Data Science and Visualization (DSV, F23)

5. Classification (II)

Hua Lu
https://luhua.ruc.dk; luhua@ruc.dk
PLIS, IMT, RUC

Agenda

- **KNN**
- Data scaling
- Model evaluation and selection

Eager *vs* Lazy Learning

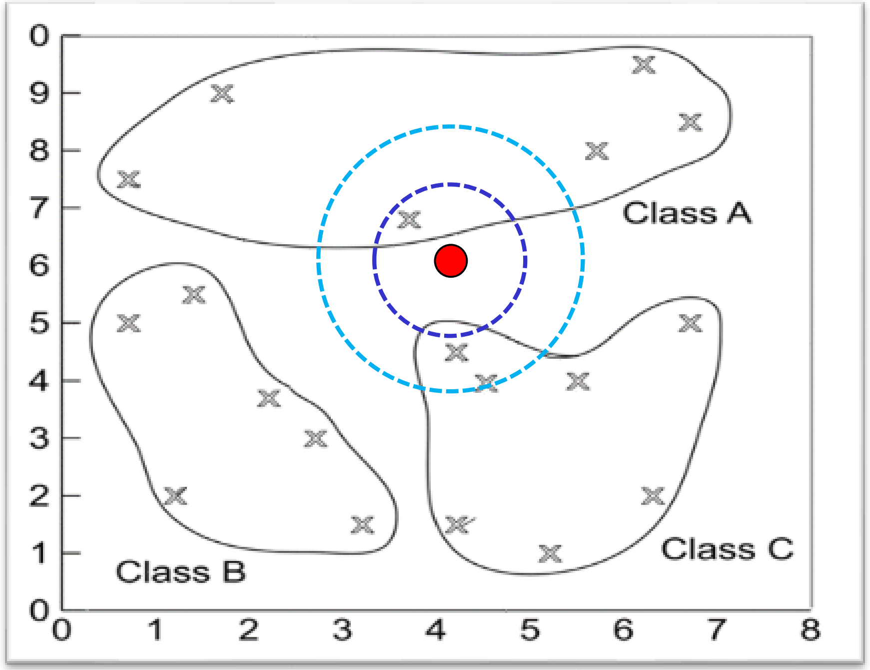
- **Eager learning** (model based methods): Given a set of training samples, constructs a classification model before receiving new (e.g., test) data to classify.
  - More time in training but less time in predicting/classification
  - E.g., we need to construct a decision tree before using it.
- **Lazy learning** (e.g., instance-based learning): Simply stores training data as instances (or only minor processing) and waits until a new instance must be classified
  - Less time in training but more time in predicting/classification
  - E.g., k nearest neighbors: Instances represented as points for which *distances* can be measured.

K Nearest Neighbors (KNN)

- Instance data set (training data)
  - A set *D* contains |*D*| (≥ K) items, each is labeled with a class.
  - *D* = {(item, class)}
  - *D* should cover *all* pre-defined class: |*D*| ≥ *C* (totally *C* classes)
- Classification
  - For a given item *t* to be classified, we find its **K** nearest neighbor items (**decision set**) from *D*.
    - Distance measurement: Usually Euclidean distance
  - Count the class labels in the K NNs, and give *t* the most frequent class label.
    - In other words, item *t* is placed in the class with the highest number of NNs.
    - NB: the **decision rule** can be different.

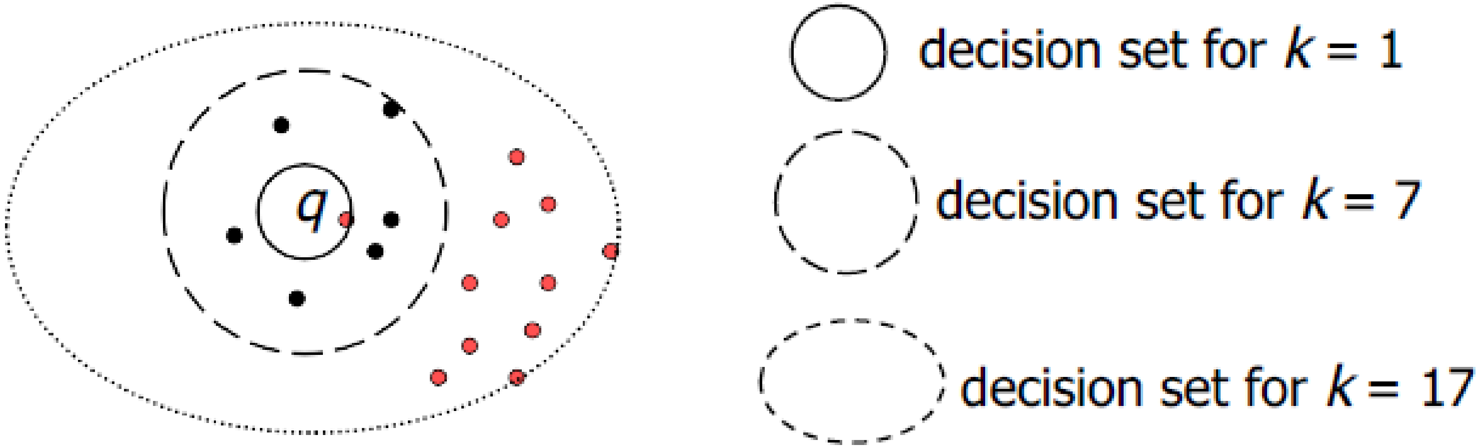
KNN Example

- Different Ks may lead to different classification results.
  - K = 1: Class A
  - K = 3: Class C



