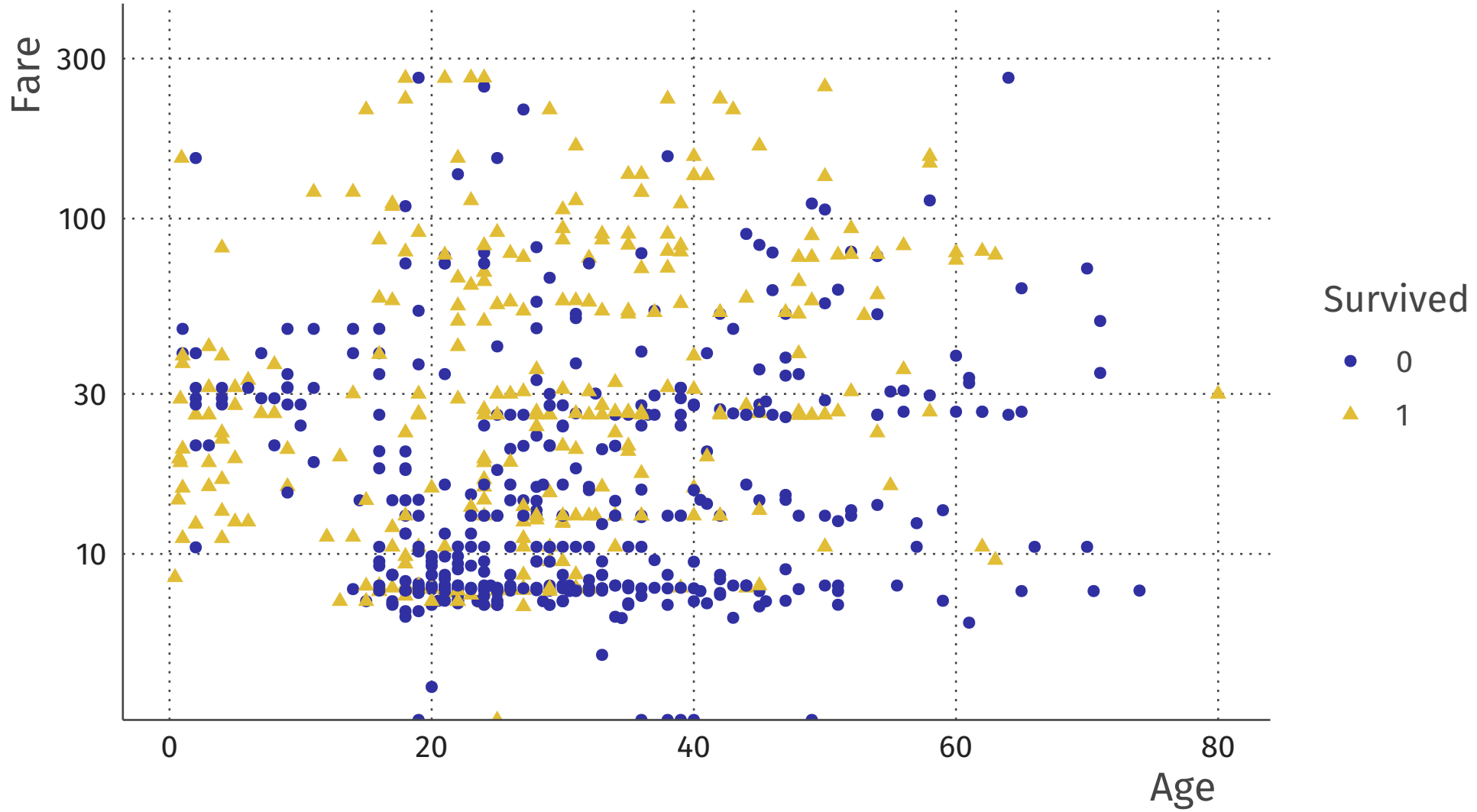
Recursive partitioning

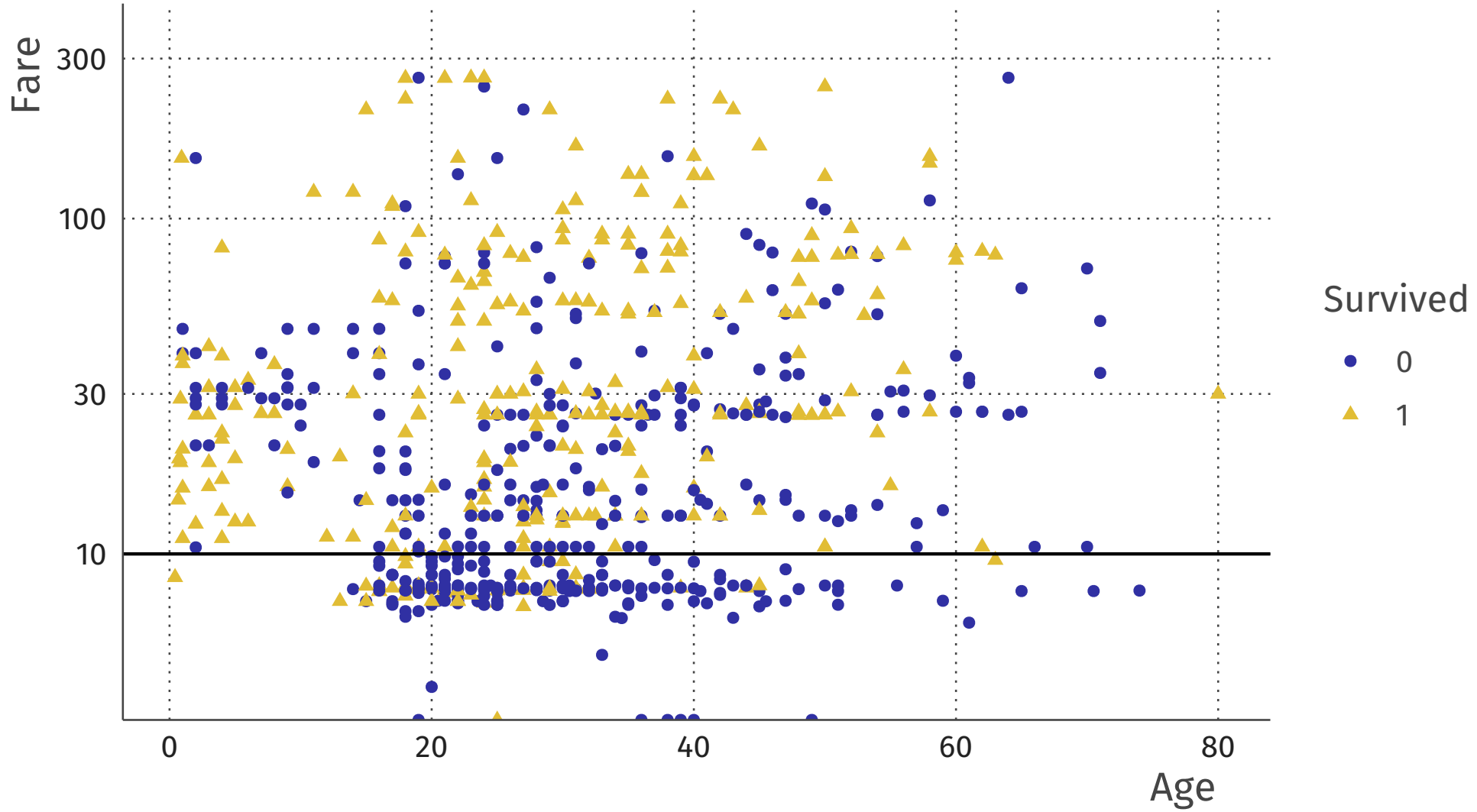
1. Find the split that makes observations (data points) as similar as possible on the outcome within that split;
2. Within each resulting group, do (1).

