# Data & Things

## (Spring 25)

Monday February 17

**Lecture 8: Classification II**

Jens Ulrik Hansen

Roskilde Universitet

# Outline of this lecture

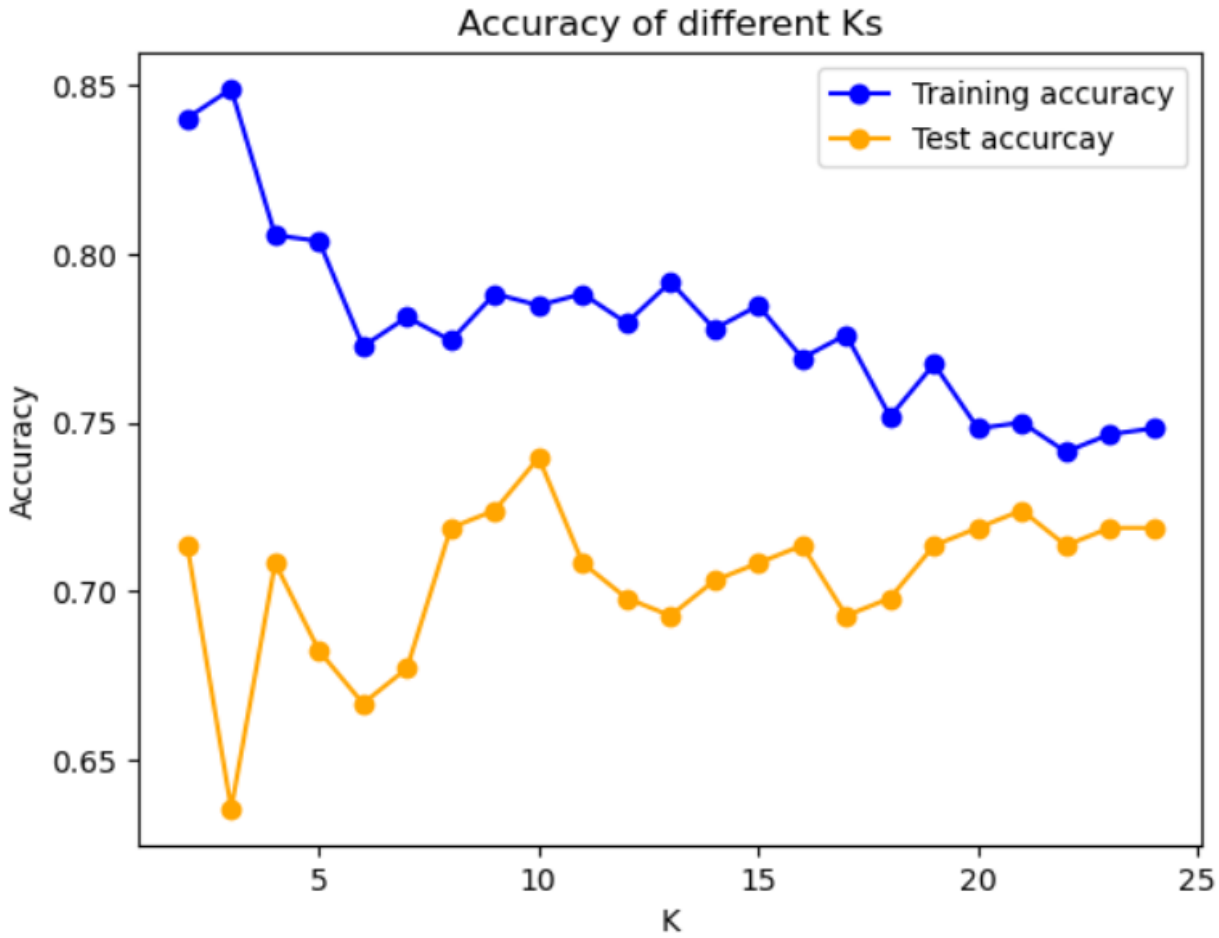
- Cross-validation
- Evaluating Classification models revisited
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting

# Cross-validation

- Our train-test split from last time, is often not enough!
  - We might want train multiple models and pick the best one, or iteratively adjust “hyper-parameters” of our models
    - if use the performance on the test set to compare different models, we might ***risk overfitting t to the test set***
    - Moreover, the performance measure on the test set is no longer a good estimate of how the model will perform on new data
  - If we have smaller datasets, or use many variables in our model, we are left with a dilemma
    - We need a large portion of the data for training, to be able to get a good model
    - We need a large portion of the data for testing, to get a good estimate of how we will perform on new unseen data
  - Finally, train-test split can be quite sensitive to the actual random split (as we saw for the KNN example!)

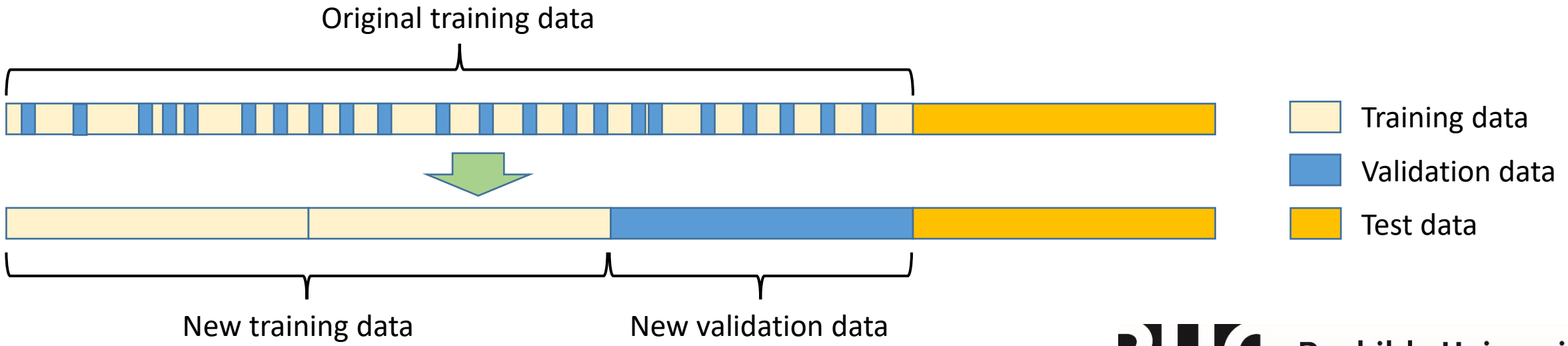
# Cross-validation

- Recall our KNN example on the diabetes example from last time…
  - We wanted to use the test accuracy to pick the right K
  - We had only 768 data points in total, so got few data points both for training and testing no matter how we did the split
  - Can we somehow utilize all the data for training and still get a good (unbiased) estimate of how well we will perform on future unseen data?