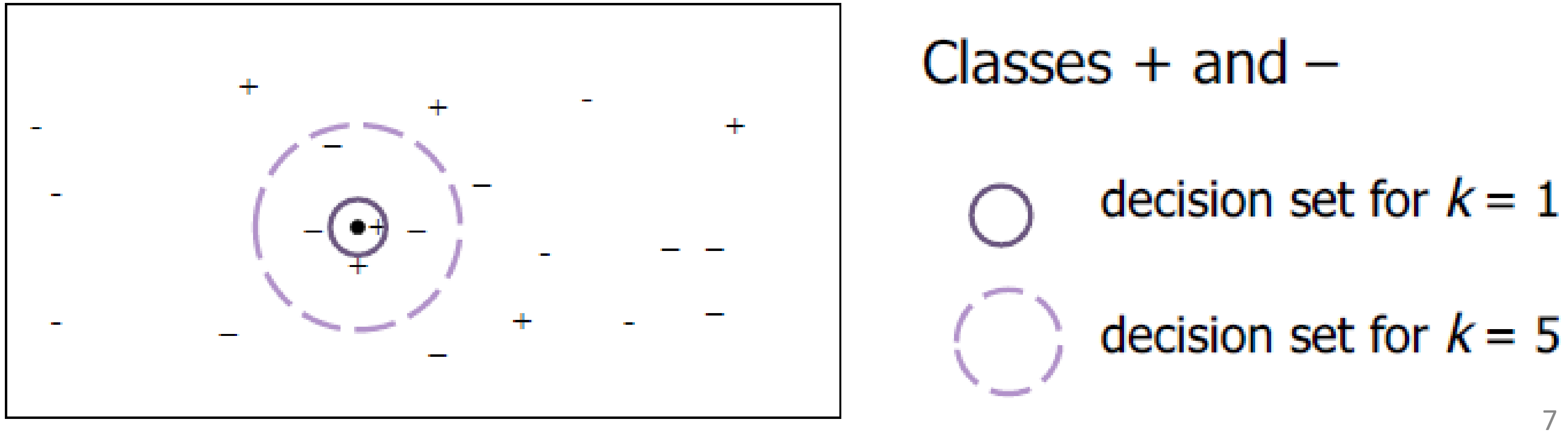
Appropriate Value for K

- Different Ks may lead to different classification results.
- Too small K: High sensitivity to outliers
- Too large K: Decision set contains many items from other classes.
- Empirically, 1<<K<10 yields a high classification accuracy in many cases.



Decision Rules of KNN

- Using unit weights (i.e., no weights) for the decision set
  - Simply “majority vote” or **standard rule**
  - For k=5 in the example, the rule yields class “-”
- Using the *reciprocal square of the distances* as weights
  - For k=5 in the example, the rule yields class “+”
- Using *a-priori probability (frequency) of classes* as weight
  - For k=5 in the example, the rule yields class “+”
    - “-”: 3/15 = 1/5
    - “+”: 2/6 = 1/3



Example in Jupyter Notebook

- Diabetes dataset
  - 768 data objects of 9 columns/attributes
  - Available in Moodle
- KNN for classification (2 classes)
  - 1: Positive of diabetes
  - 0: Negative
- Lecture5_KNN_diabetes.ipynb

Pros and Cons of KNN

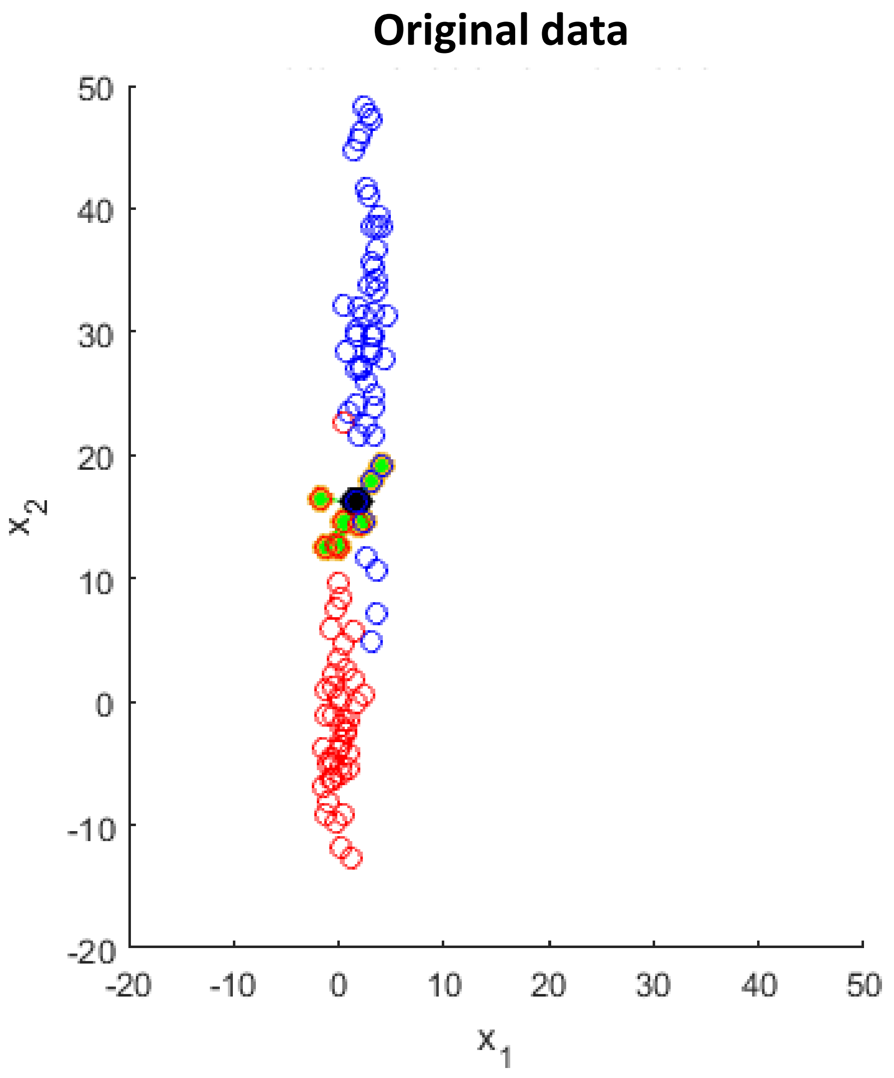
- Applicability: sample (training) data required only without training
- High classification accuracy in many applications
- Easy incremental adaptation to new sample objects
- Also useful for prediction
- Robust to noisy data by averaging K nearest neighbors

