

Machine & Deep Learning

Pr Rahhal ERRATTAHI
rahhal.errattahi@um6p.ma
errattahi.r@ucd.ac.ma

Lecture 02

Supervised ML problem setup and Data Preprocessing

Lecture 2 Overview

- Supervised ML problem setup
- Loss Function
- Data Pre-processing
- Model Evaluation
- Nearest Neighbours and KNN

Lecture 2 Overview

- Supervised ML problem setup
- Loss Function
- Data Pre-processing
- Model Evaluation
- Nearest Neighbours and KNN

Supervised ML problem setup

- **Classification**: Classification is the process of predicting the class of given data points. Classes are sometimes called as targets / labels or categories.
- **Target function**: The target function $f : X \rightarrow Y$ is the function f that we want to model. It maps data points to targets / labels.
- **Machine Learning Model / Classifier / Hypothesis**: A Machine Learning Model is the learned program / function that maps inputs to outputs / predictions. For example: **decision tree is a classifier** or this can be a **set of weights** for a linear model or for a neural network.

Supervised ML problem setup

Problem Setting:

- Set of possible instances X i.e. $\{ \langle x_i, y_i \rangle \}$
- Dataset D , given by $D = \{ \langle \vec{x}_i, y_i \rangle, \dots, \langle \vec{x}_n, y_n \rangle \} \subseteq X \times Y$

Where:

- \vec{x}_i is a feature vector (\mathbb{R}^d),
- y_i is a label / target variable,
- X is space of all features and
- Y is space of labels.
- Unknown target function $f : X \rightarrow Y$
- Set of function hypotheses $H = \{ h | h : X \rightarrow Y \}$

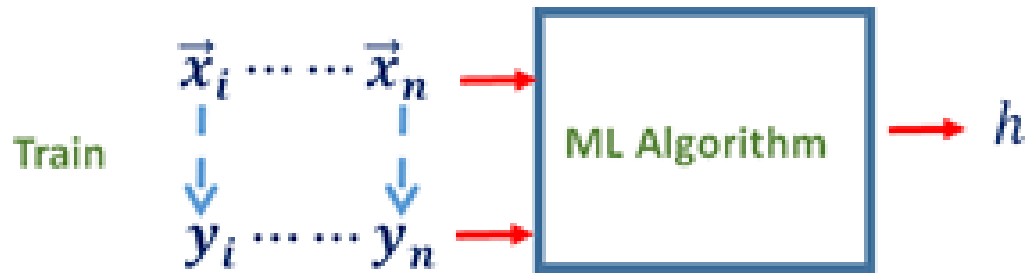
Supervised ML problem setup

Output:

- Hypothesis $h \in H$ that best approximates target function f . Or a classification “rule” that can determine the class of any object from its attributes values.
- If training is done correctly $h(\vec{x}_i) \approx y_i$