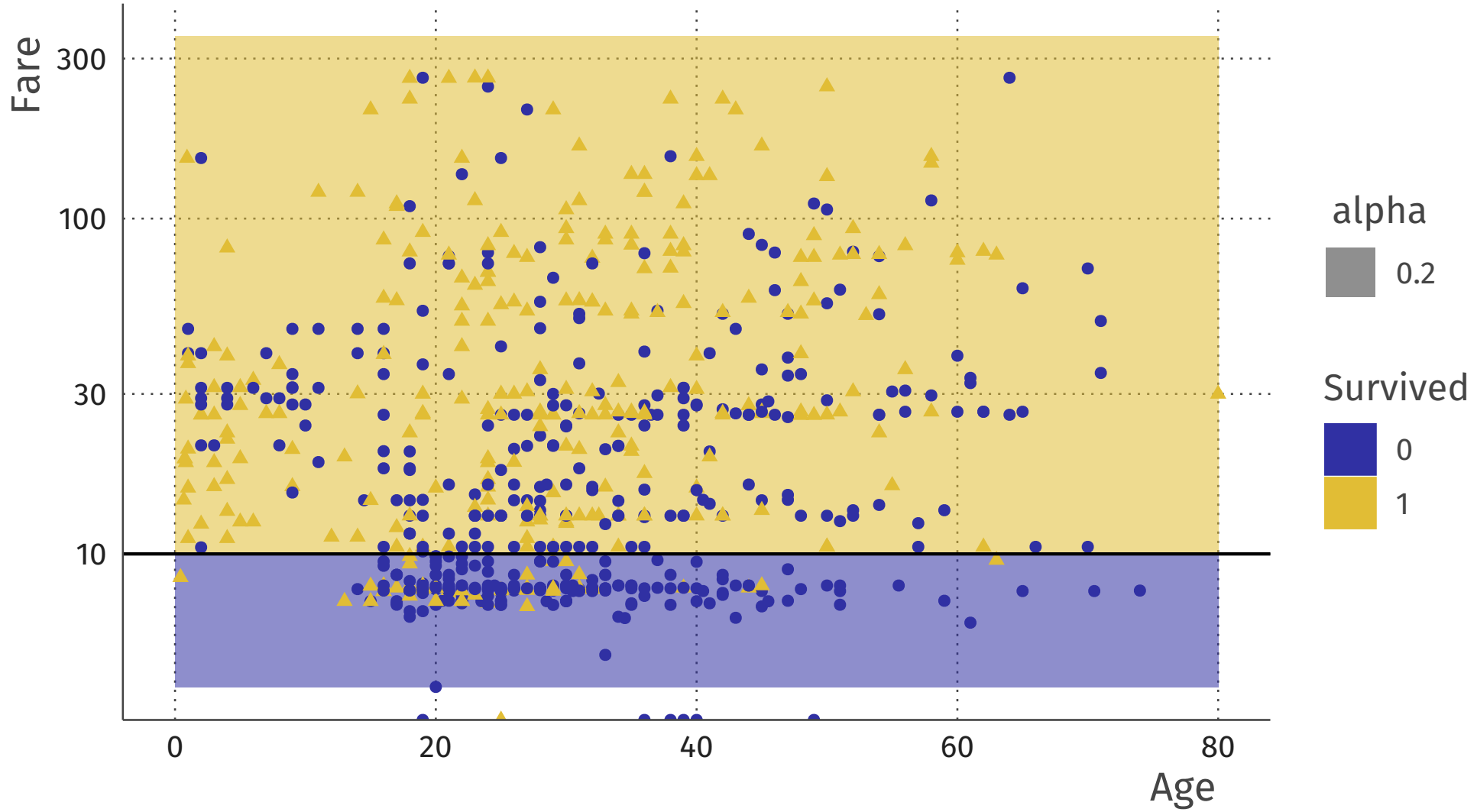
Recursive partitioning

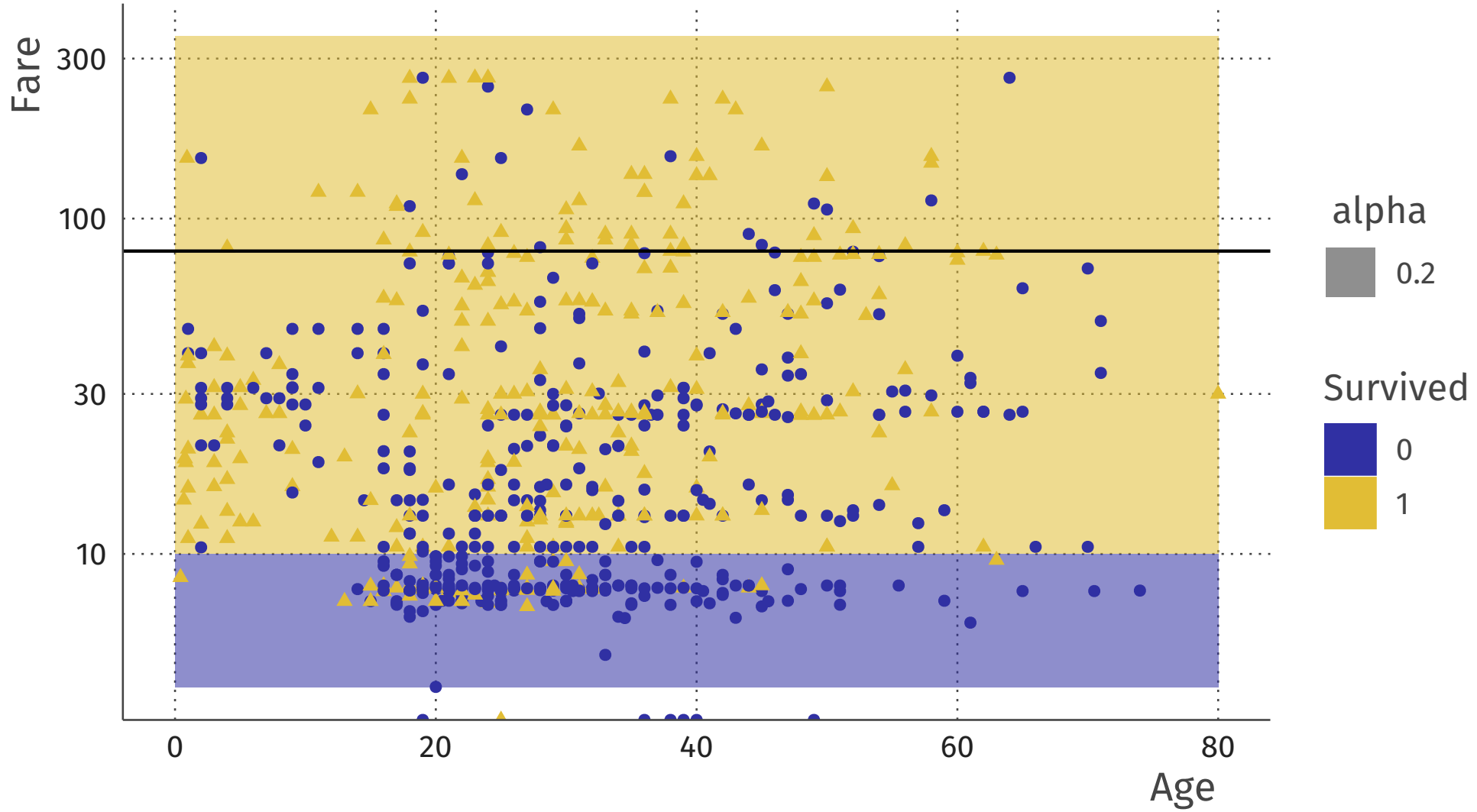
1. Find the split that makes observations as similar as possible on the outcome within that split;
2. Within each resulting group, do (1).

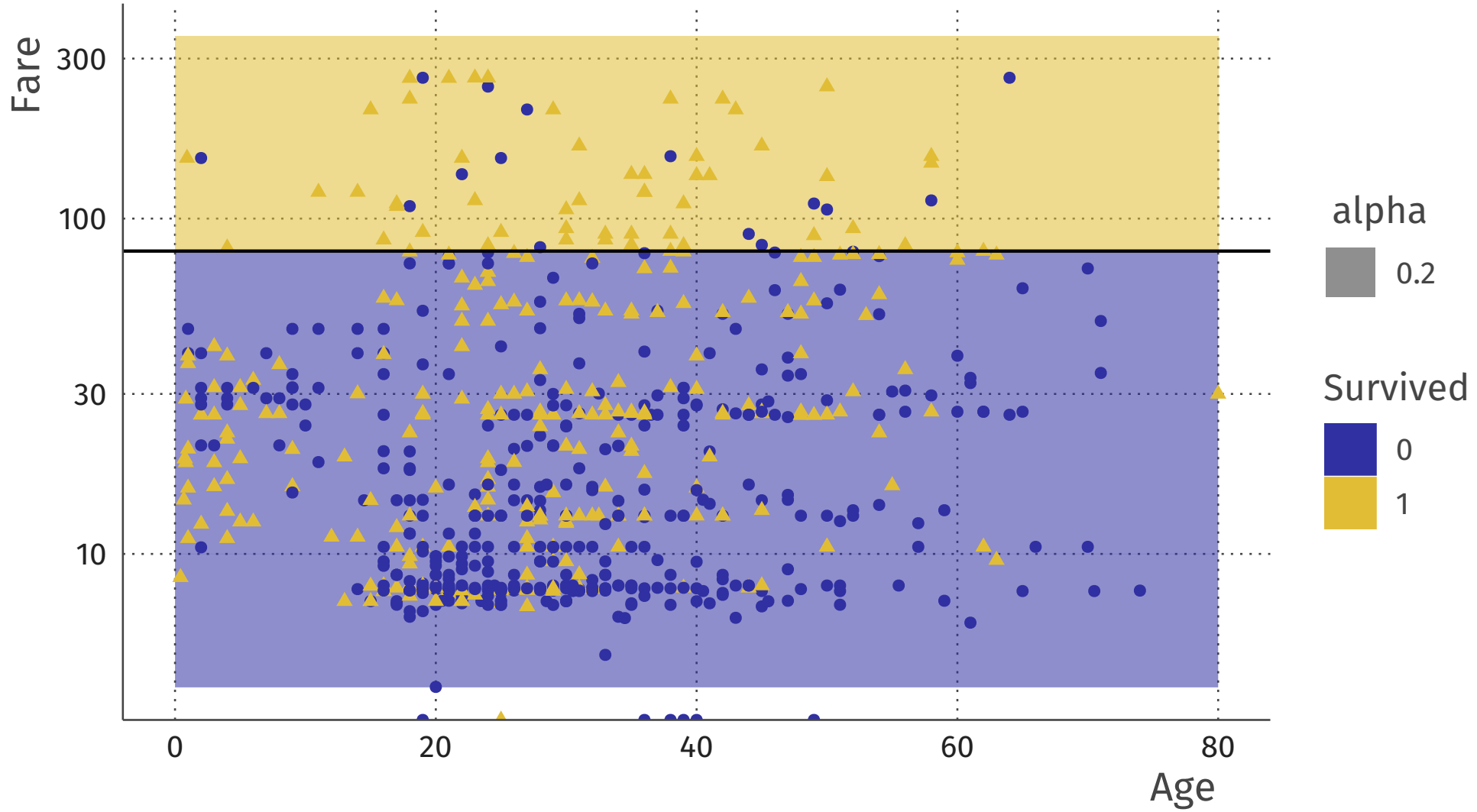
- **Criteria** for “as similar as possible”: Purity, MSE reduction, ...
- **Early stopping**: add after (2):
  - “unless fewer than `nmin` observations in the group” (typically 10);
  - “unless improvement less than `cp`” (typically 0.05);