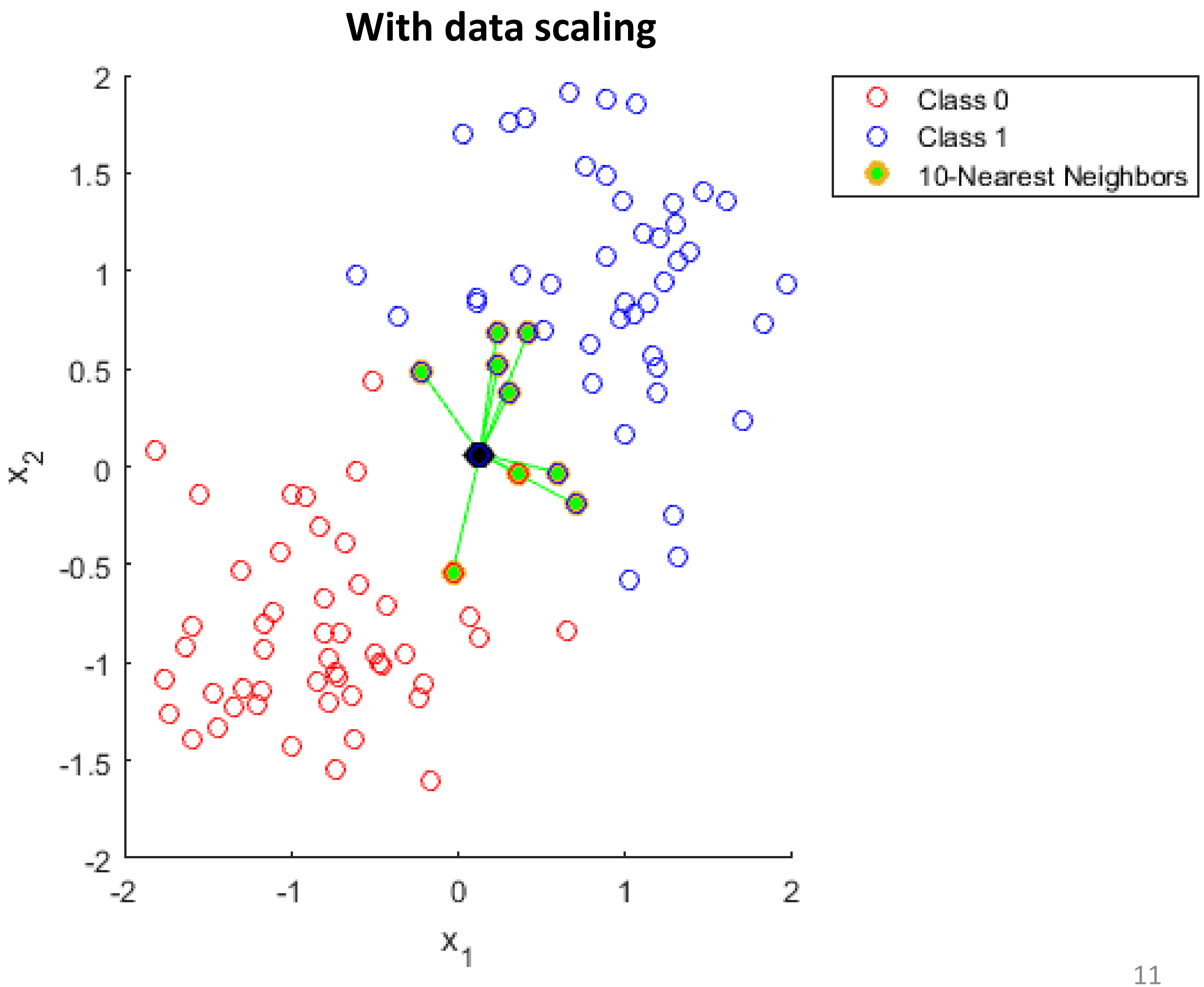
- Naïve implementation is inefficient
  - KNN search is not straightforward. Support by database in query processing may help.
- Does not produce explicit knowledge about classes but some explanation information.
- Curse of dimensionality: distance could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