Supervised ML problem setup

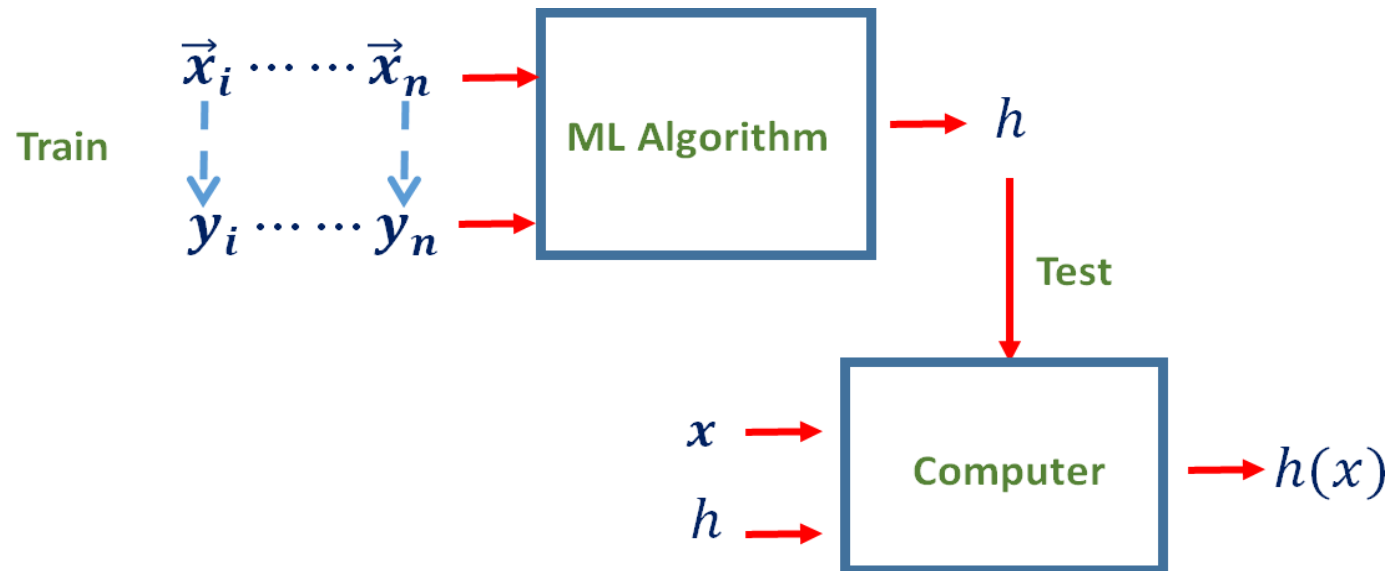
Test / Train setup:



- Aim is that algo. should learn to map
- If **training** is done correctly
 $h(\vec{x}_i) \approx y_i$

Supervised ML problem setup

Test / Train setup:



- For test, take \vec{x}_i whose label is unknown.
- Then computer passes that \vec{x}_i to h to make **prediction** on unknown data.
- It will only work if train and test data are drawn from the **same distribution**.

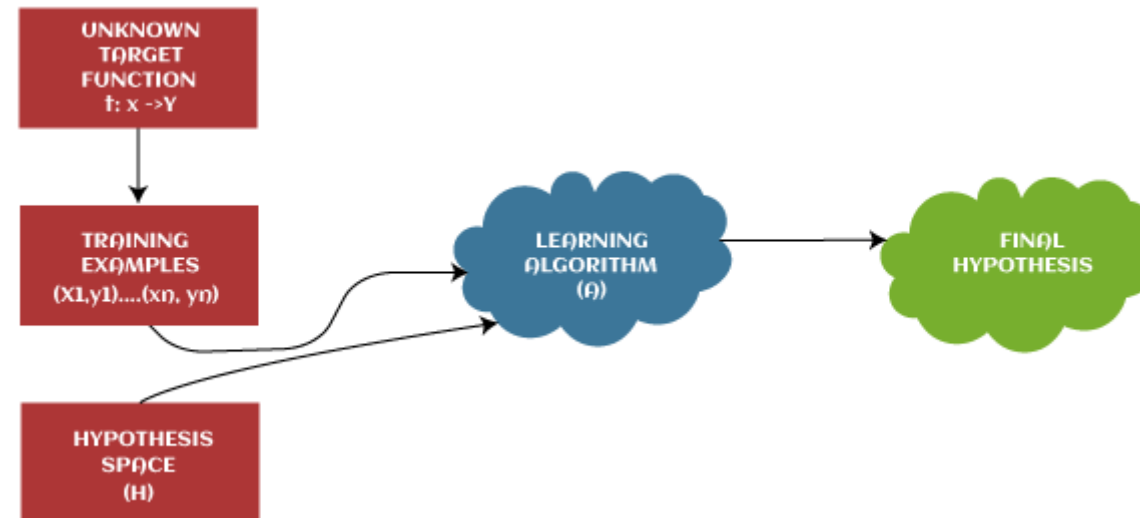
Supervised ML problem setup

- Hypothesis $h \in H$ that best approximates target function f .
- Before we can find a function h from infinite many possibilities H , we must specify what type of function it is that we are looking for. It could be:
 - Decision Tree
 - Nearest Neighbor
 - SVM
 - ANN
 - Bayesian classifier
 - ...
- There is **NO** best algorithm. It all depends on the problem and on the data.