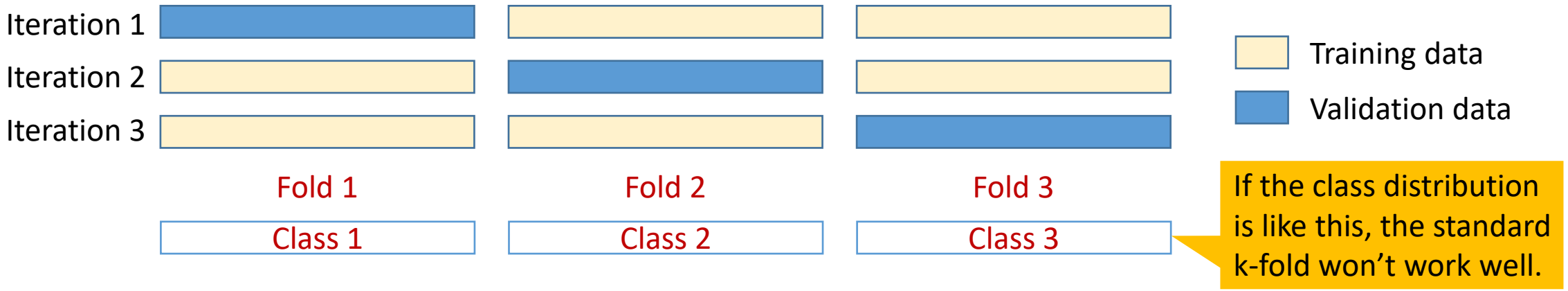
# Cross-validation

- **Train-Validation-Test split**
  - Use an additional validation set to evaluate and compare models, trying different models and setting to chose the best
    - More specific, we leave the test set as is and split the training set further into a new training set and a validation set.
    - We then train our models on the new training set
    - Evaluate, compare and chose final model based on performance on the new validation set
    - Finally, calculate an unbiased estimate of our model performance on new unseen data by calculating the performance on the test set
    - There is no one right proportion of splits, but an often sensible split is: 60% for training, 20% for validation, 20% for test



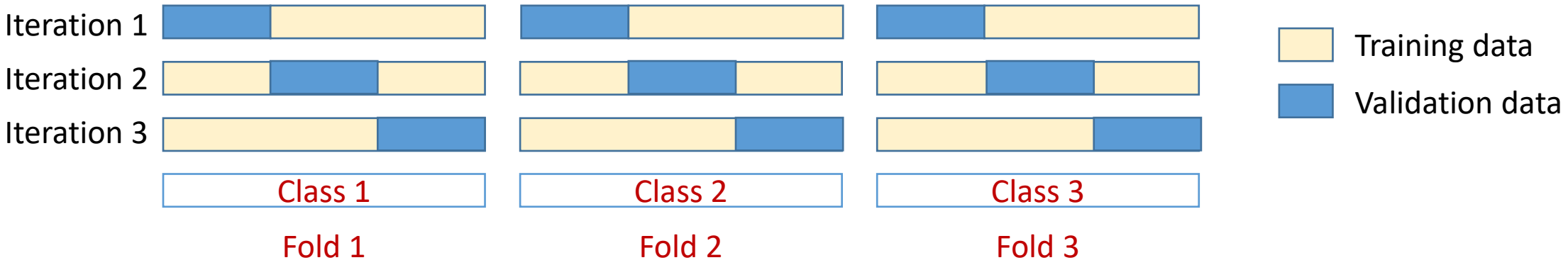
# Cross-validation

- ***k*-fold cross-validation**
  - (k = 5 or k = 10 are popular choices)
  - Split the (train and validation) data D into *k* mutually exclusive subsets, each of approximately equal size: $D_1 \ldots D_k$. Each $D_i$ is called a *fold*.
  - Do model construction and evaluation *k* times. Use the *average* accuracy.
    - At the *i*-th iteration, use fold $D_i$ as the validation set and the others as the training set.
- Example of standard 3-fold cross validation



# Cross-validation

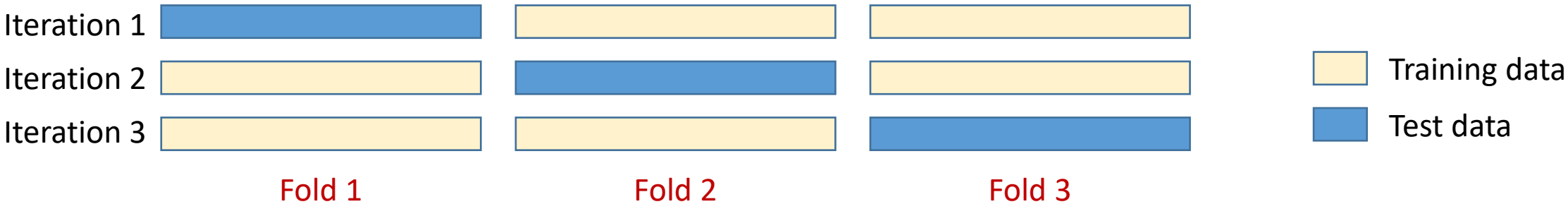
- **Stratified cross-validation**
  - folds are stratified so that *class distribution* in each fold is approximately the same as that in the initial given data.



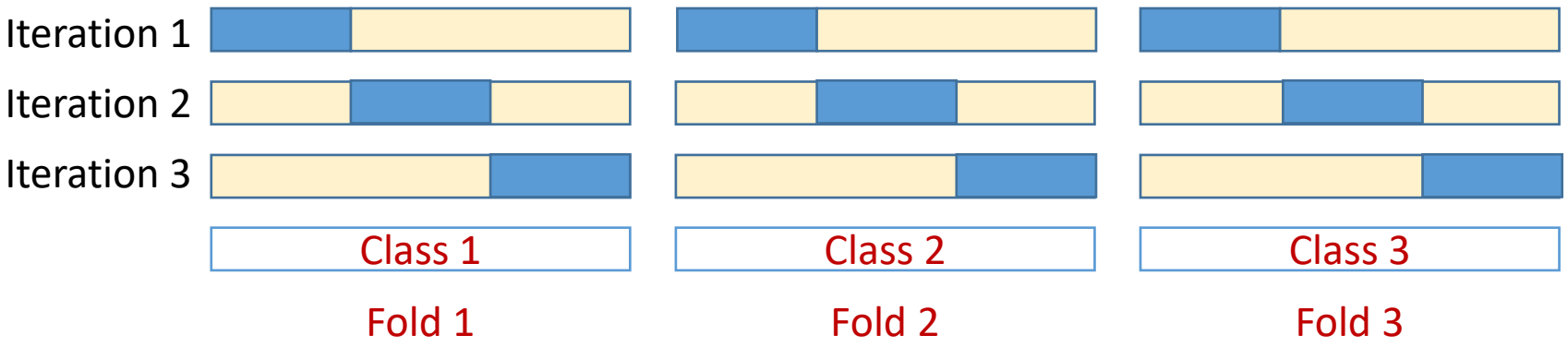

- **Leave-one-out cross-validation**
  - *k* is set to the number of data points/rows in the dataset
  - Use it only for small sized data; otherwise, too many models to construct!
  - Often 5- or 10-fold cross validation is just as good or even better