Agenda

- KNN
- **Data scaling**
  - Why, what and how
- Model evaluation and selection

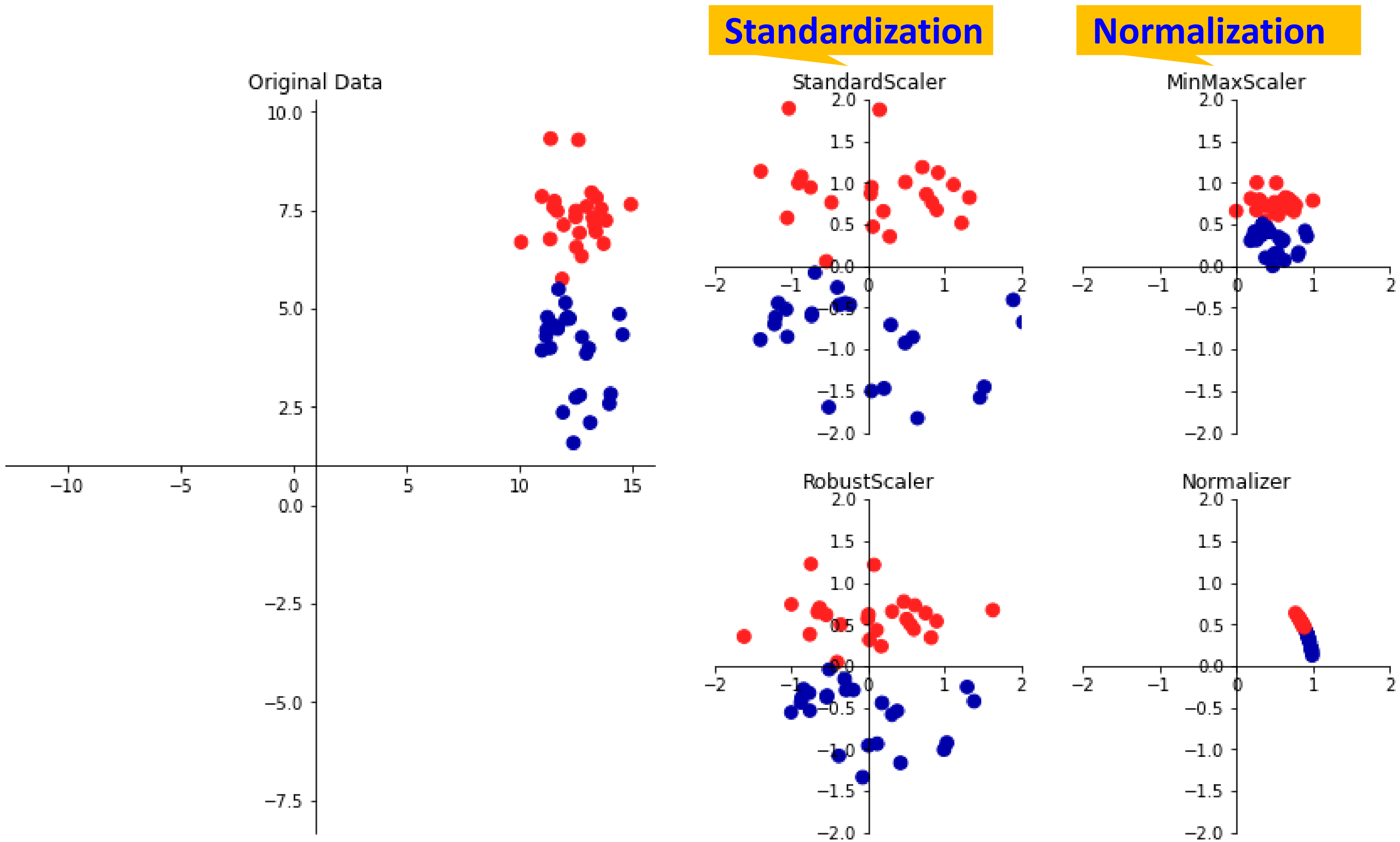
A Motivation Example





https://stats.stackexchange.com/questions/287425/why-do-you-need-to-scale-data-in-knn

Data Scaling



- **StandardScaler**
  - For each feature: mean=0 and variance=1
- **MinMaxScaler**
  - Shifts the data, each feature falls in [0..1]
- RobustScaler
  - Similar to SS but uses median and quartile to avoid outliers
- Normalizer
  - Scales each data point *s.t.* its Euclidean distance to (0, 0) is 1
  - Used when *only* the direction matters

Notes on Data Scaling

- Observe and/or plot your data to see how it skews
- Choose the right scaler you want to use
- Apply the scaler to *both* training and testing data
    - Apply the scaling on the whole original dataset
    - Then split the scaled dataset
- Standardization *or* Normalization? (Rule of thumb)
    - Normal data distribution: standardization; otherwise normalization
    - If uncertain: normalization; or standardization followed by normalization
    - Try different ways and decide the option with the best model performance

Continued Example in Jupyter Notebook

- Diabetes dataset
  - 768 data objects of 9 columns/attributes
  - Available in Moodle
- Data scaling effect for classification
- Lecture5_KNN_diabetes.ipynb
  - No scaling
  - StandardScaler
  - MinMaxScaler

