Data & Things

(Spring 26)

Monday February 16

Lecture 8: Classification II

Jens Ulrik Hansen

Roskilde Universitet

Outline of this lecture

- Hyper-parameter tuning and cross-validation
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting
- Dealing with unbalanced classes
- Exercises

Hyper-parameter tuning and cross-validation

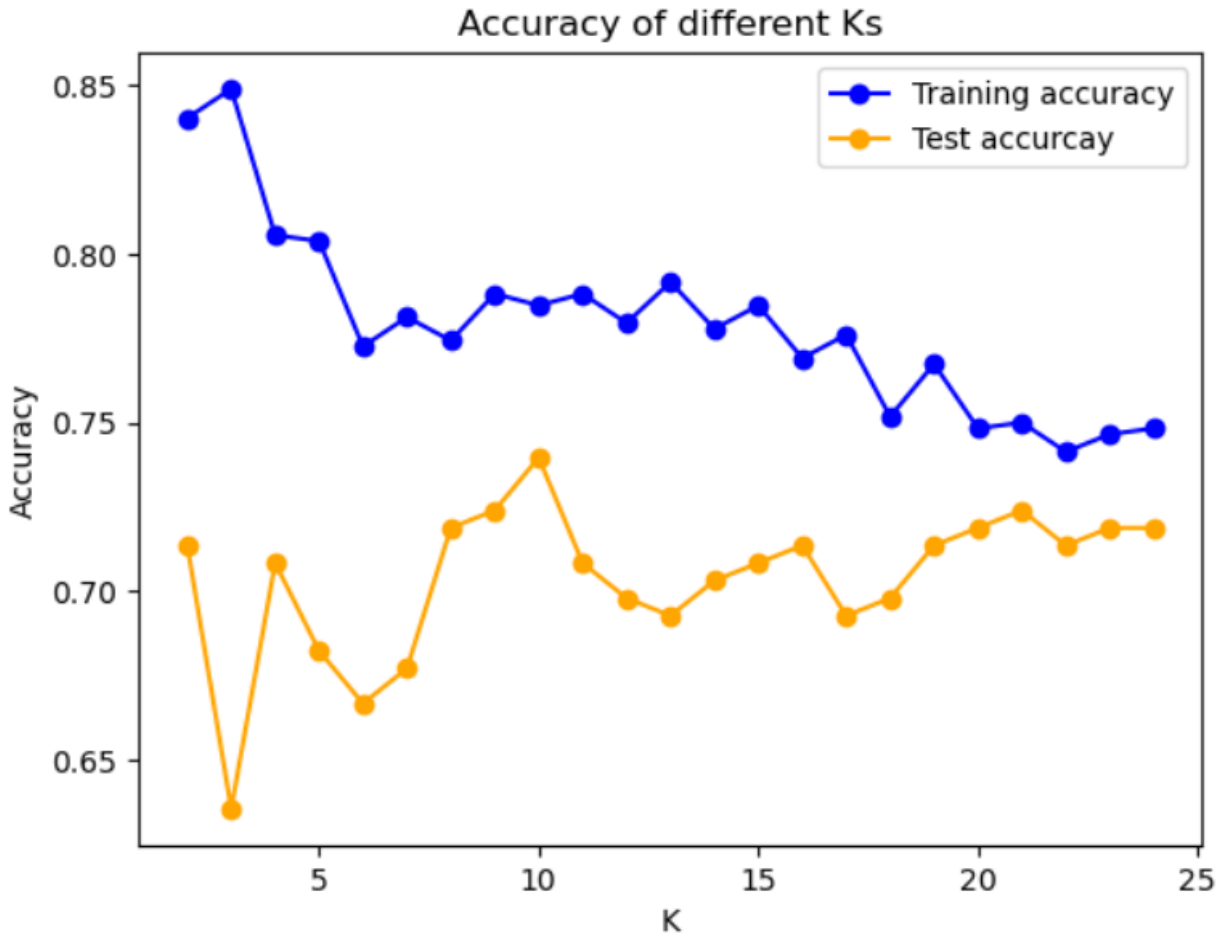
- **Hyper-parameters**
  - Many machine learning ***models are parametric***, in the sense of having parameters which values ***we optimize through training*** – for instance, the a and b of the linear regression model y = a + b*x.
  - Furthermore, many machine learning models also have ***hyper-parameters,*** that is, ***parameters we set in advance*** for each model, which are ***never changed during training*** of a model – for instance, the K in KNN classifiers.
  - Each setting of hyper-parameter values give rise to a different model. Finding the setting of hyper-parameters that result in the best model, is called ***hyper-parameter tuning***.
  - Visually looking for the optimal K in KNN classification, was a way of doing simple hyper-parameter tuning – once we have multiple hyper-parameters however, we need a more systematic and precise way of doing hyper-parameter tuning!
- More broadly, we sometimes also want to do model selection on other things than hyper-parameters, such as variable selection