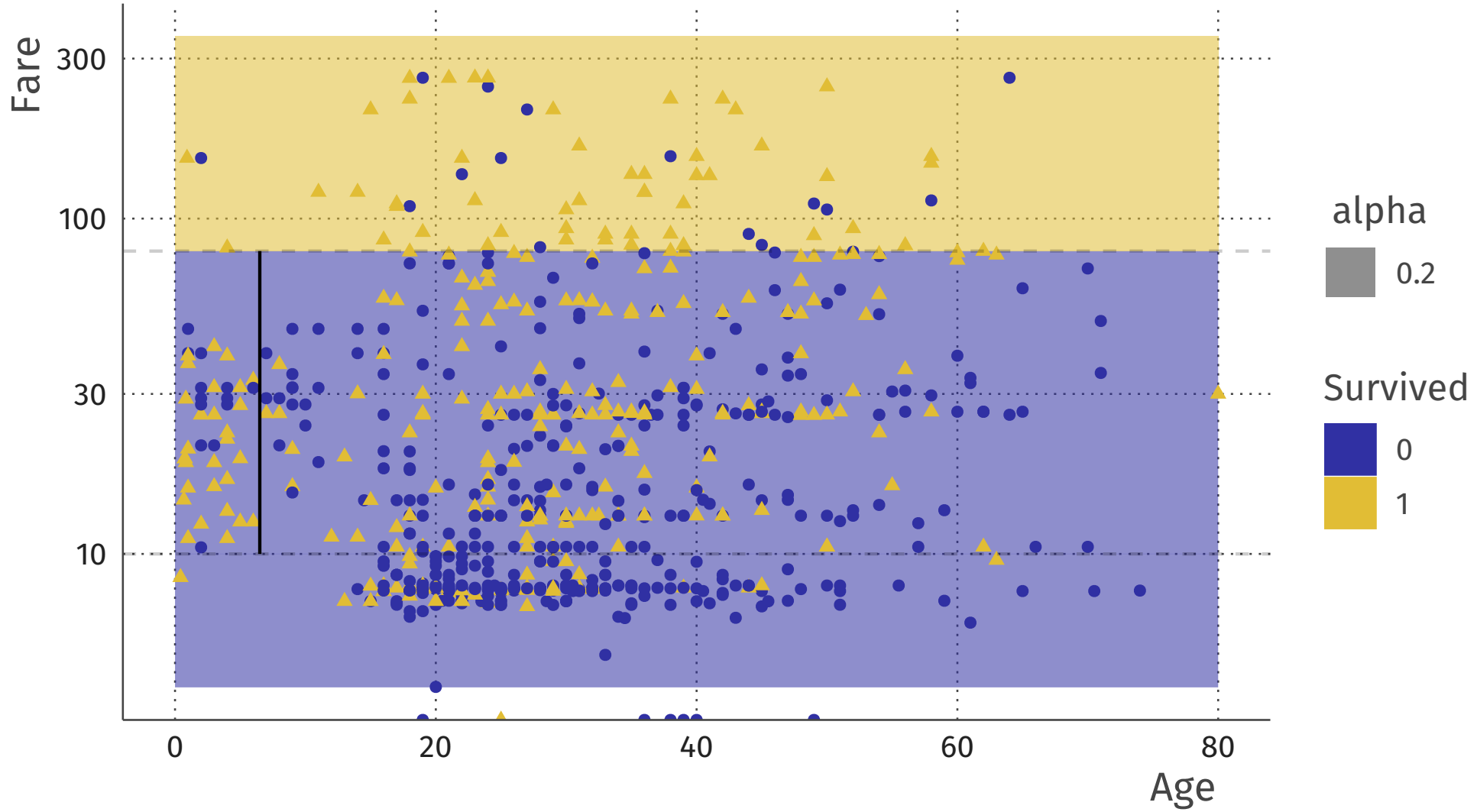

Fare
300
100
30
10
0
20
40
60
80
Age
Survived
0
1

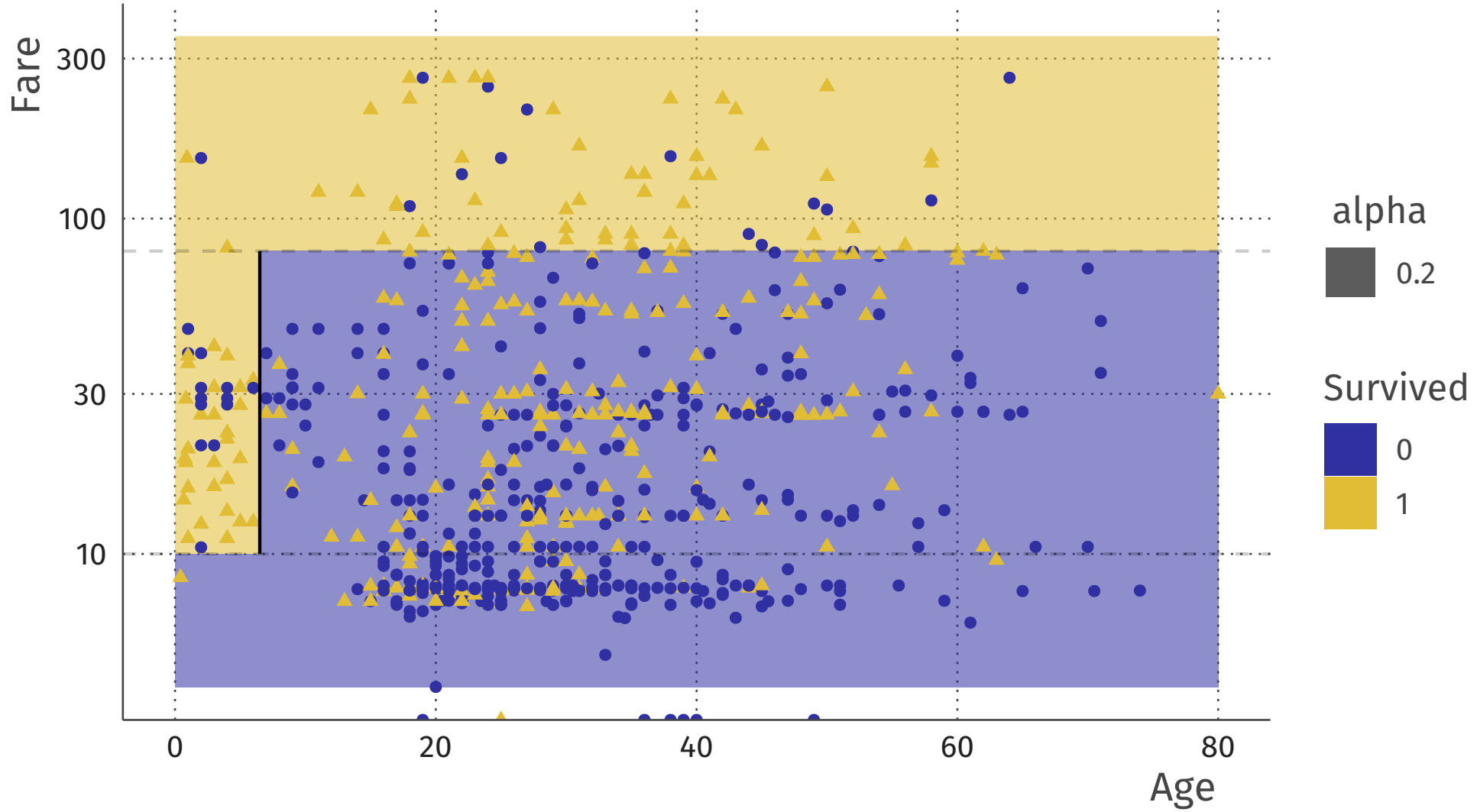

Fare
300
100
30
10
0
20
40
60
80
Age
Survived
0
1


Fare
300
100
30
10
0
20
40
60
80
Age
alpha
0.2
Survived
0
1

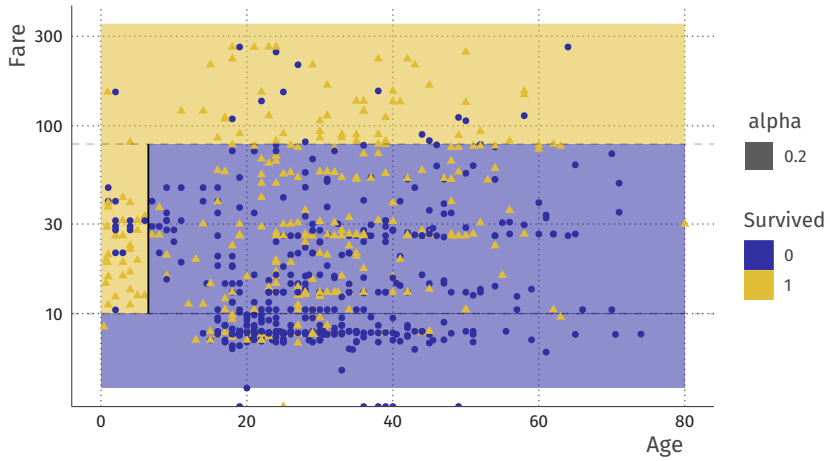

Fare
300
100
30
10
0
20
40
60
80
Age
alpha
0.2
Survived
0
1

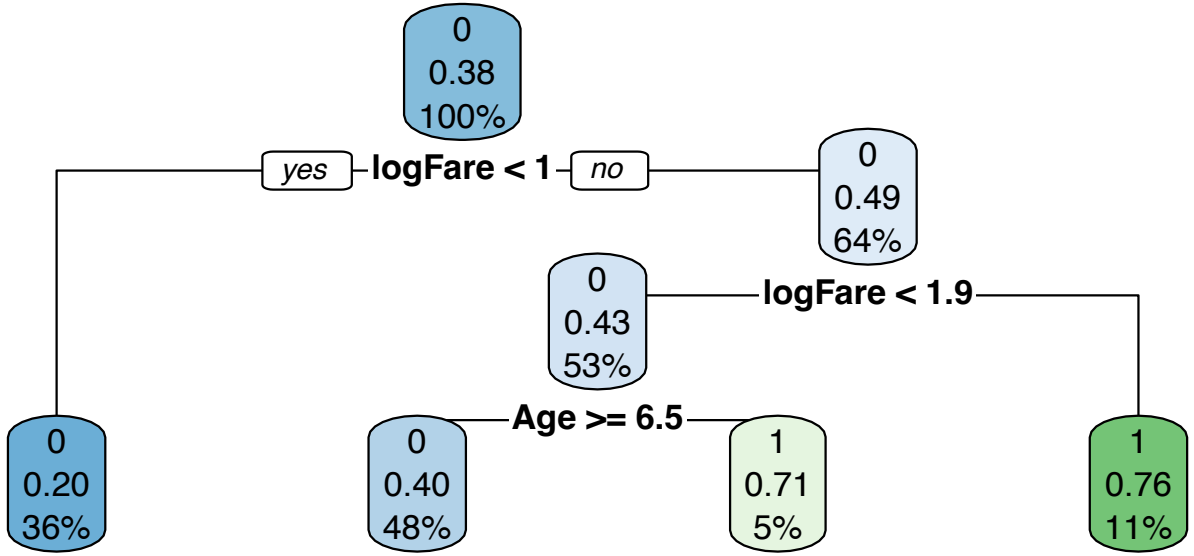

Fare
300
100
30
10
0
20
40
60
80
Age
alpha
0.2
Survived
0
1


Fare
300
100
30
10
0
20
40
60
80
Age
alpha
0.2
Survived
0
1


Fare
300
100
30
10
0
20
40
60
80
Age
alpha
0.2
Survived
0
1

These are the same!





Ensemble learners based on trees

ensemble methods use multiple learning algorithms to obtain better predictive performance (Wikipedia)

The problem with trees

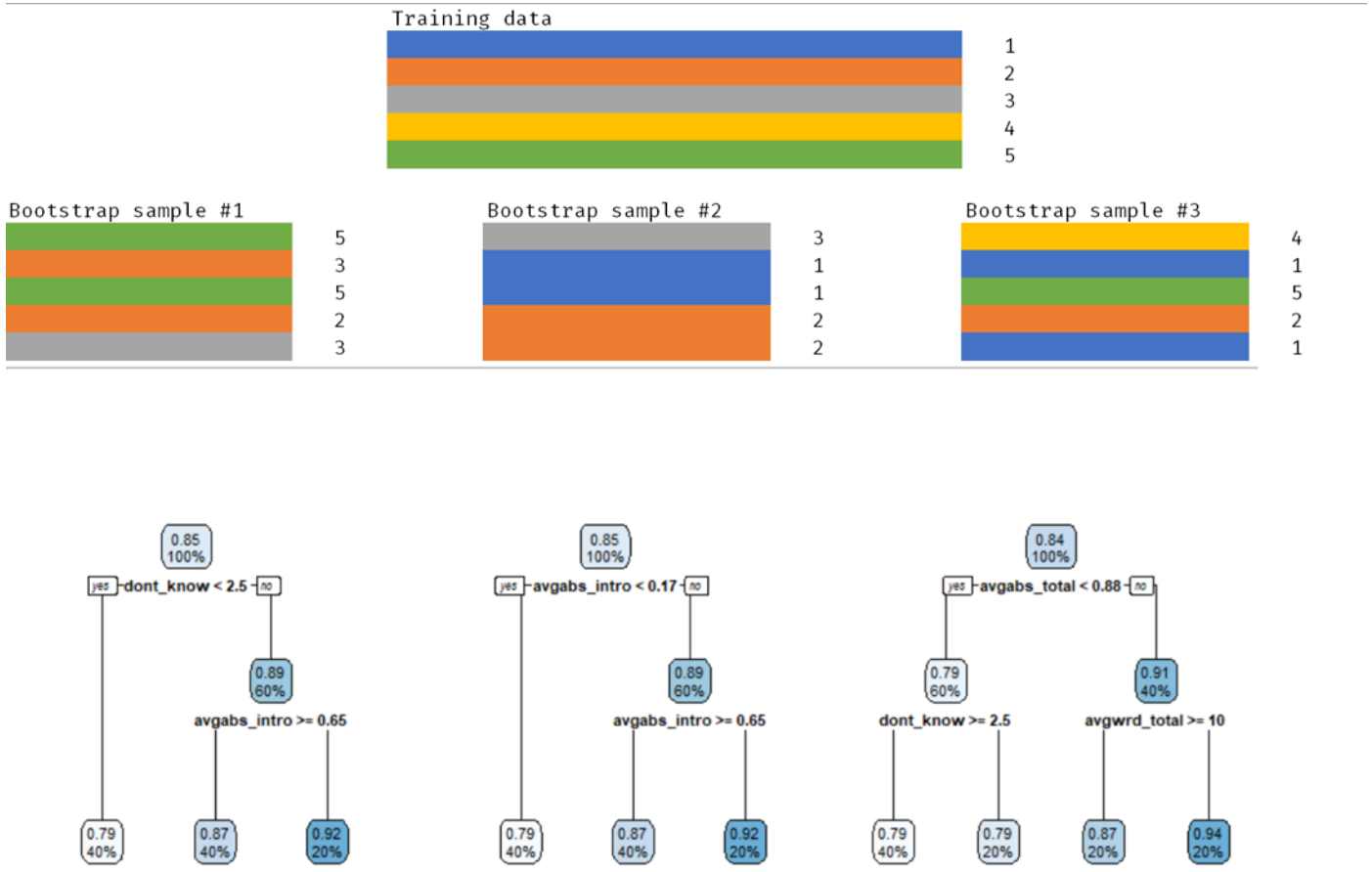
- Trees are not a bad idea, but in practice they tend to overfit