DEMO

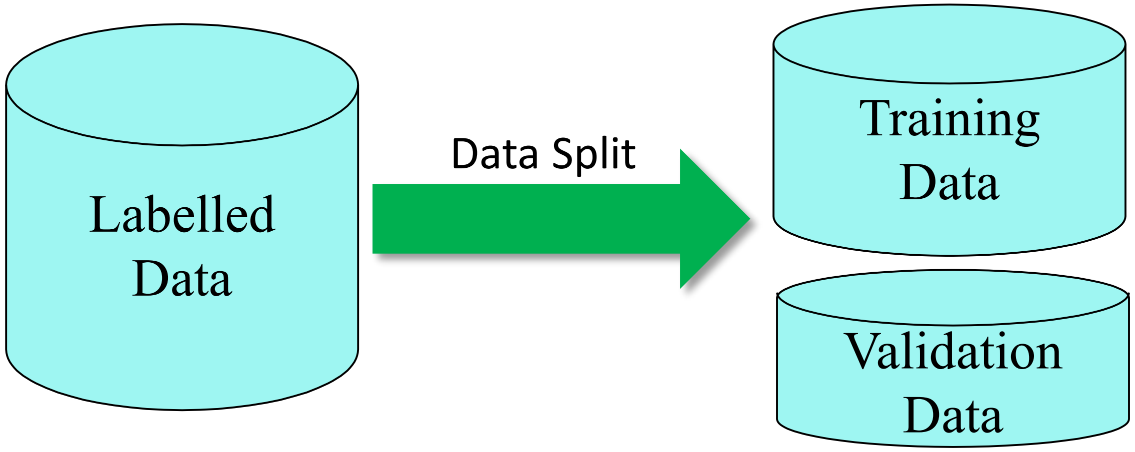
Agenda

- KNN
- Data scaling
- **Model evaluation and selection**
  - Data split and model evaluation
  - ROC and AUC for binary classification

Split Labelled Data

- sklearn.model_selection.train_test_split
- X_train, X_test, y_train, y_test = **train_test_split**(X, y, test_size=0.30, random_state=42)
  - test_size: validation/test data percentage
  - random_state: randomization of the split
- Different ways of split can result in different models and performance
  - Lecture5_KNN_diabetes.ipynb
    - Effect of train_test_split(.)

jupyter DEMO



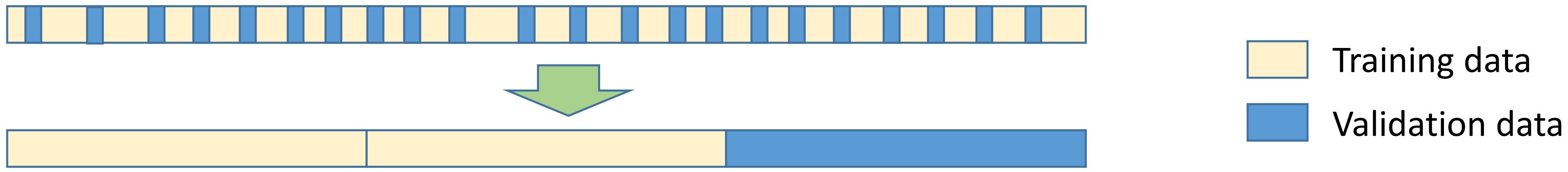
Model Evaluation and Selection

- Use validation set of labeled data samples instead of training set when assessing model accuracy
  - Otherwise, overfitting!
    - A model focuses so much on the training data that it does not generalize well to unseen data in predication.
- All labeled samples form D. How to split D into training and validation sets?
  - Holdout method, random subsampling
  - Cross-validation (k-fold)
  - Bootstrap (use it only when your data is not sufficient)
- These methods differ in how you partition/split all your labelled sample data into training set and validation set

Holdout

sklearn.model_selection.**train_test_split(.)**

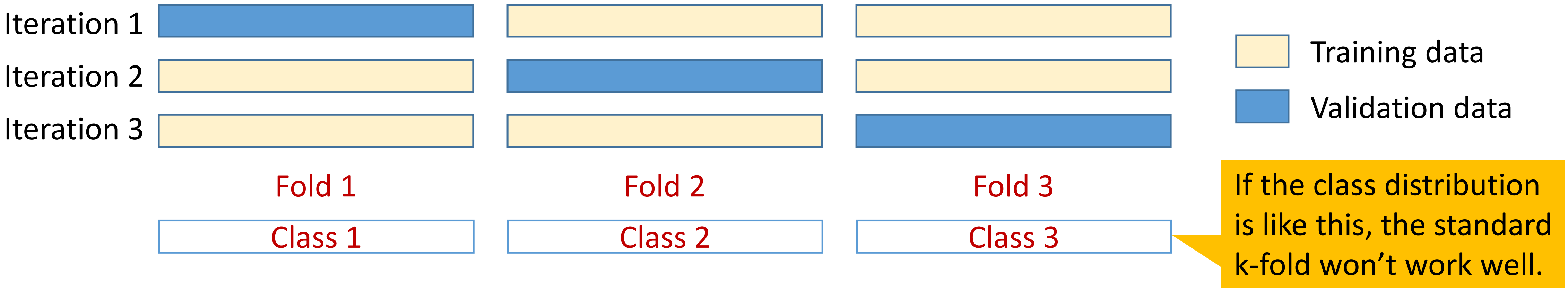
- Split the given labelled data *randomly* into two *independent* sets
  - Training set (e.g., 2/3) for model construction
  - Validation set (e.g., 1/3) for accuracy assessment