Hyper-parameter tuning and cross-validation

- **Our train-test split is often not enough!**
  - We 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 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

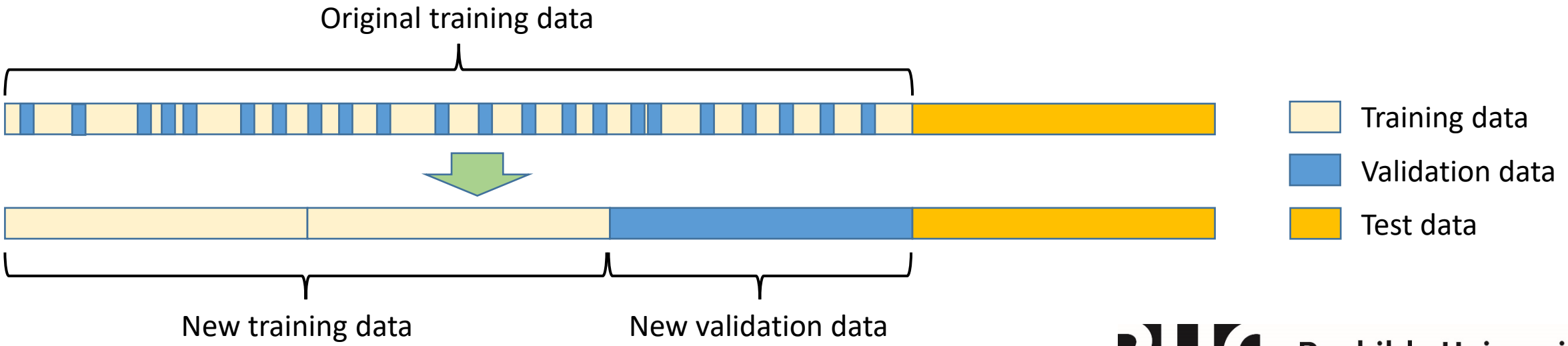
Hyper-parameter tuning and cross-validation

- 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 we got few data points both for training and testing no matter how we did the split
  - Different train-test split give different optimal Ks (as we shall see)
  - 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?



Hyper-parameter tuning and cross-validation

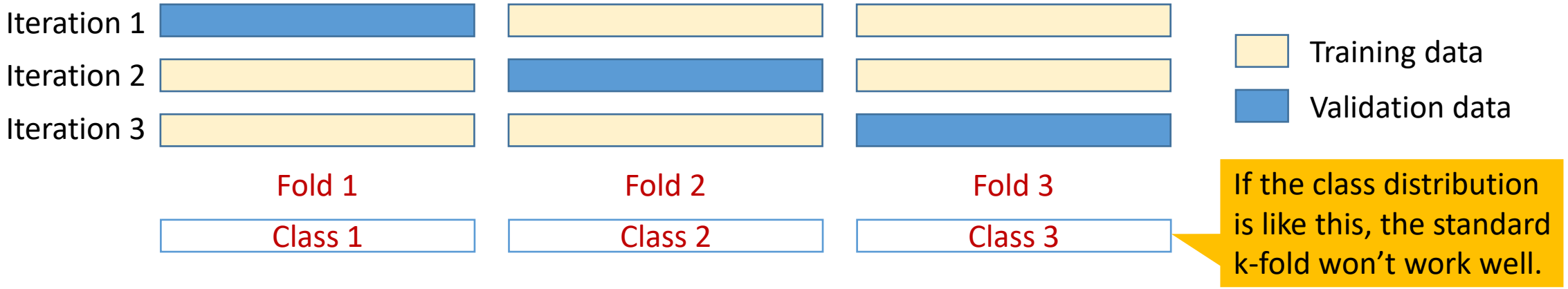
- **Train-Validation-Test split**
  - Use an additional validation set to evaluate and compare models
    - We leave the test set as is and split the training set further into a new training set and a validation set.
    - We then train our models on the new training set
    - We evaluate, compare and chose the final model based on performance on the validation set
    - Finally, we 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 used split is: 60% - training, 20% - validation, 20% - test