→ Use them as **basic building block** for **ensembles**

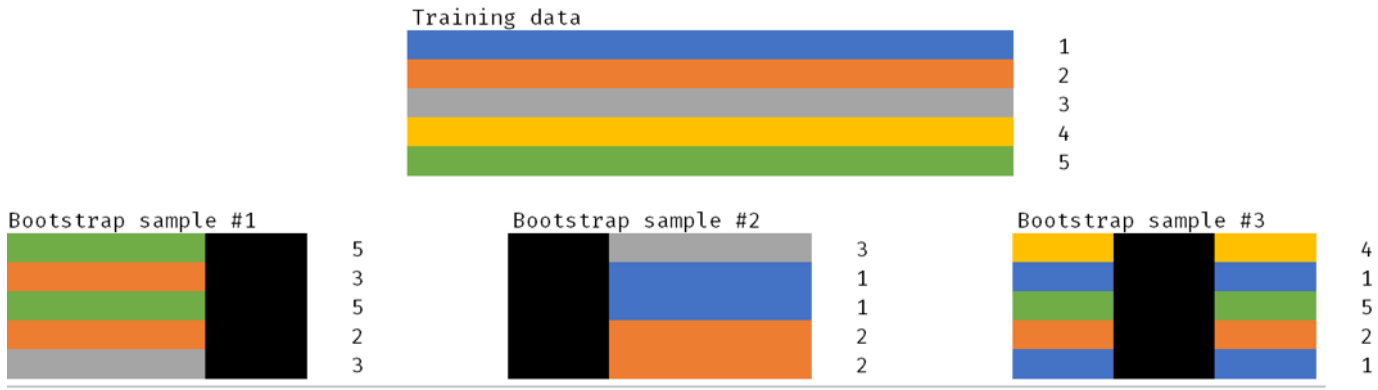
- **Random forests**: “bagged trees with feature sampling”
  - Make trees that are too complex (low bias, high variance);
  - Average over bootstrapped samples to cancel out the overfitting parts.

- **Boosting**: “ensemble of weak learners”
  - Make trees that are too simple (high bias, low variance);
  - Make more of them for observations with big residuals;
  - Average them.

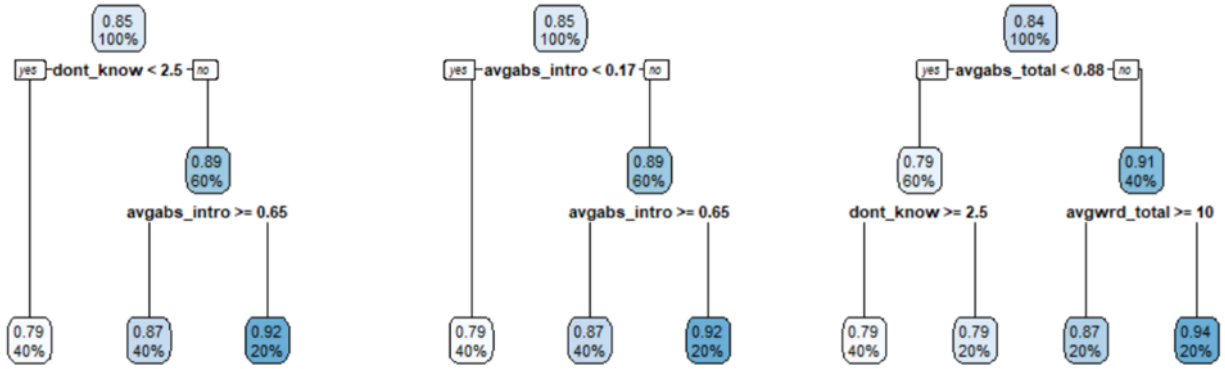
“Bagged” trees



“Bagged” trees with feature sampling



Note: features are sampled at each *node*



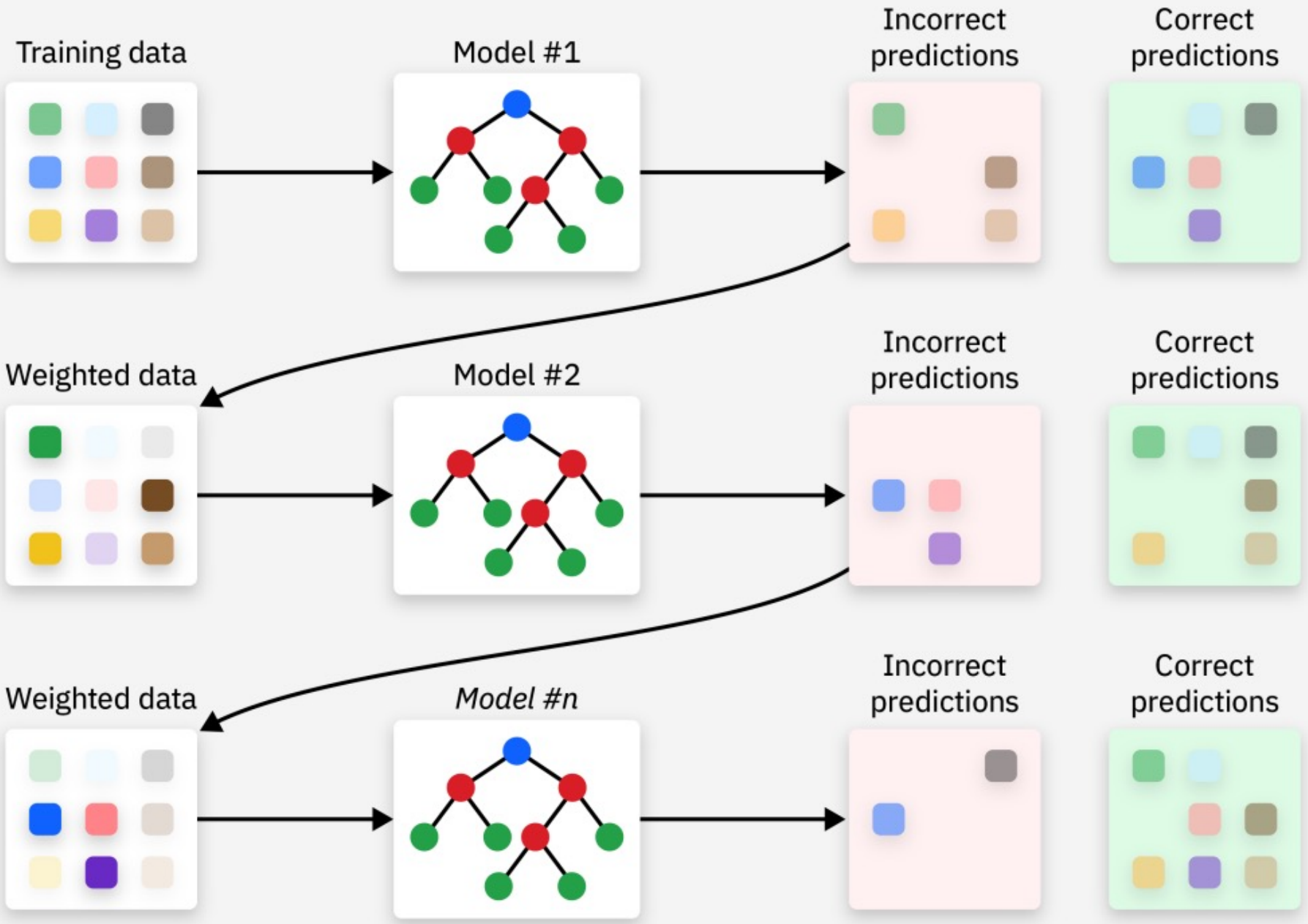
Boosting

Start with “**weak learner**” (e.g. tree stump)

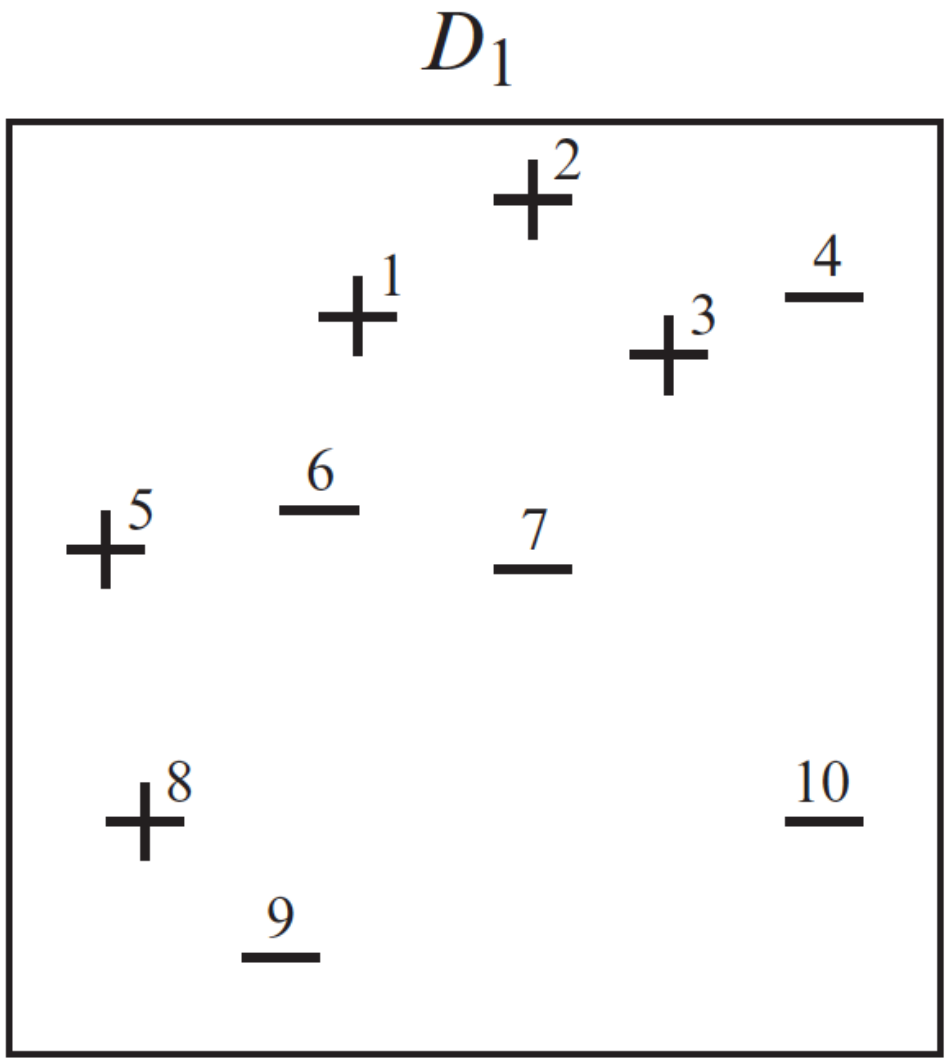
Repeat for `nrounds` “rounds”:

1. ↑ Upweight the **mistakes**, ↓ Downweight the **correct** classifications
2. Train another **weak learner** on the mistake-upweighted data
3. Back to step 1

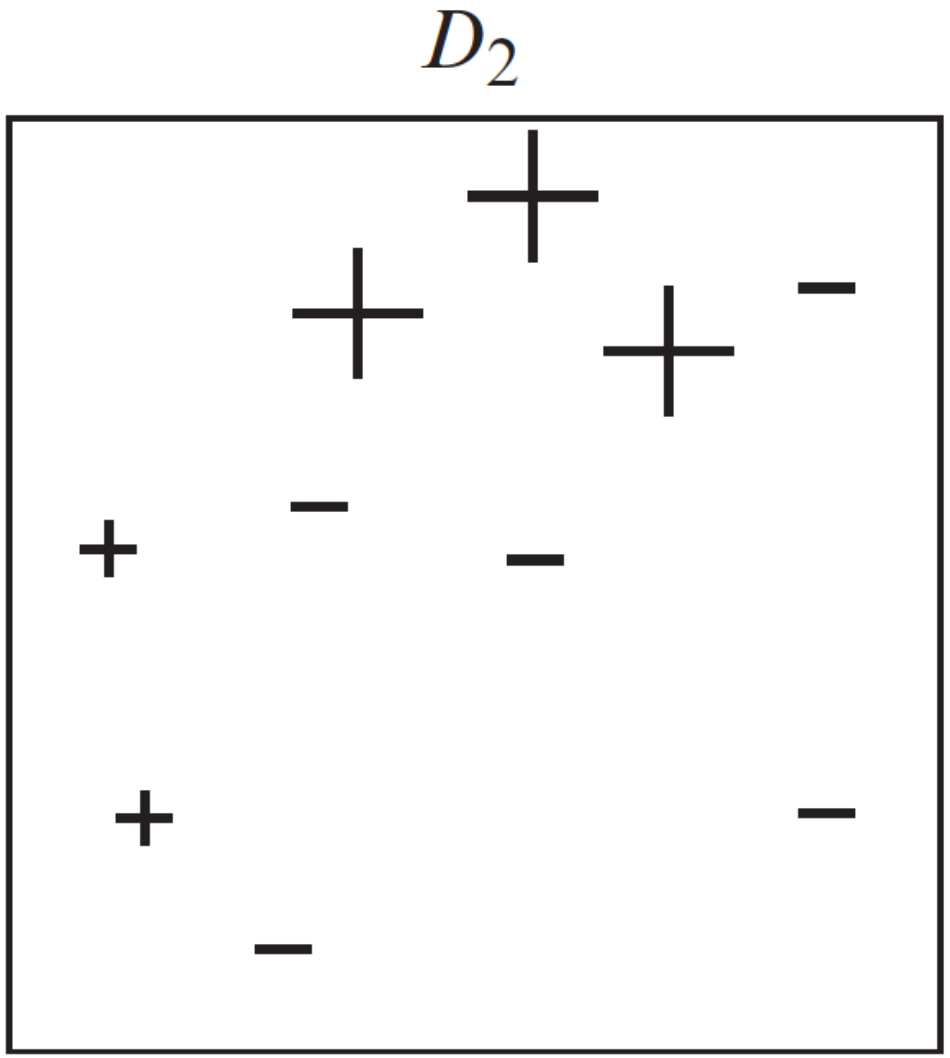
- Final prediction is **weighted sum of the weak learners**
- By combining many “weak learners”, a **strong learner** is created



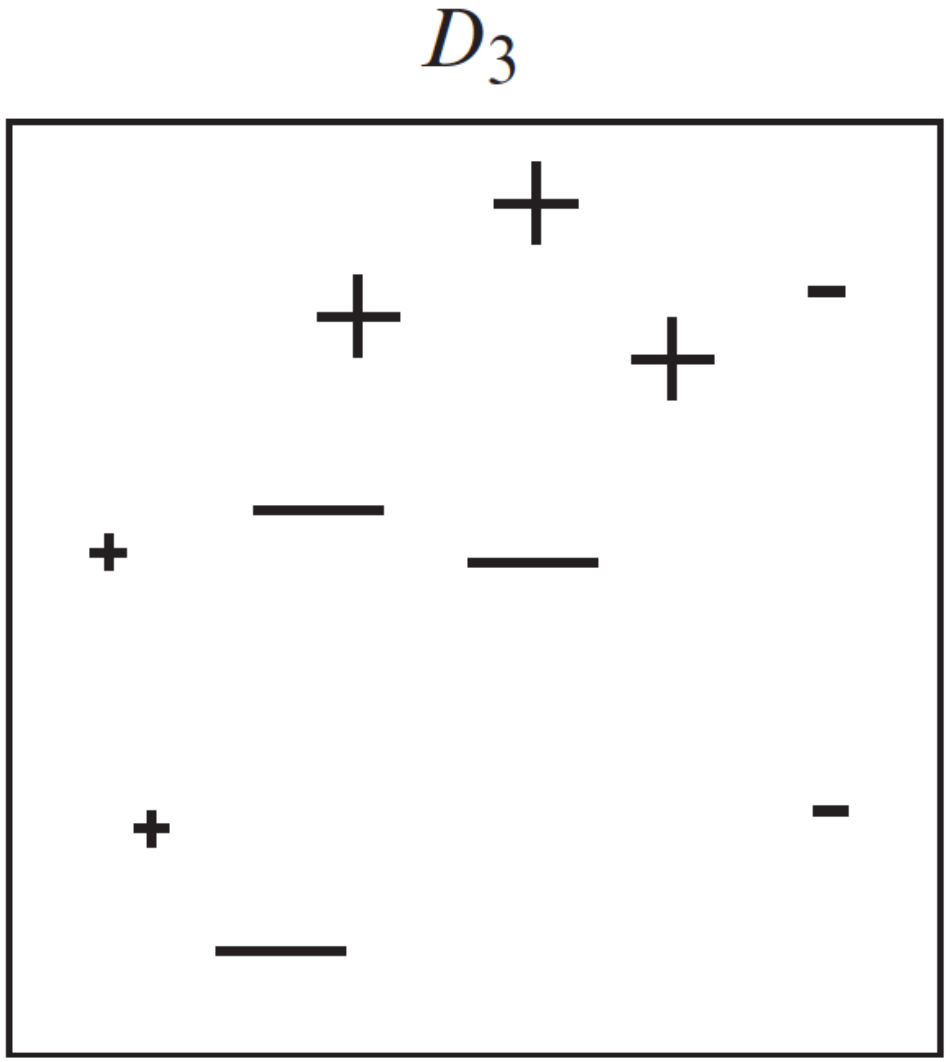
Source: `https://www.ibm.com/think/topics/gradient-boosting`



Source: Schapire & Freund (2012). *Boosting: Foundations and Algorithms.*



Source: Schapire & Freund (2012). *Boosting: Foundations and Algorithms.*



Source: Schapire & Freund (2012). *Boosting: Foundations and Algorithms.*

The final prediction model (classifier)

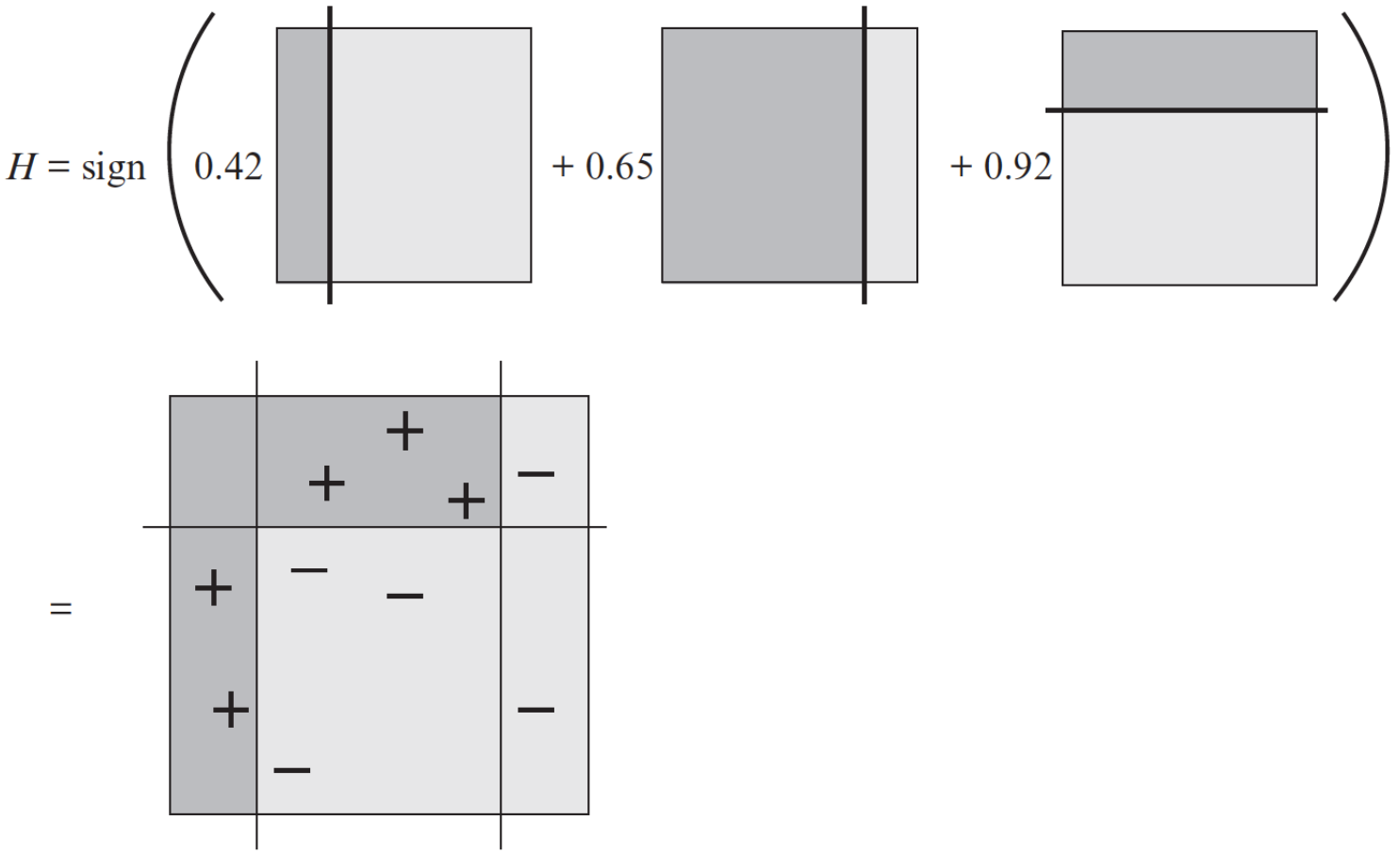
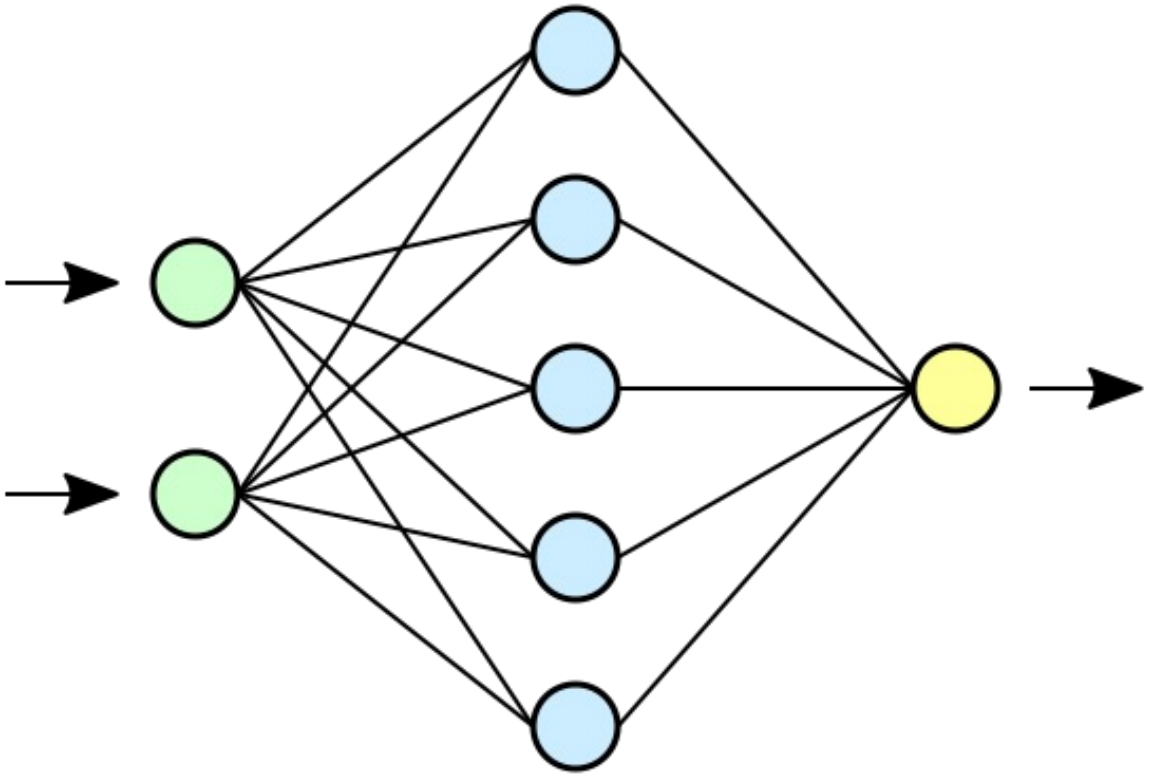