- **Random sampling**: a variant of holdout
  - Repeat holdout *k* times, accuracy = *average* of the accuracies obtained

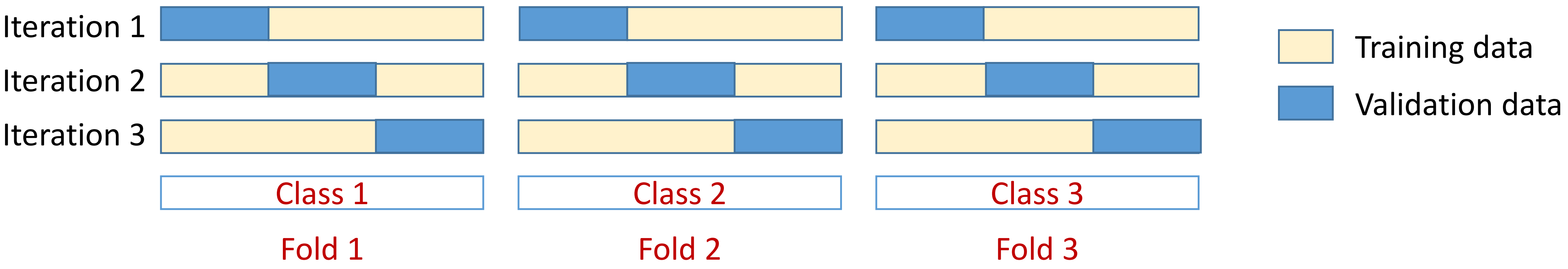
Standard Cross-Validation (CV)

- Aka ***k*-fold** (*k* = 10 is most popular)
  - Split the sample 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 for *k* time. 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



Variants of k-fold

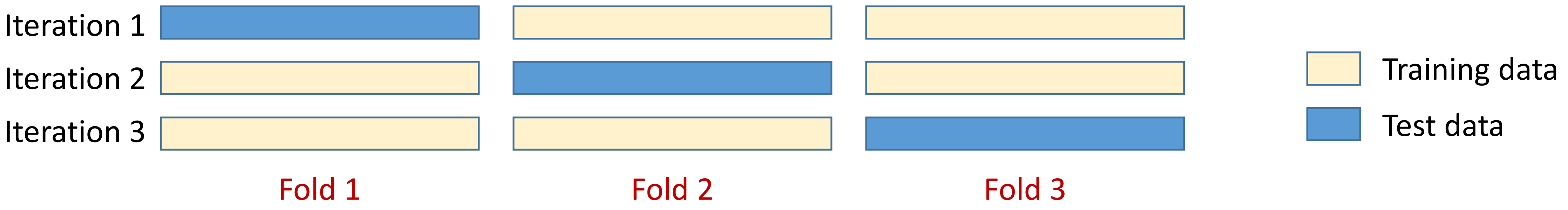
- Stratified cross- validation
  - folds are stratified so that *class distribution* in each fold is approximately the same as that in the initial given data.



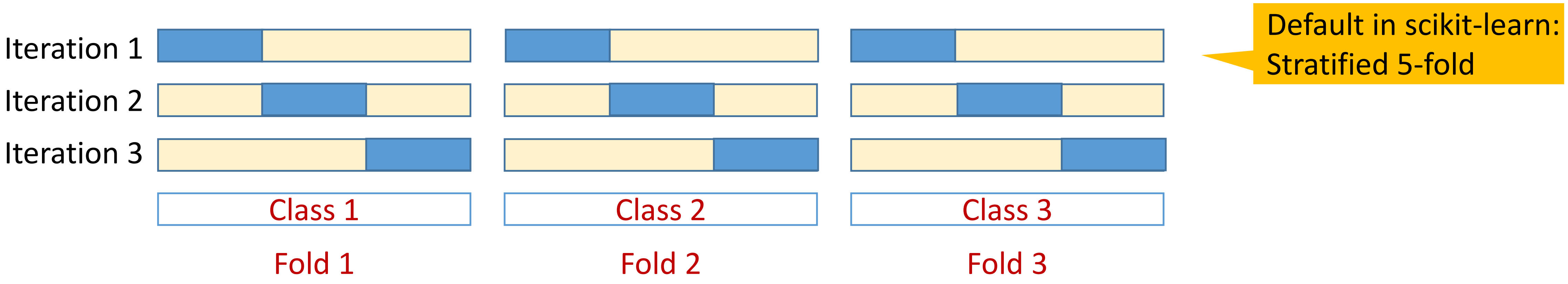
- **Leave-one-out**: k-fold where k = # of sample points
  - Use it only for small sized data; otherwise too many models to construct.

k-fold Cross Validation

- Standard 3-fold cross validation



- Stratified 3-fold cross validation



Continued Example in Jupyter Notebook

- Diabetes dataset
  - 768 data objects of 9 columns/attributes
  - Available in Moodle
- Cross-validation for classification
- Lecture5_KNN_diabetes.ipynb
  - Stratified cross validation
  - Standard cross validation
  - LeaveOut

Notes on Cross-Validation

- CV 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 CV 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, …
- By using CV, we can decide what type of model to use, and tune hyperparameters for constructing a model
    - Hyperparameters: algorithm parameters that can be set by the user before training a model. E.g., gini or entropy for a DT, K for KNN, test_size and random_state for train_test_split(.) …
    - In contrast, model parameters are learned internally from training data
        - E.g., how many levels actually in a DT?