Hyper-parameter tuning and cross-validation

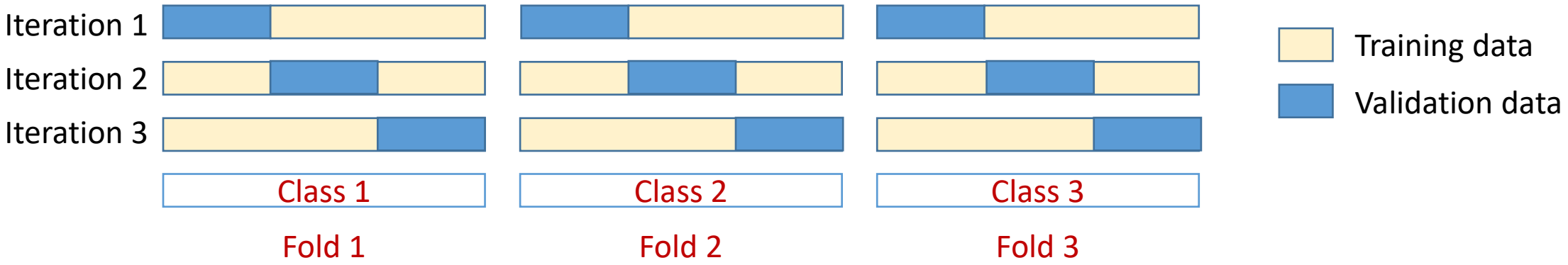
- ***k*-fold cross-validation** (k = 5 or k = 10 are popular choices)
  - Split the (train and validation) data D into *k* mutually exclusive subsets, each of approximately equal size: $D_1 \ldots D_k$. Each D_i is called a *fold*.
  - Do model construction and evaluation *k* times. Use the *average* 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



Hyper-parameter tuning and cross-validation

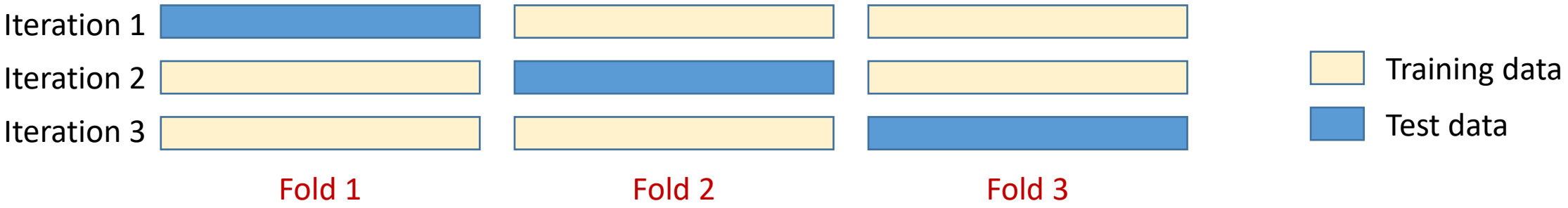
- **Stratified cross-validation**
  - folds are stratified so that *class distribution* in each fold is approximately the same as that in the initial given data.



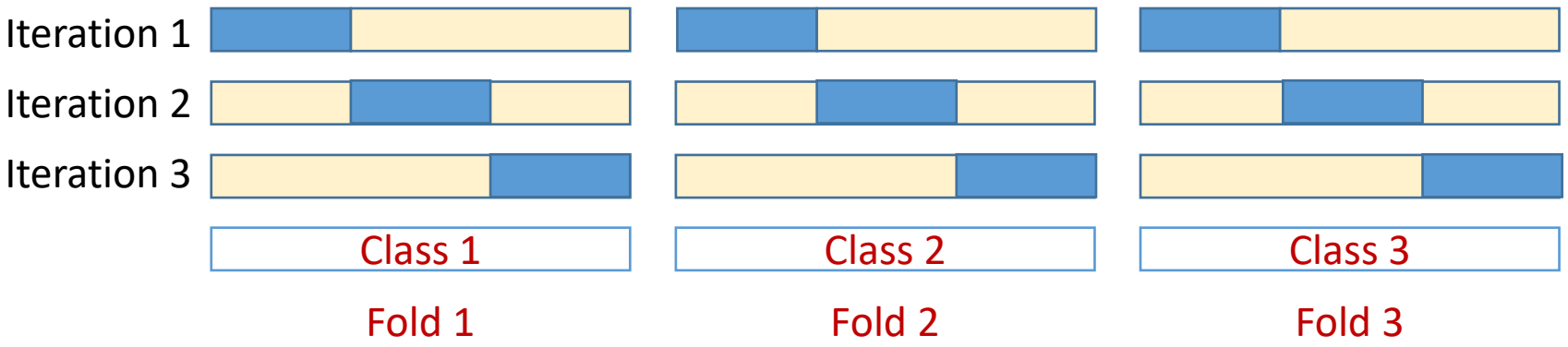
- **Leave-one-out cross-validation**
  - *k* is set to the number of data points/rows in the dataset
  - Use it only for small sized data; otherwise, too many models to construct!
  - Often 5- or 10-fold cross validation is just as good or even better