# Cross-validation

- Standard 3-fold cross validation




- Stratified 3-fold cross validation



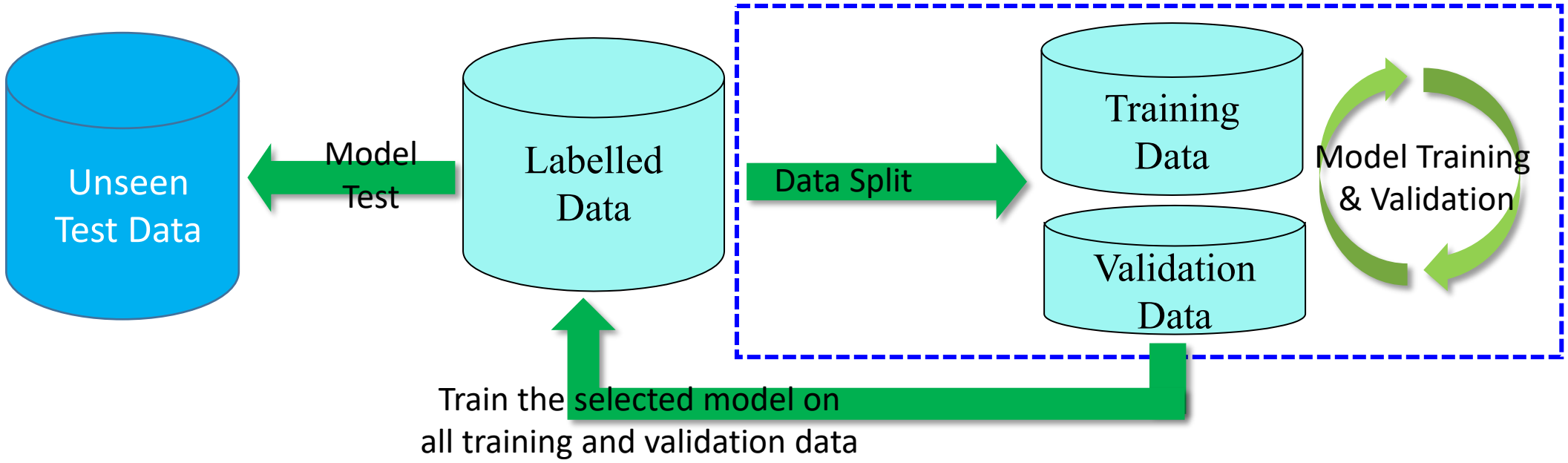

Default in scikit-learn:
Stratified **5**-fold

# Cross-validation

- Cross-validation is not a way to construct an applicable model.
  - The function cross_val_score(.) builds multiple models *internally*, but these models are not returned.
- The purpose of cross-validation is to evaluate how well a *type* of model will generalize when it is trained on a specific dataset.
  - Model type: decision tree, random forest, KNN, SVM, …
- Moreover, by using cross-validation, we can decide what type of model to use, and tune hyper-parameters for constructing a model
  - ***Hyper-parameters***: algorithm parameters that can be set by the user before training a model.
    - E.g., K for KNN, test_size and random_state for train_test_split(.), gini or entropy for a DT, …
  - In contrast, ***model parameters*** are learned internally from training data
    - E.g., a and b coefficients in simple linear regression or logistic regression, weights in a neural network, …

# Cross-validation

- ***After having used cross-validation to decide model type and tune hyper-parameters, we train the final model one last time on all the (training and validation) data (before evaluating it on the untouched test data)***

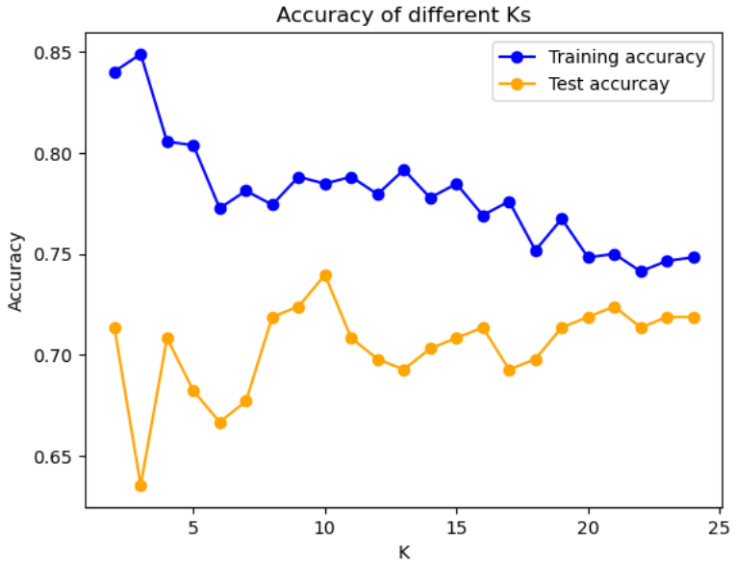


# Outline of this lecture

- Cross-validation
- Evaluating Classification models revisited
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting

# Evaluating Classification models revisited

- For classification models, we evaluated them by:
  - Accuracy: the fraction of correct predictions
  - Error rate: the fraction of incorrect predictions
- However, not all errors are equally important:
  - In medical diagnostics (1 = having the disease, 0 = not having it) two type of errors:
    - Predicted 1, but truly 0: Bad in the sense that it is not nice to be diagnosed with a disease, but later tests will likely be taken revealing that the patient did not have the disease after all.
    - Predicted 0, but truly 1: Much worse, the patient is led to believe she does not have the disease, and no further tests are made, which might lead to severe health conditions later.
  - In spam email detection (1 = it is spam, 0 = it is not spam) two types errors:
    - Predicted 1, but truly 0: Problematic if an actual mail is deleted as being spam.
    - Predicted 0, but truly 1: A little annoying to have a spam mail arrive in your inbox, but usually not a big problem