Figure 1.2
The combined classifier for the toy example of figure 1.1 is computed as the sign of the weighted sum of the three weak hypotheses, $\alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$, as shown at the top. This is equivalent to the classifier shown at the bottom. (As in figure 1.1, the regions that a classifier predicts positive are indicated using darker shading.)

Boosting

- Current go-to implementation is `xgboost`
- Very powerful idea and easy to apply to other things than classification trees
- Regression boosting, Survival boosting, etc.
- Often SotA in tabular data challenges, …
- … *after* extensive hyperparameter tuning

Neural network (topic of tomorrow)
Deep learning



"Hidden" nodes:

Example:

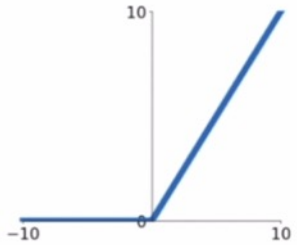
$$h_1 = f(w_{11}x_1 + w_{12}x_2)$$

Output:

$$y = f(w_{21}h_1 + w_{22}h_2 + \cdots + w_{25}h_5)$$

"Activation function":

- RElu $f(z) =$
- …



Evaluating classifiers

Confusion matrix: Counts

```
> p_pred <- predict(titanic_tree, newdata = val_df)

> with(val_df, table(p_pred > 0.5, Survived))

       Survived
          0   1
  FALSE 134  40
  TRUE   19  75
```

Confusion matrix: counts

	Survived (observed)			
	No		Yes	
Survived (predicted)				
No	134	(TN)	40	(FN)
Yes	19	(FP)	75	(TP)

- False positives (FP): 19
- False negatives (FN): 40
- Total errors: FP + FN

Confusion matrix:
Sensitivity ("recall") and Specificity

```
> with(val_df, table(p_pred > 0.5, Survived)) %>% prop.table(2)
```

	Survived (observed)	
	No	Yes
Survived (predicted)		
No	0.876	0.348
Yes	0.124	0.652
TOTAL	1	1

- Specificity: $\frac{\text{TN}}{\text{TN}+\text{FP}}$ = 134 / (134 + 19) ≈ 0.876
- Sensitivity ("recall"): $\frac{\text{TP}}{\text{TP}+\text{FN}}$ = 75 / (75 + 40) ≈ 0.652
- Accuracy (ACC): $\frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}} \approx 0.780$
- Error rate: $1 - \text{Accuracy} \approx 0.220$

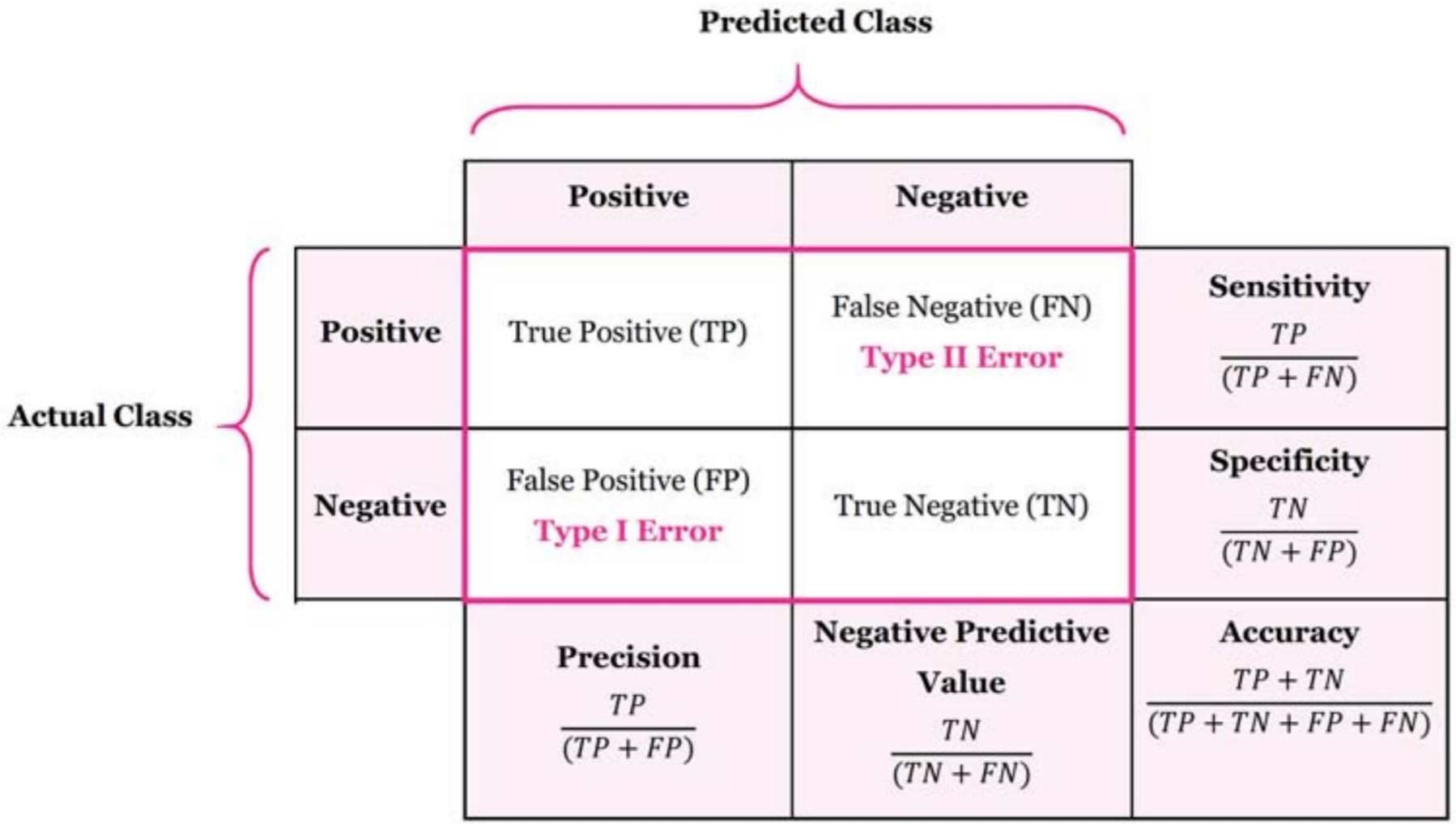
Confusion matrix:
Negative & positive predictive value

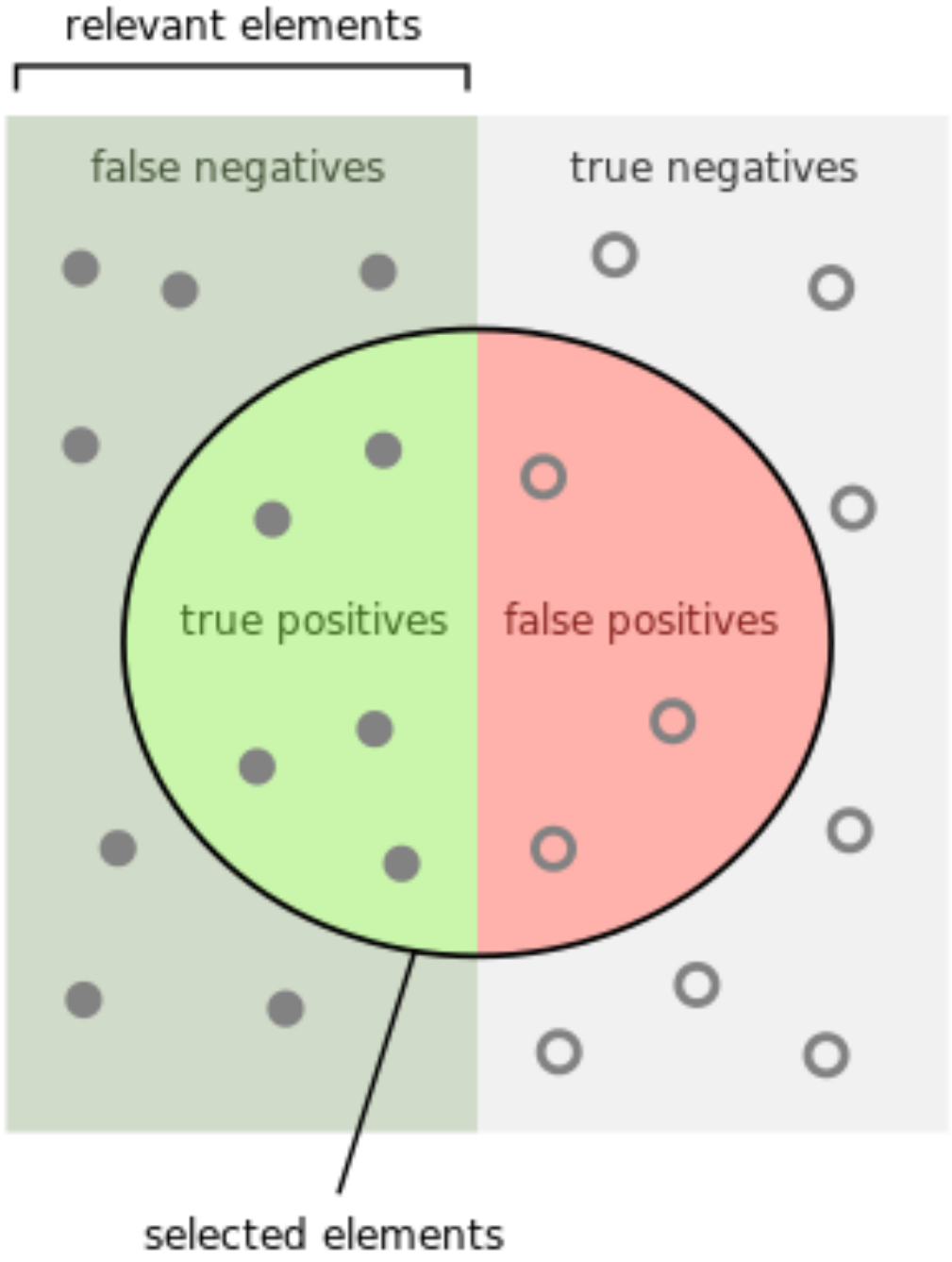
```
> with(val_df, table(p_pred > 0.5, Survived)) %>% prop.table(1)
```