Supervised ML problem setup

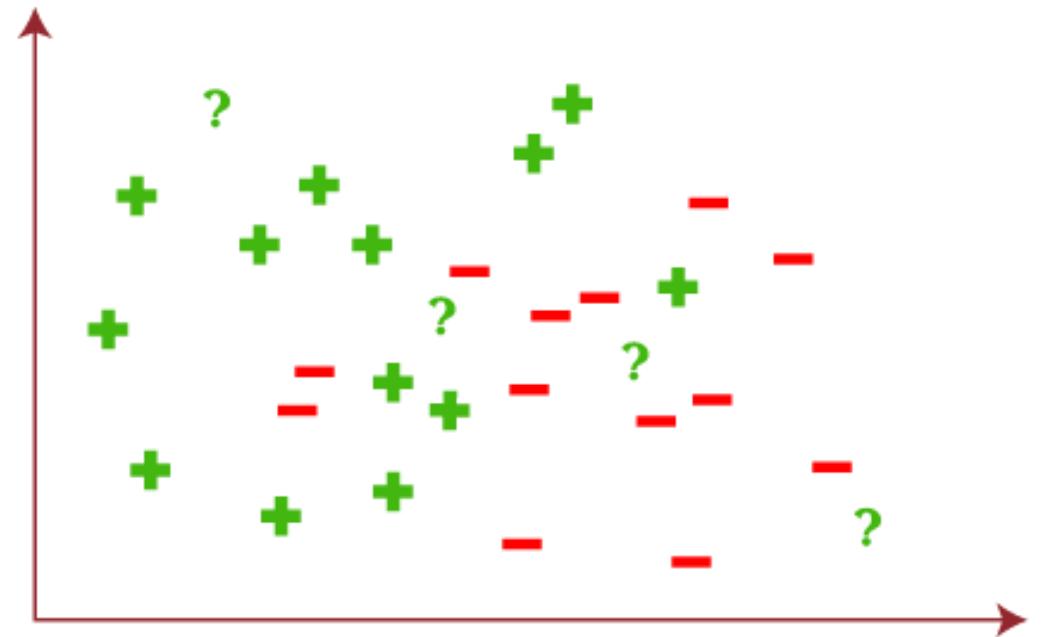
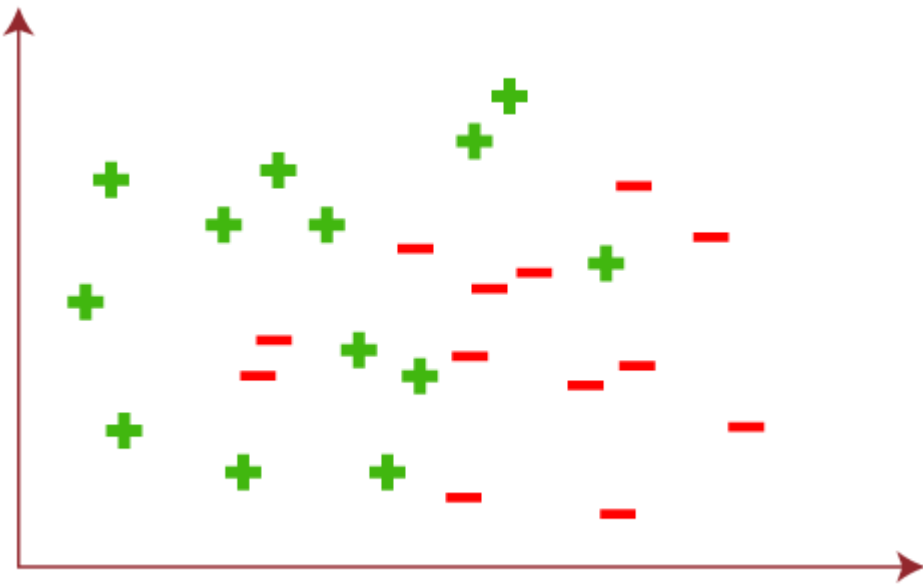
How to select $h \in H$?