# Evaluating Classification models revisited

- **The confusion matrix** (binary case)
  - Display predicted class versus actual class
  - **Accuracy:**
    - (TP + TN) / (TP + FN + FP + TN)
    - The fraction of the time the classifier makes the correct classification

| | Predicted 1 (positive) (ex. spam) | Predicted 0 (negative) (ex. not spam) |
|---|---|---|
| Actual 1 (positive) (ex. spam) | **TP** (True Positive) | **FN** (False Negative) |
| Actual 0 (negative) (ex. not spam) | **FP** (False Positive) | **TN** (True Negative) |

# Evaluating Classification models revisited

- **The confusion matrix** (binary case)
  - Display predicted class versus actual class
  - Accuracy:
    - (TP + TN) / (TP + FN + FP + TN)
    - The fraction of the time the classifier makes the correct classification
  - **Precision** (/positive predictive value):
    - TP / (TP + FP)
    - The fraction of the positive classifications that is truly positive
    - If this is high, we have few real emails that are classified as spam

| | **Predicted 1 (positive) (ex. spam)** | **Predicted 0 (negative) (ex. not spam)** |
|---|---|---|
| **Actual 1 (positive) (ex. spam)** | **TP** (True Positive) | **FN** (False Negative) |
| **Actual 0 (negative) (ex. not spam)** | **FP** (False Positive) | **TN** (True Negative) |

# Evaluating Classification models revisited

- **The confusion matrix** (binary case)
  - Display predicted class versus actual class
  - Accuracy:
    - (TP + TN) / (TP + FN + FP + TN)
    - The fraction of the time the classifier makes the correct classification
  - Precision (/positive predictive value):
    - TP / (TP + FP)
    - The fraction of the positive classifications that is truly positive
    - If this is high, we have few real emails that are classified as spam
  - **Recall** (Sensitivity/true positive rate):
    - TP / (TP + FN)
    - The fraction of the actual positive cases that the classified detect as positive
    - If this is high, we have few cases of the disease that are not detected

| | **Predicted 1 (positive) (ex. spam)** | **Predicted 0 (negative) (ex. not spam)** |
|---|---|---|
| **Actual 1 (positive) (ex. spam)** | **TP** (True Positive) | **FN** (False Negative) |
| **Actual 0 (negative) (ex. not spam)** | **FP** (False Positive) | **TN** (True Negative) |

# Evaluating Classification models revisited

- **The confusion matrix** (binary case)
  - Display predicted class versus actual class
  - Accuracy:
    - (TP + TN) / (TP + FN + FP + TN)
    - The fraction of the time the classifier makes the correct classification
  - Precision (/positive predictive value):
    - TP / (TP + FP)
    - The fraction of the positive classifications that is truly positive
    - If this is high, we have few real emails that are classified as spam
  - Recall (Sensitivity/true positive rate):
    - TP / (TP + FN)
    - The fraction of the actual positive cases that the classified detect as positive
    - If this is high, we have few cases of the disease that are not detected
  - **F1 score**:
    - (2*Precision*Recall) / (Precision+Recall)

| | **Predicted 1 (positive) (ex. spam)** | **Predicted 0 (negative) (ex. not spam)** |
|---|---|---|
| **Actual 1 (positive) (ex. spam)** | **TP** (True Positive) | **FN** (False Negative) |
| **Actual 0 (negative) (ex. not spam)** | **FP** (False Positive) | **TN** (True Negative) |

# Evaluating Classification models revisited

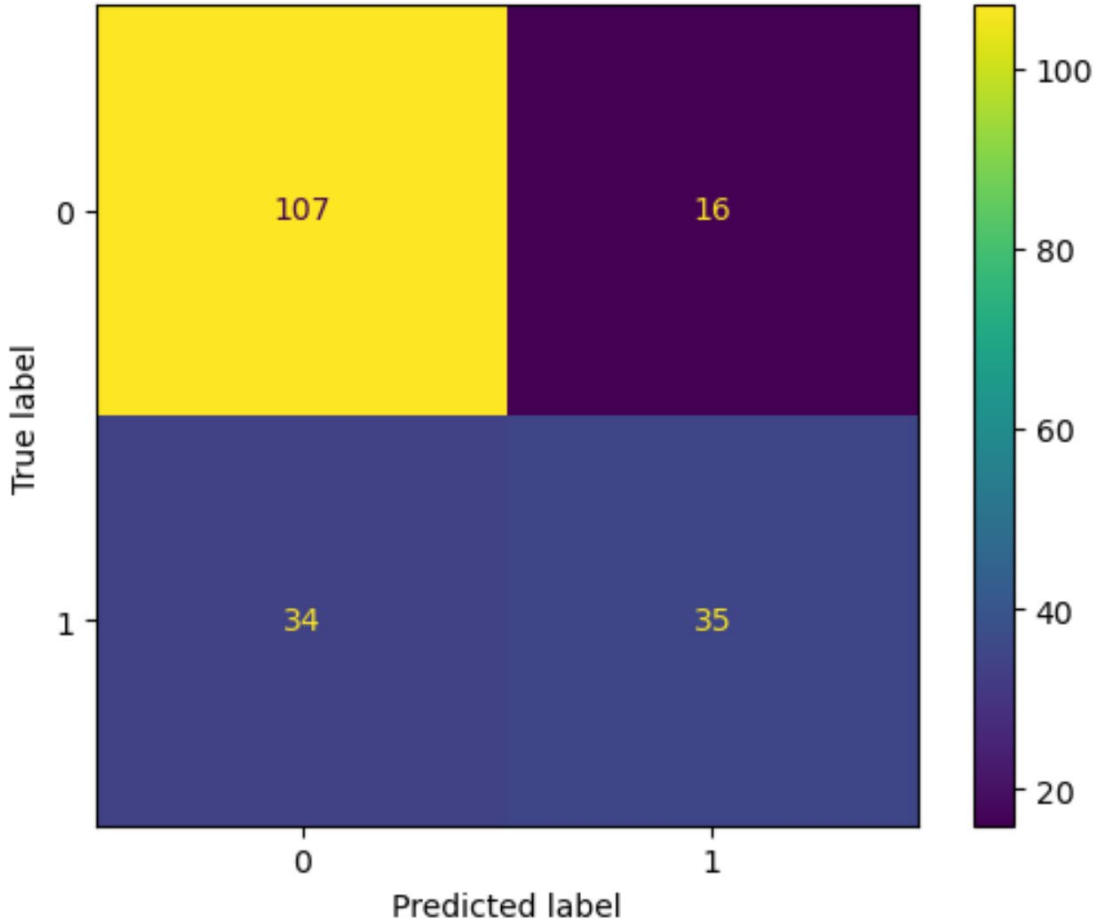
- **Other measures:**
  - **False positive rate**:
    - FP / (FP+ TN)
  - **False negative rate**:
    - FN / (FN+ TP)
  - **Specificity (/true negative rate)**:
    - TN / (TN + FP)
  - **Negative predictive value**:
    - TN / (TN + FN)
  - ….