Hyper-parameter tuning and cross-validation

- Standard 3-fold cross validation



- Stratified 3-fold cross validation



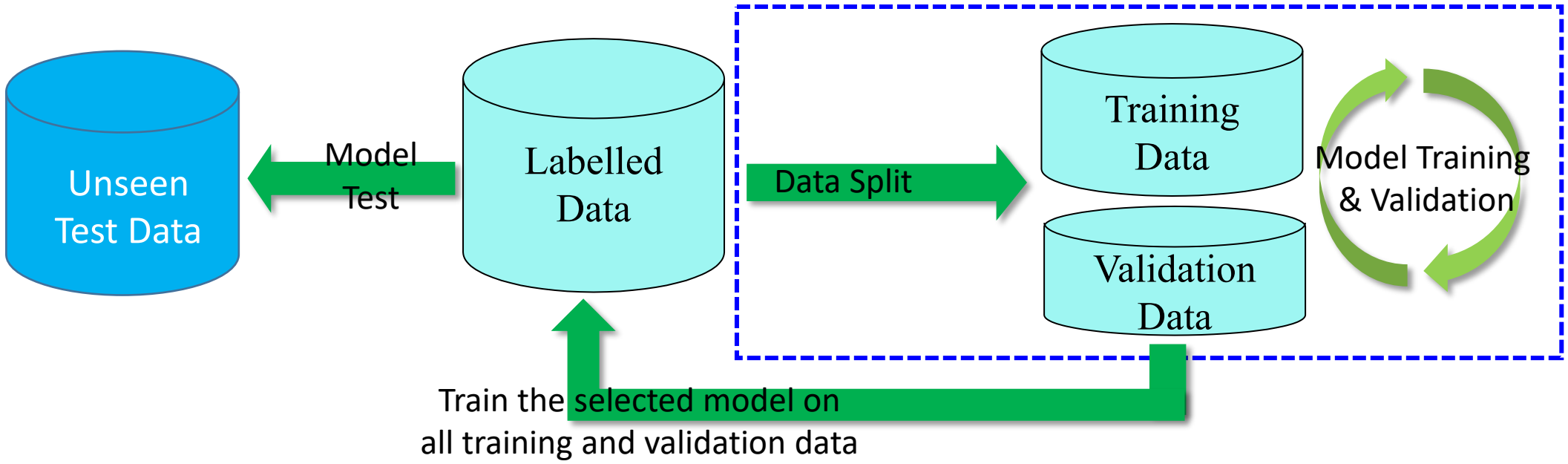
Default in scikit-learn:
Stratified **5**-fold

Hyper-parameter tuning and 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, …

Hyper-parameter tuning and 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)***



Hyper-parameter tuning and cross-validation

- **Examples**
  - Let us look at the notebook “Hyper-parameter tuning and cross-validation.ipynb”.