- Essentially, we try to find a function h within the hypothesis class that makes the fewest mistakes within training data.



Supervised ML problem setup

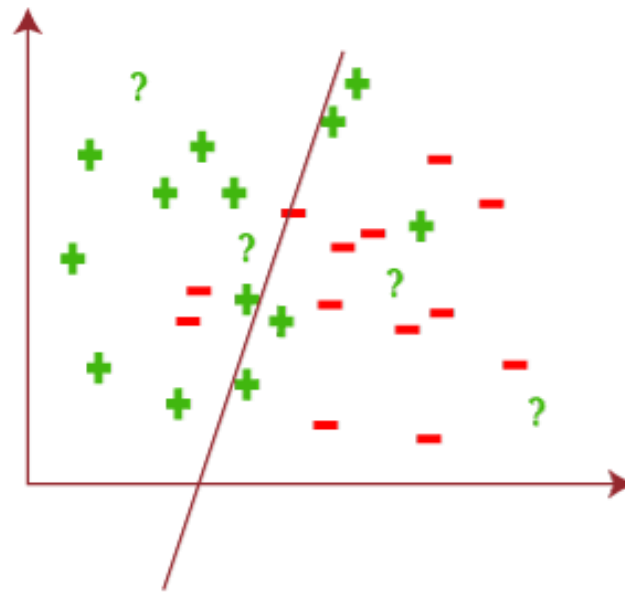
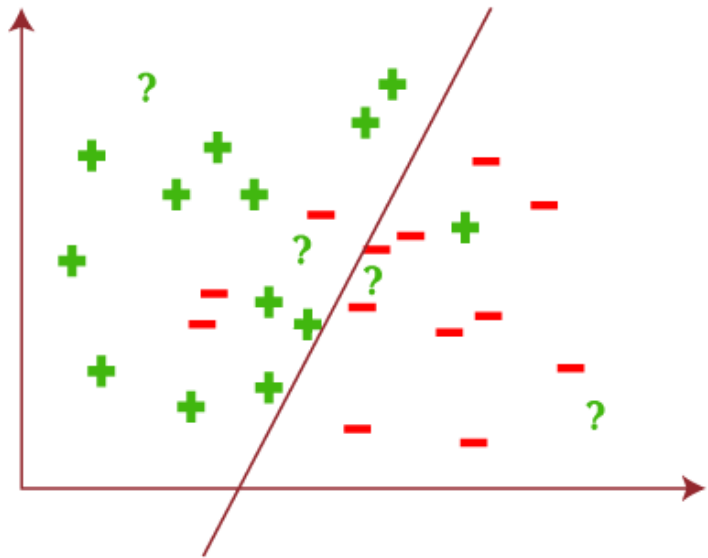
Example: Let's understand the hypothesis (h) and hypothesis space (H) with a two-dimensional coordinate plane showing the distribution of data as follows:



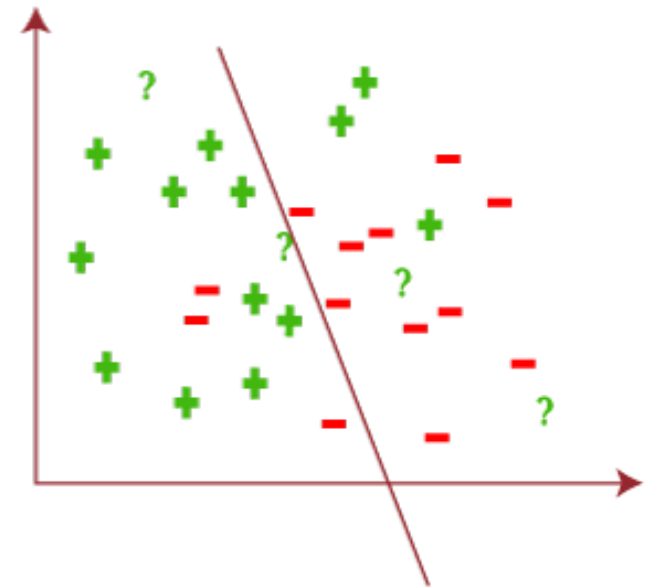
Now, assume we have some test data by which ML algorithms predict the outputs for input as follows:

Supervised ML problem setup

Example: Based on data, algorithm, and constraints, this coordinate plane can be divided in the following ways as follows:



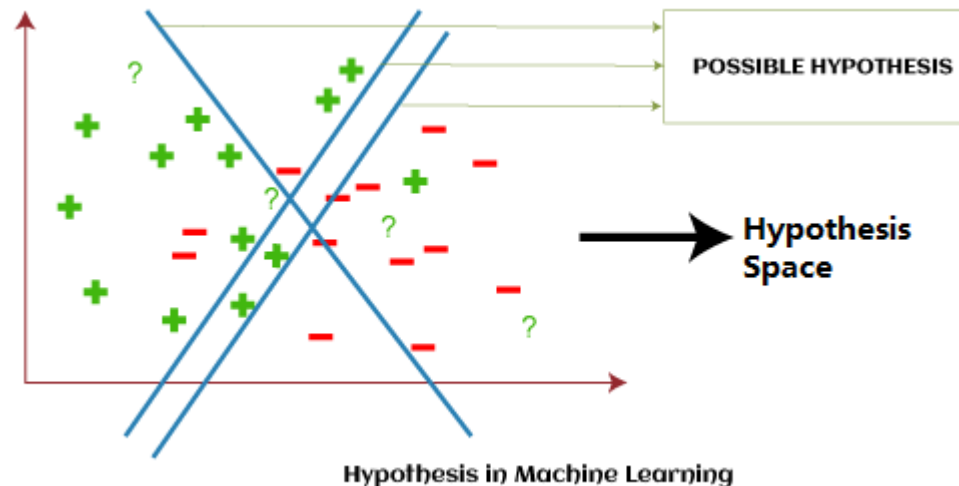
or



Supervised ML problem setup

Example:

- Hypothesis space (H) is the composition of all legal best possible ways to divide the coordinate plane so that it best maps input to proper output.
- Further, each individual best possible way is called a hypothesis (h). Hence, the hypothesis and hypothesis space would be like this:

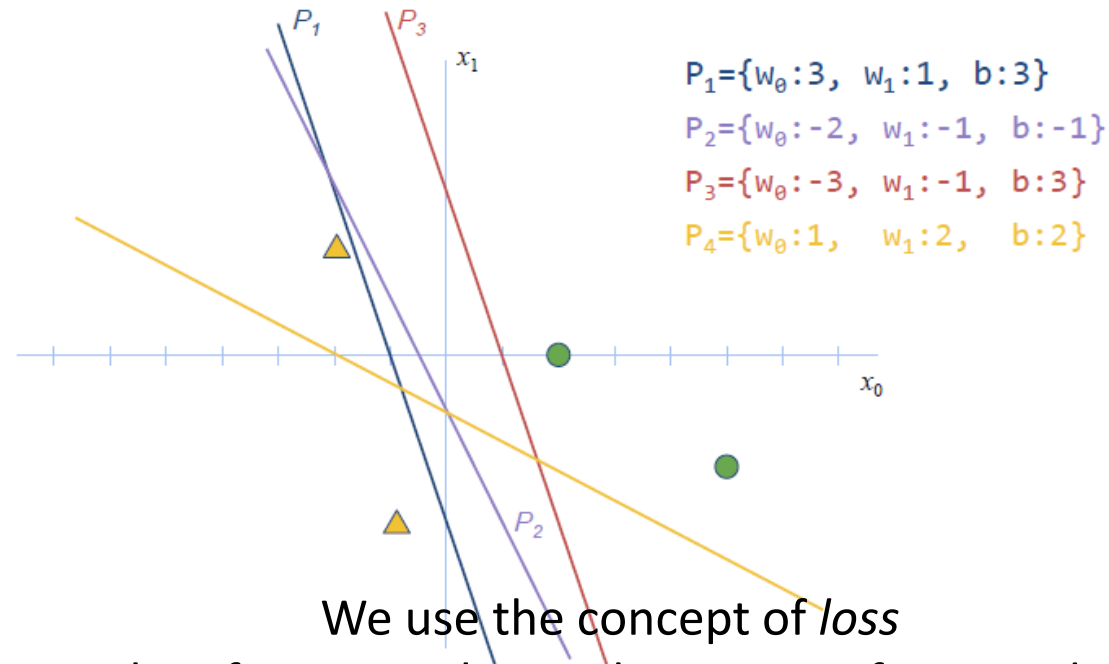


Lecture 2 Overview

- Supervised ML problem setup
- **Loss Function**
- Data Pre-processing
- Model Evaluation
- Nearest Neighbours and KNN

Loss Function

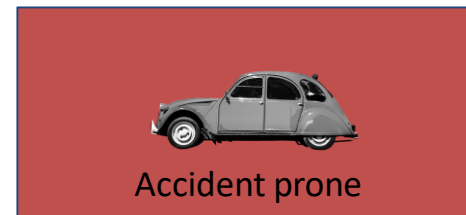
- Given a set of Hypothesis $H=\{h_1, h_2, \dots\}$, how do you know which one to use?



We use the concept of *loss*
A loss function takes in the output of our model,
compares it to the true value and then gives us a
measure of how “far” our model is.

Loss Function

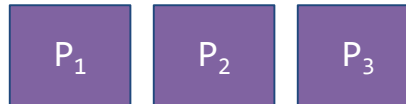
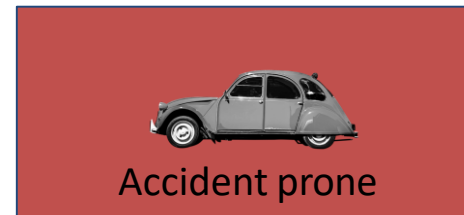
A loss function is any function that gives a measure of how far your scores are from their true values Loss Function



$[1.0, 0.0]$ \leftarrow True values \rightarrow $[0.0, 1.0]$

Loss Function

Consider two cars and three sets of parameters



Which set of parameters is the best?

$$f(\text{Sports Car}, P_1) = [0.5, 0.5]$$

$$f(\text{Sports Car}, P_2) = [0.7, 0.3]$$

$$f(\text{Sports Car}, P_3) = [0.1, 0.9]$$

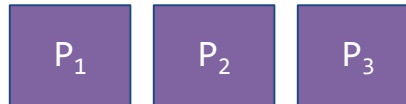
$$f(\text{Beetle}, P_1) = [0.1, 0.9]$$

$$f(\text{Beetle}, P_2) = [0.3, 0.7]$$


$$f(\text{Beetle}, P_3) = [0.9, 0.1]$$


Loss Function


Consider two cars and three sets of parameters





Which set of parameters is the best?



$$f(\text{car}, P_1) = [0.5, 0.5]$$


$$f(\text{car}, P_1) = [0.1, 0.9]$$


$$f(\text{car}, P_2) = [0.7, 0.3]$$


$$f(\text{car}, P_2) = [0.3, 0.7]$$


$$f(\text{car}, P_3) = [0.1, 0.9]$$


$$f(\text{car}, P_3) = [0.9, 0.1]$$

Confused
model

Less confident but
correct model

Very confident but
wrong model

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

$$\begin{aligned} L(\text{🏎️}, P_1) &= \text{sum}(\text{f}(\text{🏎️}, P_1) - [1.0, 0.0]) \\ &= \text{sum}([|-0.5|, |0.5|]) = 1 \end{aligned}$$

$$\begin{aligned} L(\text{🐛}, P_1) &= \text{sum}(\text{f}(\text{🐛}, P_1) - [0.0, 1.0]) \\ &= \text{sum}([|0.1|, |-0.1|]) = 0.2 \end{aligned}$$

$$L = \frac{1}{n} \sum_{i=1}^n |\hat{y}^i - y^i|$$

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

$$\begin{aligned} L(\text{Ford Mustang}, P_1) &= \text{sum}(f(\text{Ford Mustang}, P_1) - [1.0, 0.0]) \\ &= \text{sum}([|-0.5|, |0.5|]) = 1 \end{aligned}$$