| | Predicted 1 (positive) (ex. spam) | Predicted 0 (negative) (ex. not spam) |
|---|---|---|
| Actual 1 (positive) (ex. spam) | TP (True Positive) | FN (False Negative) |
| Actual 0 (negative) (ex. not spam) | FP (False Positive) | TN (True Negative) |

# Evaluating Classification models revisited

- Example: The KNN model (K=5) for the diabetes dataset from last time

| K | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| 5 | 0.682292 | 0.557143 | 0.565217 | 0.561151 |

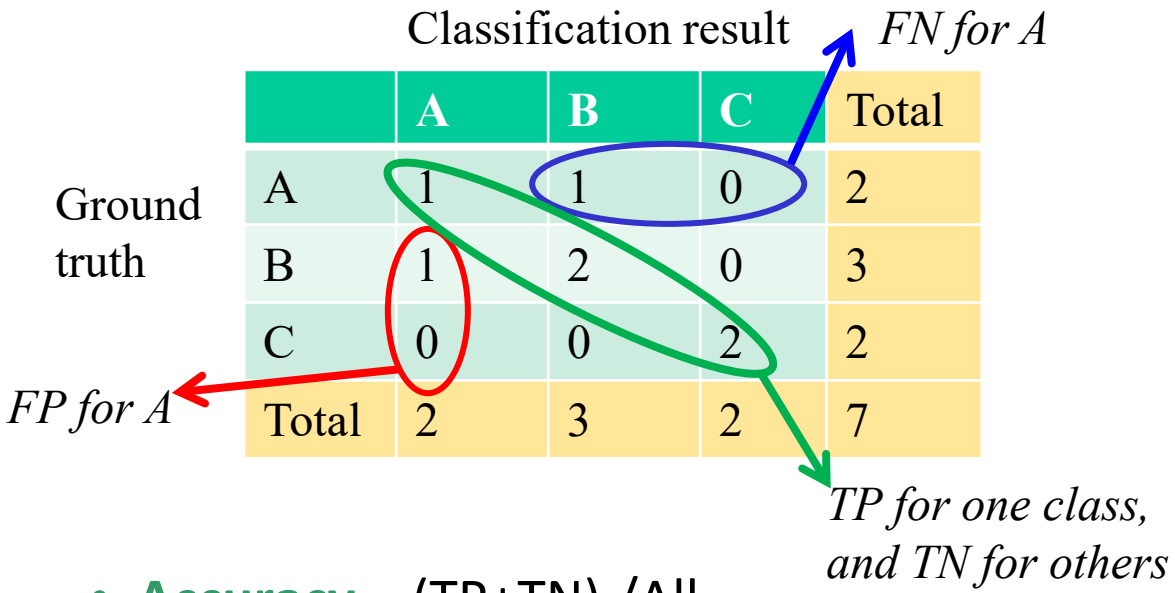


# Evaluating Classification models revisited

- **Beyond the binary Confusion Matrix**
  - Three pre-defined classes A, B, C
  - Ground truth and classification result

| Object | Ground Truth | Classification Result |
|---|---|---|
| object-1 | *A* | A |
| object-2 | *B* | A |
| object-3 | *C* | C |
| object-4 | *C* | C |
| object-5 | *B* | B |
| object-6 | *A* | B |
| object-7 | *B* | B |

- Confusion Matrix (N x N for N classes)
  - The cell M[*i*, *j*] counts the case that groundtruth *i* is classified as *j*




| | A | B | C | Total |
|---|---|---|---|---|
| A | 1 | 1 | 0 | 2 |
| B | 1 | 2 | 0 | 3 |
| C | 0 | 0 | 2 | 2 |
| Total | 2 | 3 | 2 | 7 |

- **Accuracy** = (TP+TN) /All
  - In this example, Accuracy = (1+2+2)/7 = 71.4%

# Evaluating Classification models revisited

- **Precision and Recall in general**
  - Calculate the precision $p_i$ for each class $C_i$
    - Overall precision is the *average* of all $p_i$ s
  - Calculate the recall $r_i$ for each class $C_i$
    - Overall recall is the *average* of all $r_i$ s
  - Example:
    - $p_A$=30/60=1/2, $r_A$=30/100=3/10
    - $p_B$=60/120=1/2, $r_B$=60/100=3/5
    - $p_C$=80/120=2/3, $r_C$=80/100=4/5
    - Overall precision = 5/9,
      overall recall = 17/30

**Confusion matrix**

Classification result

| Ground truth | A | B | C | Total |
|---|---|---|---|---|
| A | 30 | 50 | 20 | 100 |
| B | 20 | 60 | 20 | 100 |
| C | 10 | 10 | 80 | 100 |
| Total | 60 | 120 | 120 | 300 |

Recall for A: $r_A$

Precision for A: $p_A$

# Evaluating Classification models revisited

- **Analyze Your Confusion Matrix**
  - Essentially, the more zeroes or smaller the numbers on all cells but the diagonal, the better a classifier is. So, you may analyze your confusion matrix and tweak your features accordingly.
  - Confusion matrix gives strong clues as to where your classifier is going wrong.
    - E.g., if for Class A you can see that the classifier incorrectly predicts Class B for majority of the mislabeled cases, it indicates the classifier is somehow confused between classes A and B.
    - One way to fix this is to add discriminating features to improve classification of class A, e.g., more training data of A.

# Evaluating Classification models revisited

- **Class imbalance**
  - Rare positive examples but numerous negative ones, e.g., medical tests, fraud detection, etc. (…or the other way around)
  - Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
  - Typical methods for imbalance data in binary class classification:
    - Oversampling: re-sampling of data from positive class
    - Under-sampling: randomly eliminate tuples from negative class
    - Threshold-moving: moves the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
    - …
  - Still difficult for class imbalance problem on multiclass tasks

# Evaluating Classification models revisited

- **Classification with class probabilities**
  - All the evaluation metrics we have seen so far are evaluating classification models at set decision threshold (-we have set it)
  - However, as we saw for logistic regression last time, some classification algorithms give us class probabilities instead of hard class labels
  - But how do we choose the right threshold?
  - Can we evaluate a model irrespective of what threshold we chose?

# Evaluating Classification models revisited

- **Prediction probabilities** (like in logistic regression).
  - A probability of class membership is assigned instead of a label
  - A threshold can be used to control how to decide the predicted class label.