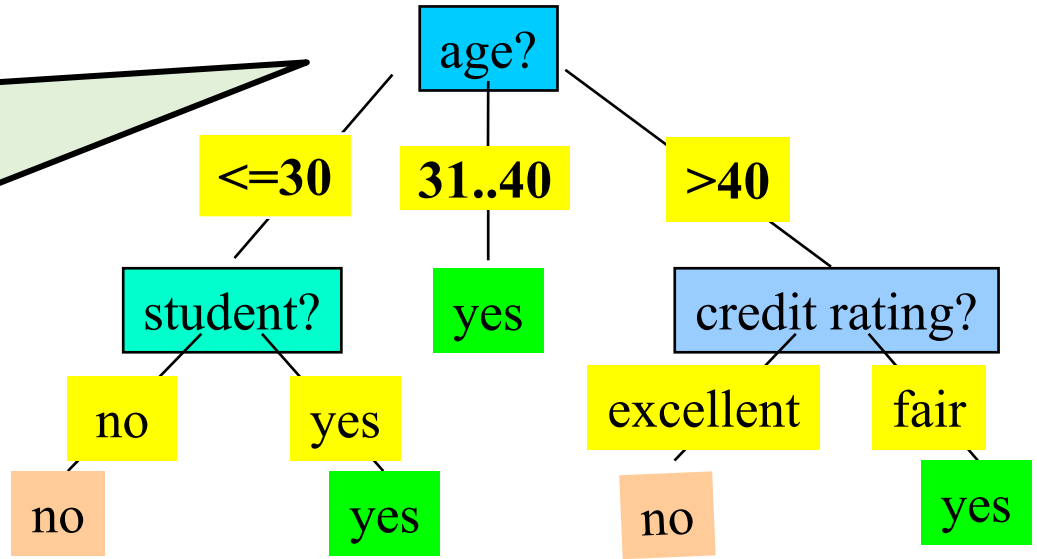
Outline of this lecture

- Hyper-parameter tuning and cross-validation
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting
- Dealing with unbalanced classes
- Exercises

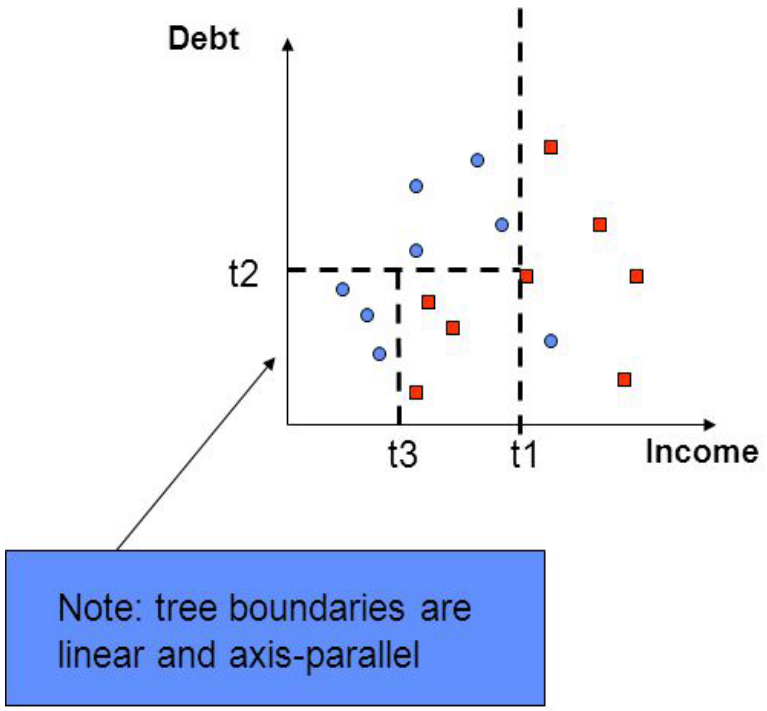
Decision trees for classification

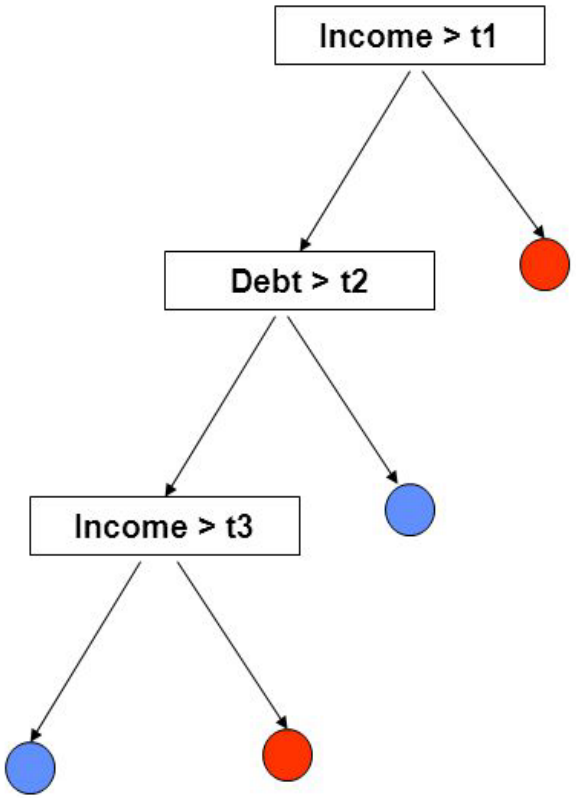
- **Decision Trees**
  - This classification model is organized as a tree for decision making – it is thus called a **decision tree**.
  - Internal nodes are associated with an *attribute/column* and arcs with *values* for that attribute.
  - A leaf node tells the predicted class label (where the branches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