	Survived (observed) No	Yes	TOTAL
Survived (predicted)			
No	0.770	0.230	1
Yes	0.202	0.798	1

- NPV: $\frac{\text{TN}}{\text{TN}+\text{FN}}$ = 134 / (134 + 40) ≈ 0.770
- PPV ("precision"): $\frac{\text{TP}}{\text{TP}+\text{FP}}$ = 75 / (75 + 19) ≈ 0.798

Confusion matrix





How many selected items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{green + red}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{green}}{\text{green + olive}}$$

Source: https://en.wikipedia.org/wiki/F-score

F_1 score

The **F_1 score** is the "harmonic mean" of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

- **Like** accuracy, the F_1 quantifies overall amount of error
- **Unlike** accuracy, F_1 is not as affected by uneven class distributions

Titanic example: $F_1 \approx \frac{2}{1/0.798 + 1/0.652} = 0.718$

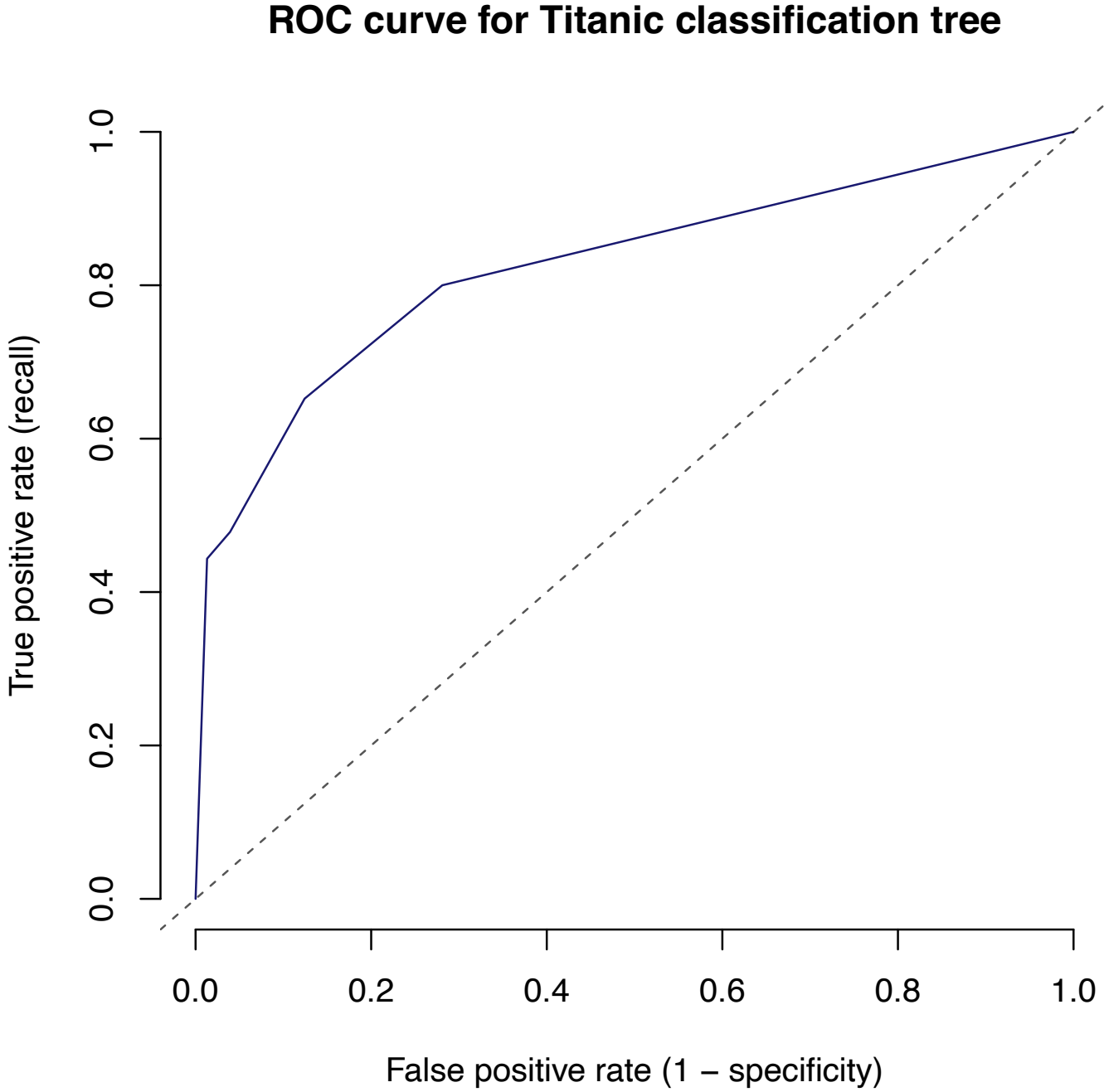
Overview: some classification metrics

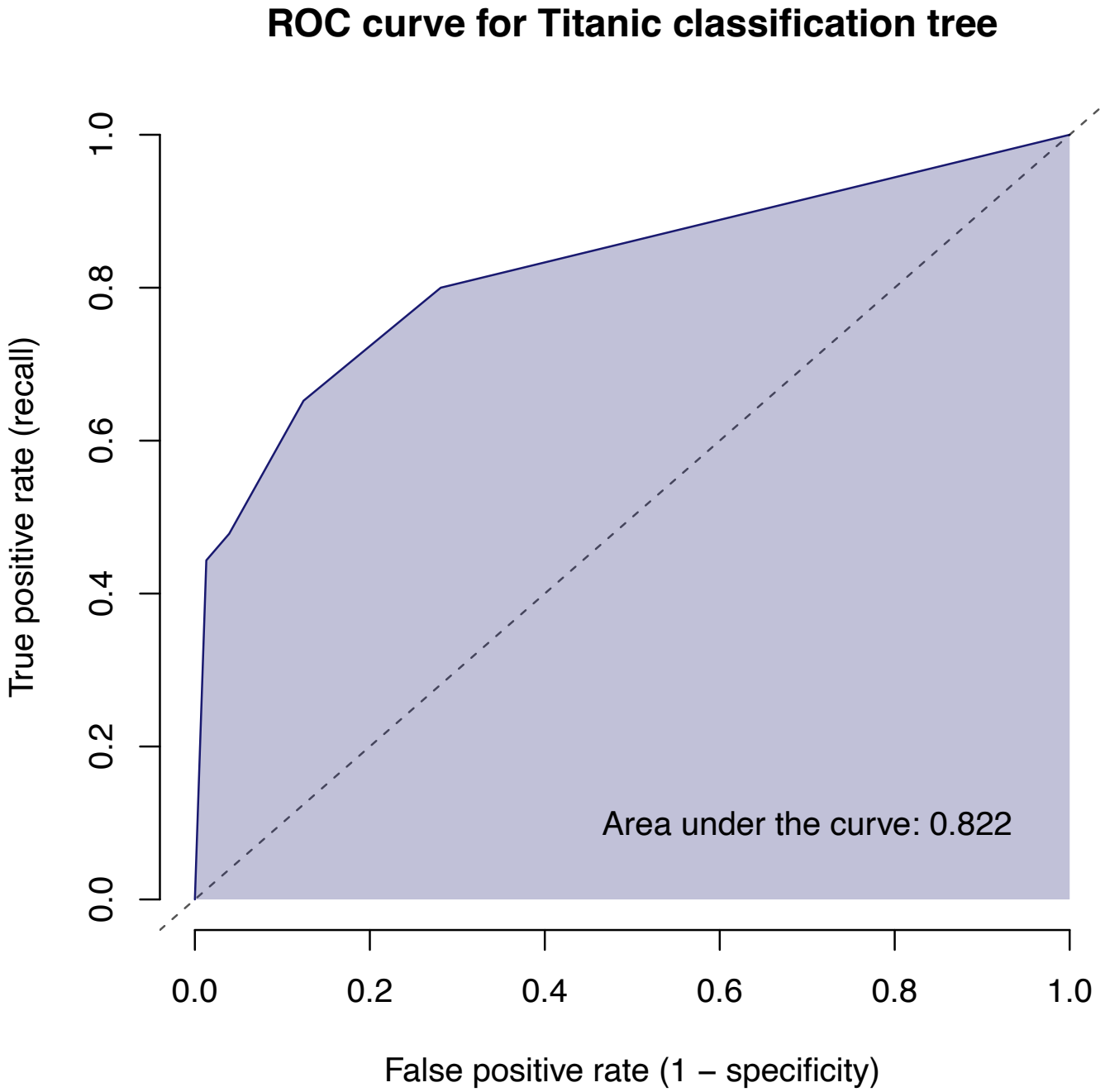
- Sensitivity (=Recall)
- Specificity
- Positive predictive value (=Precision)
- Negative predictive value
- Accuracy
- Many! more: https://en.wikipedia.org/wiki/Sensitivity_and_specificity