- Different thresholds lead to different metric values.
- This requires us to generate different confusion matrixes ☹

| ID | Actual | Prediction Probability | >0.6 | >0.7 | > 0.8 | Metric |
|---|---|---|---|---|---|---|
| 1 | 0 | 0.98 | 1 | 1 | 1 | |
| 2 | 1 | 0.67 | 1 | 0 | 0 | |
| 3 | 1 | 0.58 | 0 | 0 | 0 | |
| 4 | 0 | 0.78 | 1 | 1 | 0 | |
| 5 | 1 | 0.85 | 1 | 1 | 1 | |
| 6 | 0 | 0.86 | 1 | 1 | 1 | |
| 7 | 0 | 0.79 | 1 | 1 | 0 | |
| 8 | 0 | 0.89 | 1 | 1 | 1 | |
| 9 | 1 | 0.82 | 1 | 1 | 1 | |
| 10 | 0 | 0.86 | 1 | 1 | 1 | |
| | | | 0.75 | 0.5 | 0.5 | TPR |
| | | | 1 | 1 | 0.66 | FPR |
| | | | 0 | 0 | 0.33 | TNR |
| | | | 0.25 | 0.5 | 0.5 | FNR |

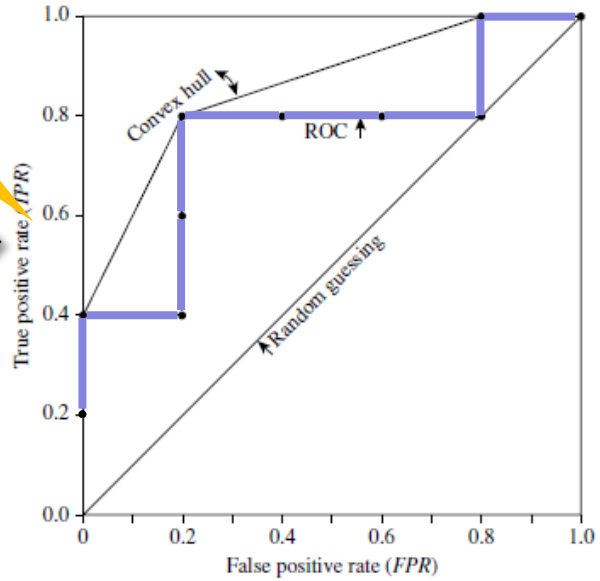
For positive label

# Evaluating Classification models revisited

- **Receiver Operating Characteristics (ROC)** curves: for visual comparison of binary classifiers
  - Rank your classification results in *descending* order of prediction probabilities
  - Calculate TPR and FPR for each current tuple in the ranked order
  - Mark each (FPR, TPR) point on the graph.
  - Connect all such points using a *convex hull*
- **NB**: TP, FP, TN, FN (and ***TPR*** and ***FPR***) change as you seen more tuples in classification result

| Tuple # | Class | Prob. | TP | FP | TN | FN | TPR | FPR |
|---|---|---|---|---|---|---|---|---|
| 1 | *P* | 0.90 | 1 | 0 | 5 | 4 | 0.2 | 0 |
| 2 | *P* | 0.80 | 2 | 0 | 5 | 3 | 0.4 | 0 |
| 3 | *N* | 0.70 | 2 | 1 | 4 | 3 | 0.4 | 0.2 |
| 4 | *P* | 0.60 | 3 | 1 | 4 | 2 | 0.6 | 0.2 |
| 5 | *P* | 0.55 | 4 | 1 | 4 | 1 | 0.8 | 0.2 |
| 6 | *N* | 0.54 | 4 | 2 | 3 | 1 | 0.8 | 0.4 |
| 7 | *N* | 0.53 | 4 | 3 | 2 | 1 | 0.8 | 0.6 |
| 8 | *N* | 0.51 | 4 | 4 | 1 | 1 | 0.8 | 0.8 |
| 9 | *P* | 0.50 | 5 | 4 | 0 | 1 | 1.0 | 0.8 |
| 10 | *N* | 0.40 | 5 | 5 | 0 | 0 | 1.0 | 1.0 |

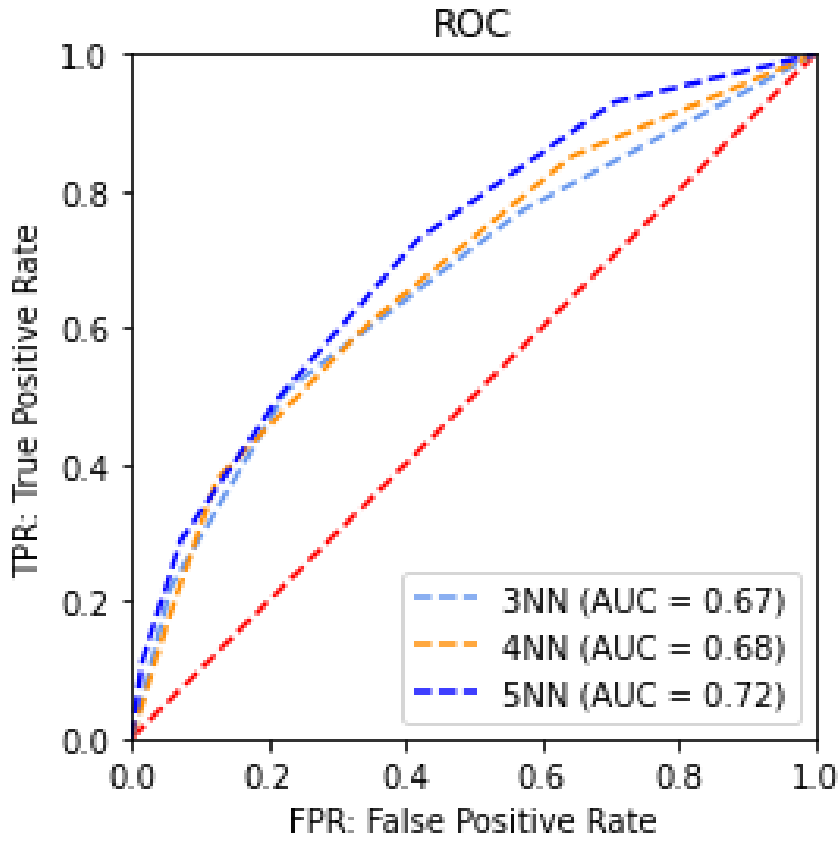
**Sensitivity/ recall**




**1- specificity**

# ROC Curves and AUC

- ROC Curves and AUC
  - A ROC curve shows the trade-off between the True Positive Rate and the False Positive Rate
  - The diagonal represents *random guessing*
  - The *area under the ROC curve* (**AUC**) is a measure of the "accuracy" of the model
  - The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model
  - A model with perfect accuracy will have an area of 1.0



# Evaluating Classification models revisited

- **Examples**
  - Let us look at the notebook “Evaluating classification models.ipynb”.