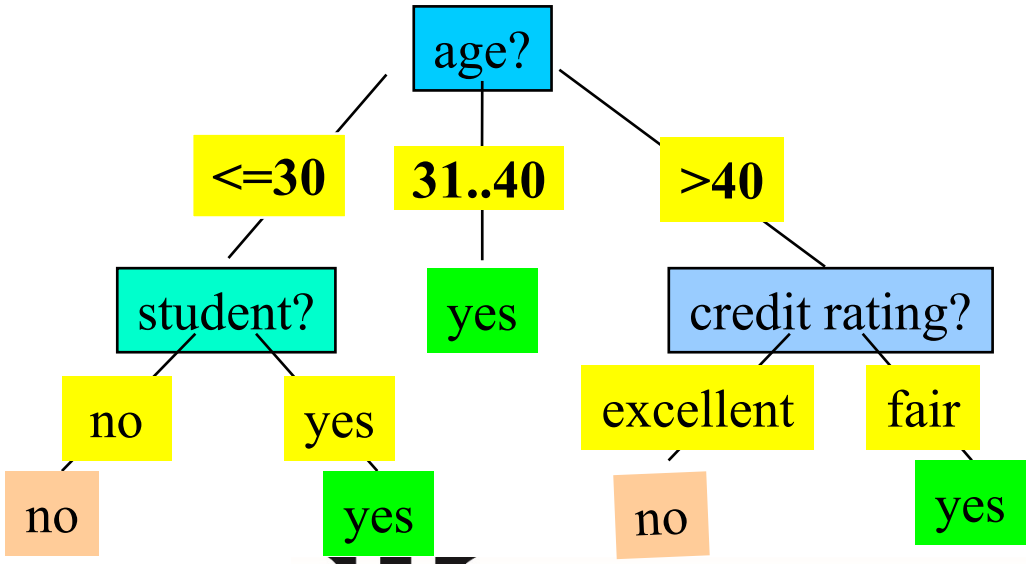
https://github.com/martian1231/decisionTreeFromScratch

- **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

- Hyper-parameter tuning and cross-validation
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting
- Dealing with unbalanced classes
- Exercises

Ensemble models using bagging and boosting

- ***Ensemble models*** combine many simple models into one single potential powerful model
- Decision trees for classification (or regression) turns out to be 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_is
    - 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 note 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 models using bagging and boosting

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

Outline of this lecture

- Hyper-parameter tuning and cross-validation
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting
- Dealing with unbalanced classes
- Exercises

Dealing with unbalanced classes

- **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

Dealing with unbalanced classes

- **Class imbalance**
  - Examples: Rare positive but numerous negative ones, e.g., medical tests, fraud detection, etc. (…or the other way around)
  - Traditional machine learning methods assume a balanced distribution of classes and equal error costs ***– these are not suitable for class-imbalanced data!***
  - Example: Image an classification task with a response variable that is either 1 for the subject a disease, and 0 if the subject does not have the disease.
    - If it is a rare disease, maybe only 5% of the people/data points in our training data has 1 for the response variable.
    - In this case, we can define a very simple model with 95% accuracy, if we just always predict 0!
    - The model might "think" a little along these line, in the sense that cases with y=0, will have less effect on the training.

Dealing with unbalanced classes

- **Typical methods to deal with unbalanced data in (binary) classification**
  - Oversampling: Multiple sampling with replacement of data from minority class
  - Undersampling: Only sample some of the data points from the majority class
  - Combing over- and under-sampling: Combining the two above approaches
  - More advanced sampling technique like SMOKE:
    - Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357. https://www.jair.org/index.php/jair/article/view/10302
  - 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
  - …

Dealing with unbalanced classes

- **Examples**
  - Let us look at the notebook “Dealing with unbalanced classes.ipynb”.

Outline of this lecture

- Hyper-parameter tuning and cross-validation
- Decision trees for classification (and regression)
- Ensemble models using bagging and boosting
- Dealing with unbalanced classes
- Exercises

Exercises

- Do Exercise 1 and 2 in the notebook “Exercises in Classification II.ipynb”