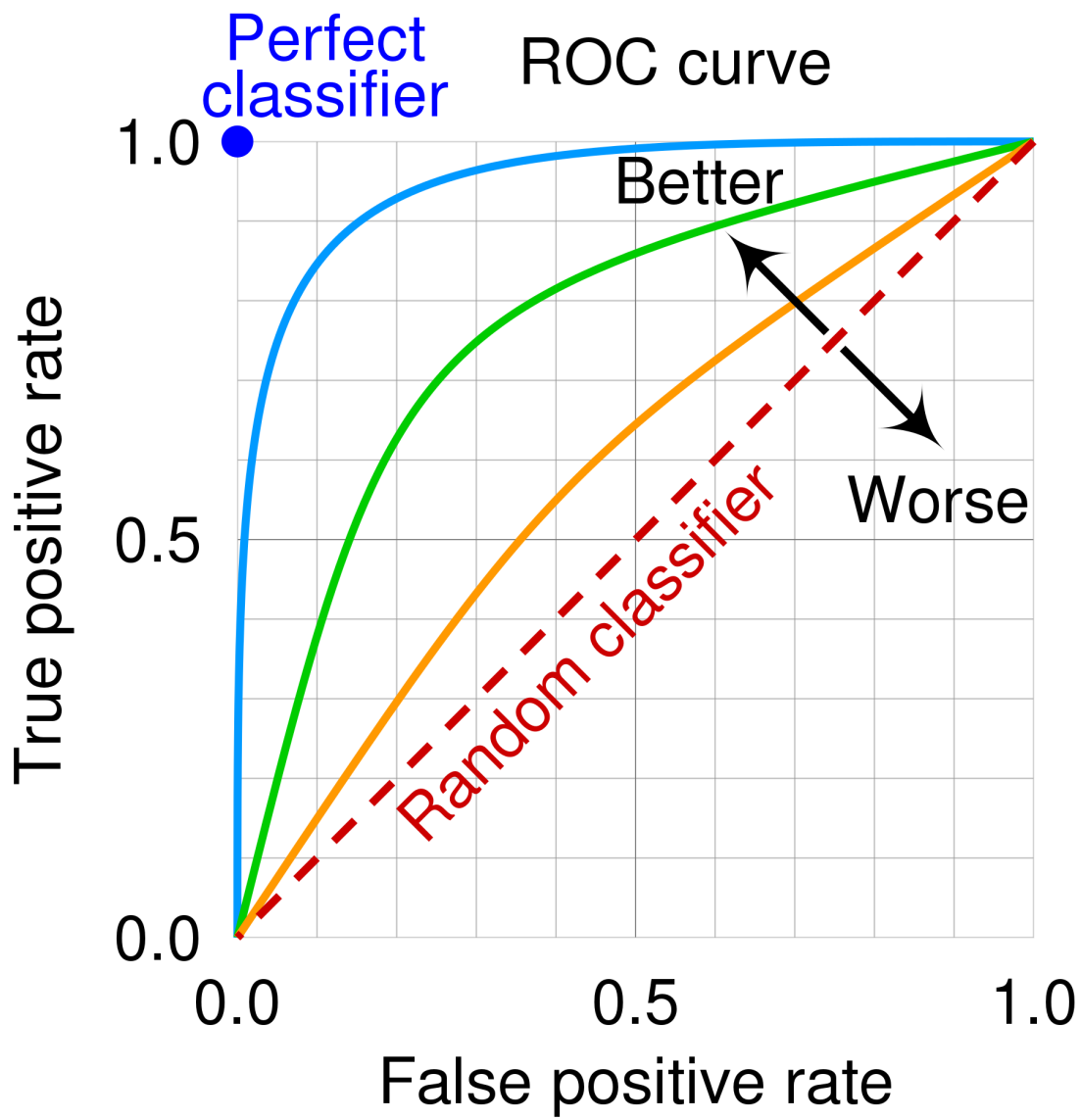
Different thresholds than 0.5

Moving around the threshold affects sensitivity and specificity!

```
> with(val_df, table(p_pred > 0.4, Survived)) %>% prop.table(2)

       Survived
            0     1
  FALSE 0.876 0.348
  TRUE  0.124 0.652

> with(val_df, table(p_pred > 0.6, Survived)) %>% prop.table(2)

       Survived
            0     1
  FALSE 0.961 0.522
  TRUE  0.039 0.478
```


ROC curve for Titanic classification tree
True positive rate (recall)
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate (1 – specificity)
0.0
0.2
0.4
0.6
0.8
1.0


ROC curve for Titanic classification tree
True positive rate (recall)
False positive rate (1 – specificity)
0.0
0.2
0.4
0.6
0.8
1.0
Area under the curve: 0.822


Perfect classifier
ROC curve
1.0
Better
Worse
0.5
Random classifier
True positive rate
0.0
0.0
0.5
1.0
False positive rate

Calibration

- Besides the quality of a single-shot predicted class (“yes/no”, “survive/not survive”, ...),
- We could also be interested in the **predicted probability.**
- E.g.: “casemix adjustment” for hospital/school evaluation, risk scores in medicine, betting, ...

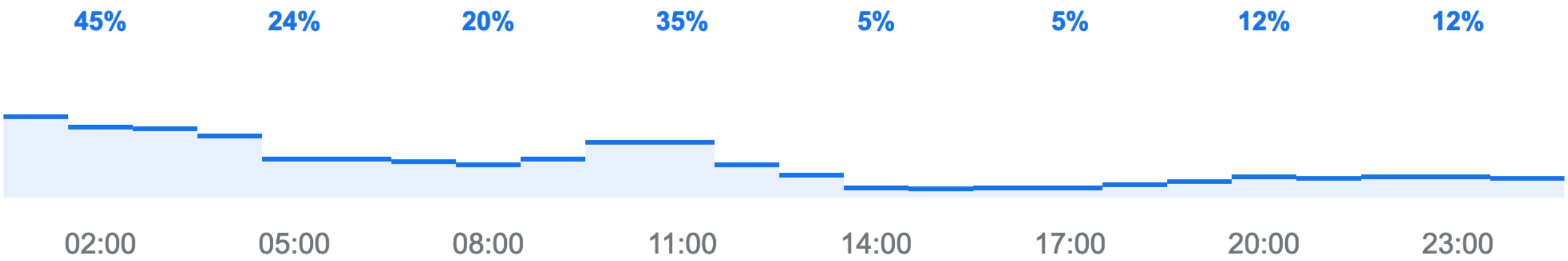
On days like today, how often does it rain?

17 °C | °F

Precipitation: 40%
Humidity: 80%
Wind: 24 km/h

Utrecht, Netherlands
Thursday
Scattered showers

Temperature | Precipitation | Wind



Probability

A *probability* is a number *p* such that the proportion of events given that number is about *p*.

- **Ideally**, the classification procedure (e.g. classification tree) outputs a predicted probability directly.
- **Unfortunately**,
  - Not all classifiers output a predicted probability (e.g. SVM);
  - Many classifiers that do give a number between 0 and 1 called a "predicted probability", the predicted probability does not give the correct proportion of events.

This is the "**calibration problem**".

Calibration plot

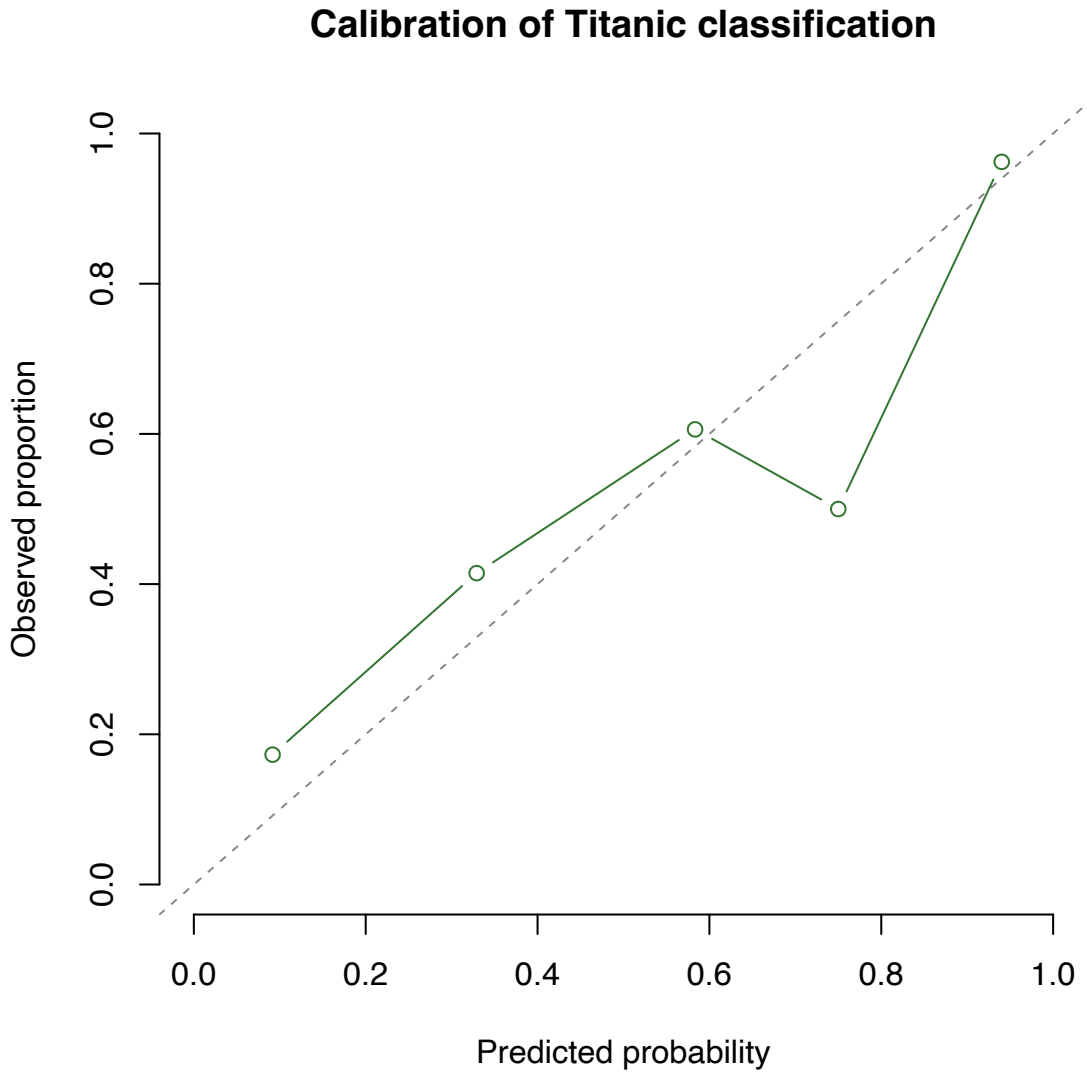
Probability

A *probability* is a number *p* such that the proportion of events given that number is about *p*.

- A predicted probability is **calibrated** when it conforms to the definition above;
- Check this using a **calibration plot**

- When the model says there's a 60% chance of survival, that should mean that about 60% of people with that same prediction actually survived.

- If the points lie on the diagonal, our model's probabilities are well calibrated. If they fall below, the model is overconfident , it predicts higher probabilities than observed. If they lie above, it's underconfident.



Post-hoc probability calibration

- Some libraries allow you to tweak the predicted probabilities so they fit on the curve. This is called “probability calibration”.
- There are many methods, but the most commonly used one takes a classification model we know is calibrated (“logistic regression”) and applies it to the uncalibrated scores outputted by the classifier;
- You may encounter this in your readings.

MSE (“Brier score”)

- We can also measure calibration numerically with the **Brier score:** it’s the mean squared error between predicted probabilities and actual outcomes (0 or 1).

$$\mathrm{MSE} = \mathrm{average}\left((\hat{p} - y)^2\right)$$

- MSE can be reworked into two terms:

MSE = Calibration term + AUC term

Calibration in evaluation of ML models

- Calibration answers the question: when my model predicts a probability, can I trust that number?
- Important for downstream decision-making
- Sometimes overlooked in ML model evaluation
- Many (most?) high-capacity models are not calibrated by default
- Evaluate using more advanced methods in standard software

Class imbalance

Problem:

- Measures such as SEN/SPE/ACC/F1 emphasize larger classes;
- What if the *smaller* classes are the most/equally interesting?

Some **solutions**:

- Oversampling minority/undersampling majority
- Weighting
- “Embedded”: class_weight='balanced’ in boosting

Conclusion: Supervised learning is a large field

- We have emphasized
  - Pluralistic approach, **coupled with**
  - Honest model evaluation
- This includes thinking about the task, performance metric, and training and testing data
- Some of these lead to specific issues in classification (e.g. calibration, class imbalance) discussed today (& in book)
- **You should now be equipped to start using supervised learning in practice!**