- **Exercise**
  - Do Exercise 1 in the notebook “Exercises in Classification II.ipynb”

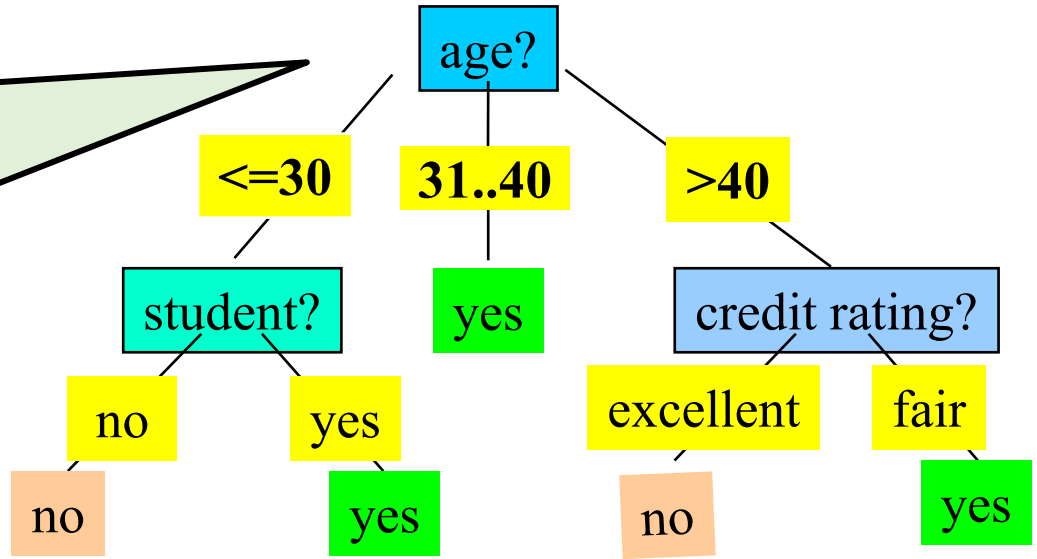
# Outline of this lecture

- Cross-validation
- Evaluating Classification models revisited
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting

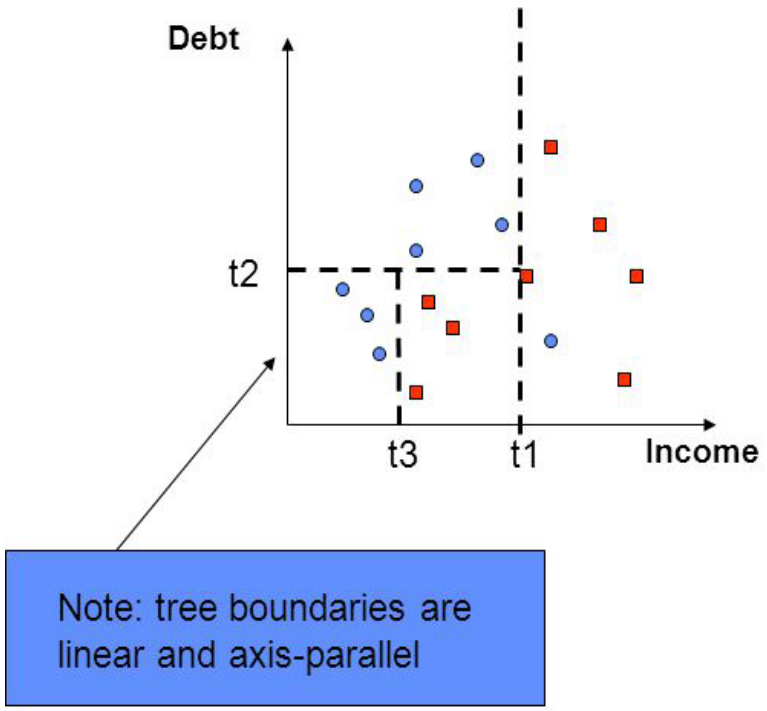
# Decision trees for classification

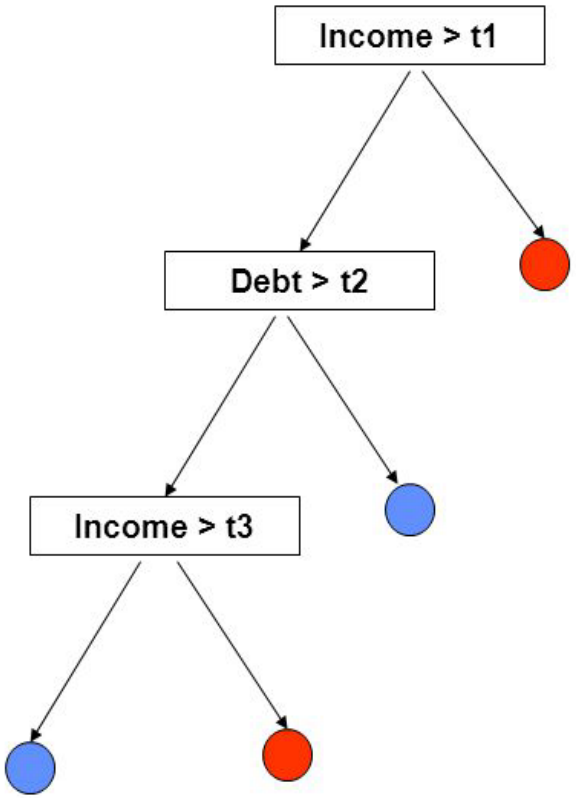
- **Decision Trees**
  - This classification model is organized as a tree for decision making – it is thus called a **decision tree**.
  - Internal nodes are associated with an *attribute/column* and arcs with *values* for that attribute.
  - A leaf node tells the predicted class label (where the branches ends).