Bootstrap

Advanced

- **Bootstrap**
  - Given a data set D with *m* tuples, sample uniformly *with replacement*
    - Select one tuple randomly, put it in a set D', and put it back to D.
    - Repeat *m* times
  - Training set: D' (with *m* tuples that may repeat)
  - Validation set: D \ D' (D is not changed)
- Remarks
  - No overfitting
    - It can be proved at about 36.8% tuples in D do not enter D'
    - When m is infinite, $(1 – 1/m)^m \approx e^{-1} = 0.368$
    - This is a.k.a. **.632 bootstrap**
  - Works well with a small data set D
  - But the original data distribution is distorted. So don't use bootstrap when your training data is sufficient.

Issues Affecting Model Selection

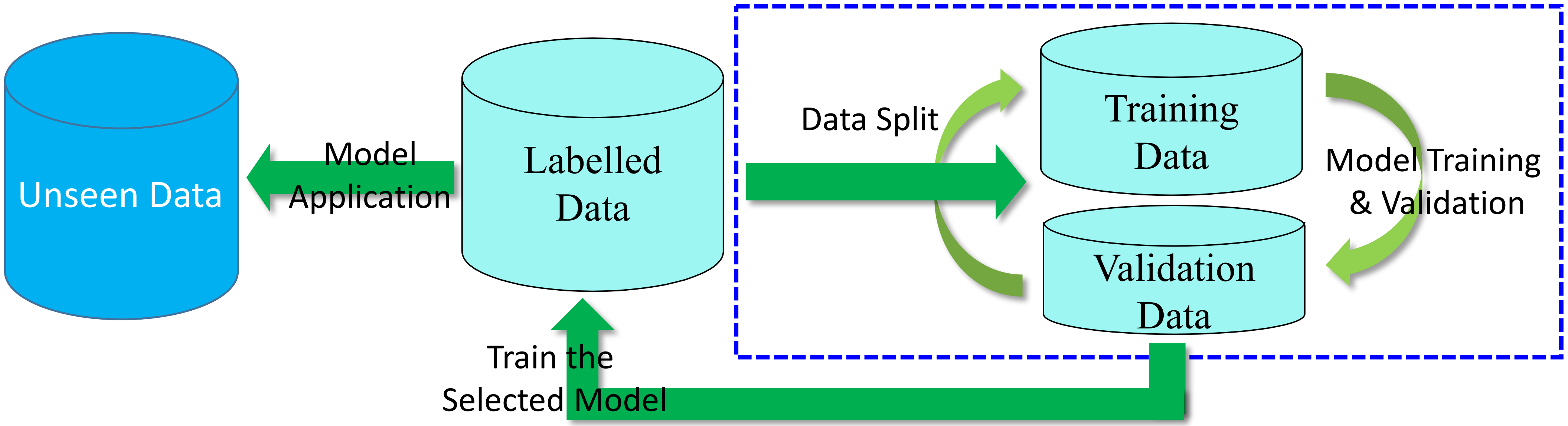
- **Accuracy**
  - Classifier accuracy: predicting class label
- **Speed**
  - Time to construct the model (training time)
  - Time to use the model (classification/prediction time)
- **Robustness**
  - How well to handle noise and missing values
- **Scalability**
  - Efficiency in disk-resident databases
- **Interpretability**
  - Understanding and insight provided by the model
- Other measures, e.g., decision tree size

Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., COVID-19 tests, fraud, oil-spill, fault, etc.
- 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
  - Ensemble techniques: Ensemble multiple classifiers to be introduced next
- Still difficult for class imbalance problem on multiclass tasks

After Validation: Making Use of All Labelled Data

- CV enables us to select the type of model (including hyperparameters) with the *best expected generalization ability* (to unseen data)
- We train the selected model using all labelled data
- We apply the final model to unseen data (test in applications)



Binary Classification Performance Metrics

- **Sensitivity** / True Positive Rate / Recall
  - TPR = TP / (TP + FN)
- False Negative Rate
  - FNR = FN / (TP + FN) = 1 - TPR
- **Specificity** / True Negative Rate
  - TNR = TN / (TN + FP)
- False Positive Rate
  - FPR = FP / (TN + FP) = 1 - TNR

		Predictions	
		POSITIVE	**NEGATIVE**
Groundtruth	**POSITIVE**	**TP**	**FN**
	NEGTIVE	**FP**	**TN**

Prediction Probability

- To predict a data object's *probability* of belonging to different classes.
  - A threshold can be used to control how to decide the predicted class label.
  - E.g., KNN's decision rule can be changed to do so

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

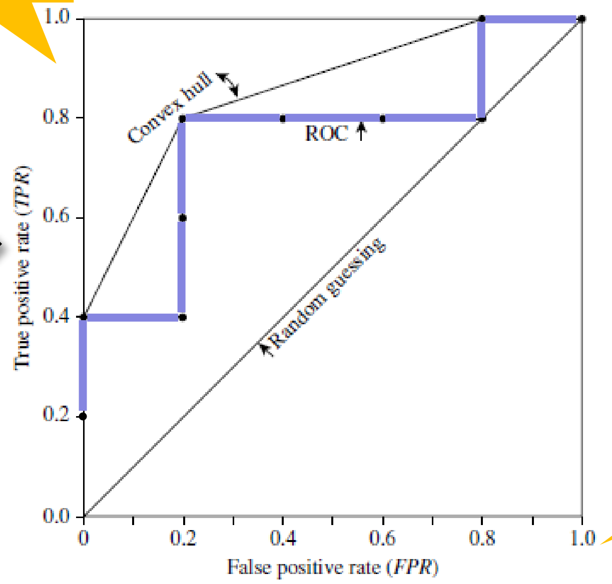
- Differen thresholds lead to different metric valuess.
- This requires us to generate different confusion matrixes ☹

ROC Curves

- **Receiver Operating Characteristics** 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

Sensitivity

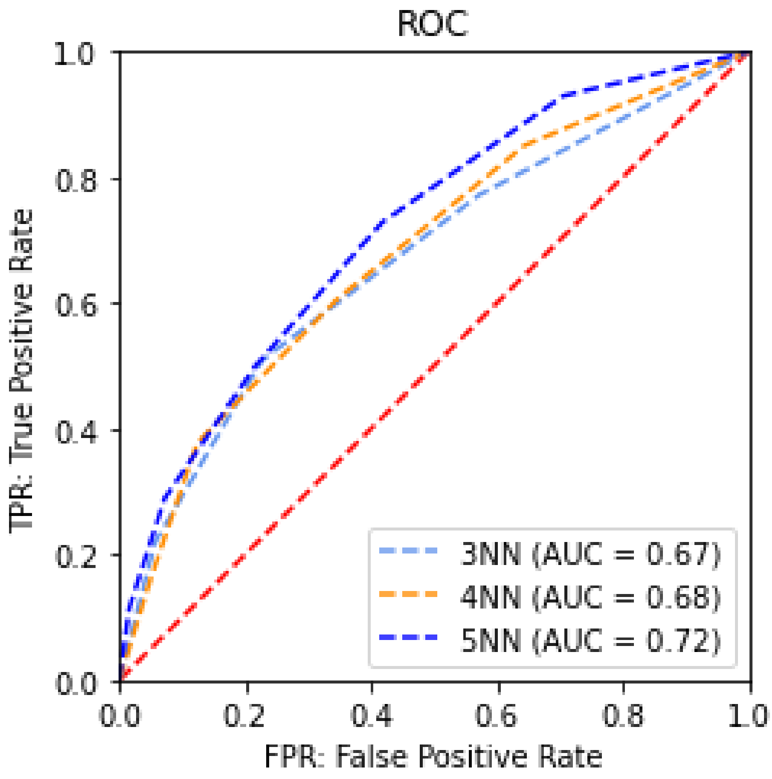
Tuple #	Class	Prob.	TP	FP	TN	FN	TPR	FPR
1	P	0.90	1	0	5	4	0.2	0
2	P	0.80	2	0	5	3	0.4	0
3	N	0.70	2	1	4	3	0.4	0.2
4	P	0.60	3	1	4	2	0.6	0.2
5	P	0.55	4	1	4	1	0.8	0.2
6	N	0.54	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.51	4	4	1	1	0.8	0.8
9	P	0.50	5	4	0	1	1.0	0.8
10	N	0.40	5	5	0	0	1.0	1.0



1- specificity

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



Continued Example in Jupyter Notebook

- Diabetes dataset
  - 768 data objects of 9 columns/attributes
  - Available in Moodle
- ROC for classification
- Lecture5_KNN_diabetes.ipynb
  - predict_proba(X_test)
  - roc_curve(y_test, pred_prob)
  - auc(fpr, tpr)
  - Plot ROC

Summary of Today

- KNN
  - No real training step
- Data scaling
  - Necessary when distances are involved in modelling and columns are of different scales
- Model evaluation
  - Holdout, Cross-validation (k-fold)
  - ROC and AUC

References

- Mandatory reading
  - Muller and Guido: Introduction to Machine Learning with Python, O'Reilly, 2016
    - Chapter 2: k-Nearest Neighbors
    - Chapter 3: Preprocessing and Scaling
    - Chapter 5: Cross-Validation
- Further reading
  - ROC and AUC
    - https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/
    - https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
  - Even further readings (*NB*: Only if you're interested in theory)
    - Jiawei Han, Micheline Kamber and Jian Pei. Data Mining: Concepts and Techniques (3rd edition), Elsevier Science Ltd, 2011.
      - Chapter 8

Exercises (1)

Using the Titanic dataset (available in Moodle), do the following in Lecture5-Exercise_Titanic_template.ipynb (available in Moodle)

1. Obtain a reduced dataset *D* that only contains the following features
   - Survived, Pclass, Sex and Age
   - *NB*: Data preprocessing is needed, e.g., transform to numerals, imputing missing values (NA)
2. Use a default KNN (K=5) to see the effect of data scaling (with vs. without).
3. Try different K's (2 to 8) for KNN, validate each classifier using stratified 3-fold.
4. Plot the ROC with AUC for each model in Step 3.

Exercises (2)

- Using the 3-class Iris flowers dataset (in Moodle), do the following in Jupyter Notebook
    1. Using 10%, 20% and 30% of the data for validation, do the following
        1. Build a default KNN (K=5) classifier and validate it using the validation data.
    2. Use Cross-Validation (5-fold) to decide a best K value from 3 to 10
        1. Standard
        2. Stratified
        3. LeaveOut

- NB: use Lecture5_Exercise_iris_template.ipynb template in Moodle.

Iris Versicolor **Iris Setosa** **Iris Virginica**

http://www.lac.inpe.br/~rafael.santos/Docs/CAP394/WholeStory-Iris.html