- Each person has attributes/columns:
  - (age, student [yes/no], credit rating)
  - p1(18, yes, fair)
  - p2(55, no, excellent)
- **Two classes**
  - **Buy computer**
  - Not buy computer



# Decision trees for classification



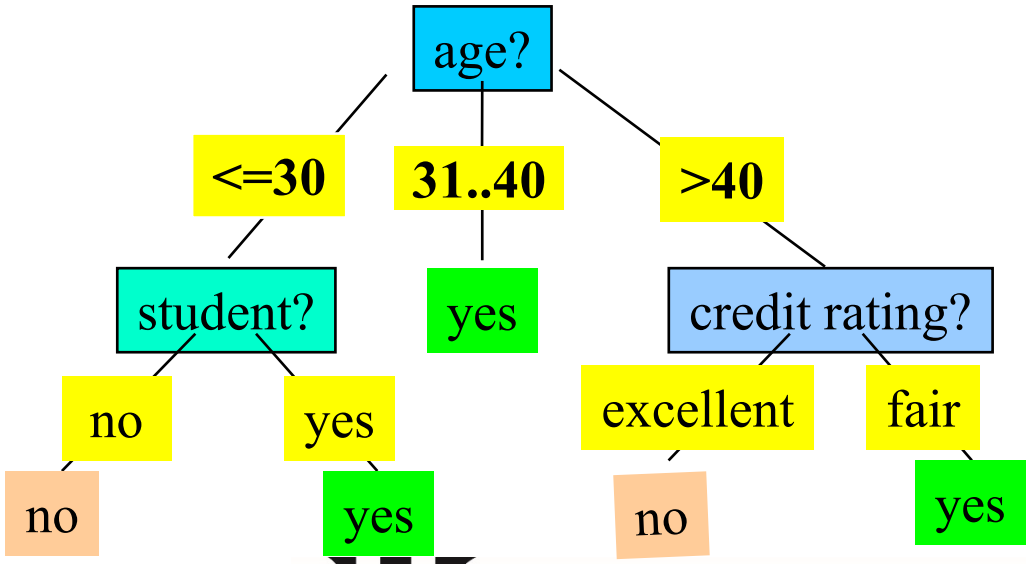

https://github.com/martian1231/decisionTreeFromScratch

- **Training a decision tree model**
  - Recursive binary splits top-down.
  - In a 2D feature dataset, each split corresponds to drawing a horizontal or vertical line.
  - Splits are chosen to minimize either *Gini index* or *Entropy* (both measures of note impurity).
  - Each final region corresponds to a leaf in the decision tree.
  - For each region, the predicted class corresponds to the most prevalent class – class probabilities can be obtained by noting the fraction of each class.

# Decision trees for classification

- **Pros and Cons of decision trees**
  - Pros: Easy to interpret (even by non experts) and visualize
  - Pros: Does not require much preprocessing such as scaling or creation of dummy variables for categorical variables
  - Pros: Highly flexible
  - Cons: Can grow big and then becomes hard to interpret
  - Cons: Does not have the same level of predictive accuracy as other models
  - Cons: Have high variance

- **Decision trees for regression**
  - The prediction is made by taking the average of values in a region
  - Instead of minimizing Gini index or entropy, we minimize a local version of RSS.



# Decision trees for classification

- **Examples**
  - Let us look at the notebook “Decision trees and ensemble models.ipynb”.

# Outline of this lecture

- Cross-validation
- Evaluating Classification models revisited
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting

# Ensemble models using bagging and boosting

- ***Ensemble models*** combine many simple models into one single potential powerful model
- Combining decision trees for classification (or regression) turns out to good candidates for such simple models
- Popular ensemble methods:
  - ***Bagging***: averaging the prediction over a set of independent classifiers
  - ***Random Forest***: a random set of decision trees
  - ***Boosting***: weighted vote with a set of dependent classifiers

# Ensemble models using bagging and boosting

- **Bagging: Boostrap aggregating**
  - Analogy: Diagnosis based on the *majority vote* of multiple doctors trained from samples of the same superset.
  - Reduce variance of models with high variance, such as decision trees
  - Training: Given a set D of d tuples, generate m new training data sets $D_i$s
    - Each training set $D_i$ of d' tuples is sampled with *replacement* from D (i.e., ***bootstrap***)
    - Set $D_i$ is used to train a classifier model $M_i$
  - Classification: classify an unknown sample x using all m models ($M_1$ to $M_m$)
    - Each classifier $M_i$ returns its class prediction
    - The bagged classifier M* counts the votes and assigns the class with the most votes to x
  - Accuracy
    - Often significantly better than a single classifier derived from D
    - For noisy data: not considerably worse, more robust

# Ensemble models using bagging and boosting

- **Random Forest (of Decision Trees)**
  - An improved version of bagging
    - In addition to bootstrap sampling, we are also sampling a subset of features for each decision tree
  - Formally, for each decision tree
    - We sample with replacement a subset of the original training set
    - We sample set of ***m*** of the feature variables – if there is a total of ***p*** features, a common choice for ***m*** is $\sqrt{p}$
    - We then train the decision tree on this subset of the training dataset and only allows for split involving the sampled m features.
  - The rationale behind Random Forest
    - That we also sample only a small subset of features for each decision tree makes the decision trees much more varied and uncorrelated, which in turn make the average more robust.
  - Decision rule
    - To make a prediction, Random forest average over all predictions from all the trees for regression and take the most popular vote for a class among all the trees for classification

# Ensemble models using bagging and boosting

- **Boosting**
  - Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the *previous* diagnosis accuracy
  - How boosting works?
    - Weights are assigned to each training tuple
    - A series of *k* classifiers is iteratively learned
    - After a classifier $M_i$ is learned, the weights are updated to allow the subsequent classifier, $M_{i+1}$, to pay more attention to the training tuples that were misclassified by $M_i$
    - The final M* combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
  - Compared with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data
  - Popular variations of boosting: ***AdaBoost*** and ***XGBoost***

# Ensemble models using bagging and boosting

- **Interpretability of ensemble models**
  - As we are combining many models to achieve higher accuracy, interpretability is no longer straight-forward – a classic example of the trade-off between predictive accuracy and interpretability
  - However, for ensemble models based on decision trees, we can record in each decision and for each feature variable $X_i$ the drop in Gini index (or local RSS) and average this across all the trees in the ensemble – this will give us the ***variable importance*** of the feature variable $X_i$.
    - In this way, we can get a ranking of which of the features are most important relative to each other in the predictions the ensemble model makes.

# Summary on Decision trees and ensemble models

- Decision trees are easy to understand, easy to train, and easy to interpret
- Decision trees are flexible and easily handle both numeric and categorical variables (without scaling) and can be used for both regression and classification
- Decision trees often have limited accuracy and very high variance
- For these reasons, Decision trees are ideal as the simple models combined in ensemble models
- Random Forest is an ensemble method that often perform well and rarely overfit
- Boosting methods are more prone to overfitting, but can also achieve higher predictive accuracy than Random Forest

# Ensemble Learning: Increasing the Accuracy

- **Examples**
  - Let us look at the notebook “Decision trees and ensemble models.ipynb”.

- **Exercise**
  - Do Exercise 2 in the notebook “Exercises in Classification II.ipynb” – the same as Exercise 2 in the notebook “Exercises in Classification I.ipynb” from last time