$$\begin{aligned} L(\text{VW Beetle}, P_1) &= \text{sum}(f(\text{VW Beetle}, P_1) - [0.0, 1.0]) \\ &= \text{sum}([|0.1|, |-0.1|]) = 0.2 \end{aligned}$$

$$L(\text{Ford Mustang}, P_2) = 0.6$$

$$L(\text{Ford Mustang}, P_3) = 1.8$$

$$L(\text{VW Beetle}, P_2) = 0.6$$

$$L(\text{VW Beetle}, P_3) = 1.8$$

Loss Function

A potential loss function in this case is the *sum of the absolute difference* of scores:

$$\begin{aligned} L(\text{Ford Mustang}, P_1) &= \text{sum}(f(\text{Ford Mustang}, P_1) - [1.0, 0.0]) \\ &= \text{sum}([|-0.5|, |0.5|]) = 1 \end{aligned}$$

$$\begin{aligned} L(\text{VW Beetle}, P_1) &= \text{sum}(f(\text{VW Beetle}, P_1) - [0.0, 1.0]) \\ &= \text{sum}([|0.1|, |-0.1|]) = 0.2 \end{aligned}$$

$$L(\text{Ford Mustang}, P_2) = 0.6$$

$$L(\text{VW Beetle}, P_2) = 0.6$$

$$L(\text{Ford Mustang}, P_3) = 1.8$$

$$L(\text{VW Beetle}, P_3) = 1.8$$

Average loss for both cars

$$L(P_1) = 0.6 \quad L(P_2) = 0.6 \quad L(P_3) = 1.8$$

Loss Function

Average loss for both cars

$$L(P_1) = 0.6 \quad L(P_2) = 0.6 \quad L(P_3) = 1.8$$

A lower value of the loss indicates a better model

i.e. we are closer to the true values

In this case, P_1 and P_2 have the *lower value* of 0.6, so we know they are better than P_3 . However, we also know that P_2 is better than P_1 , and this implies our loss function is not very good right now!

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

$$L = \frac{1}{n} \sum_{i=1}^n (\hat{y}^i - y^i)^2$$

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores



$$f(\text{car}, P_1) = [0.5, 0.5]$$



$$f(\text{car}, P_2) = [0.7, 0.3]$$



$$f(\text{car}, P_3) = [0.1, 0.9]$$



$$f(\text{car}, P_1) = [0.1, 0.9]$$



$$f(\text{car}, P_2) = [0.3, 0.7]$$



$$f(\text{car}, P_3) = [0.9, 0.1]$$

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

$$\text{MSE}(\text{Ford Mustang}, P_1) = 0.50$$

$$\text{MSE}(\text{Ford Mustang}, P_2) = 0.18$$

$$\text{MSE}(\text{Ford Mustang}, P_3) = 1.62$$

$$\text{MSE}(\text{VW Beetle}, P_1) = 0.02$$

$$\text{MSE}(\text{VW Beetle}, P_2) = 0.18$$

$$\text{MSE}(\text{VW Beetle}, P_3) = 1.62$$

Loss Function

Better loss function:

Mean Squared Error

Loss is equal to the sum of the square of the differences in the scores

$$\text{MSE}(\text{Ford Mustang}, P_1) = 0.50$$

$$\text{MSE}(\text{Ford Mustang}, P_2) = 0.18$$

$$\text{MSE}(\text{Ford Mustang}, P_3) = 1.62$$

$$\text{MSE}(\text{VW Beetle}, P_1) = 0.02$$

$$\text{MSE}(\text{VW Beetle}, P_2) = 0.18$$

$$\text{MSE}(\text{VW Beetle}, P_3) = 1.62$$

Average loss

$$L(P_1) = 0.26$$

$$L(P_2) = 0.18$$

$$L(P_3) = 1.62$$

Loss Function

Mean Squared Error works better, as it penalizes values that are further away from the true value

Loss Function

Many other choices for loss functions:

- Absolute Distance loss
- Hinge loss
- Logistic loss
- Cross Entropy loss
- \vdots

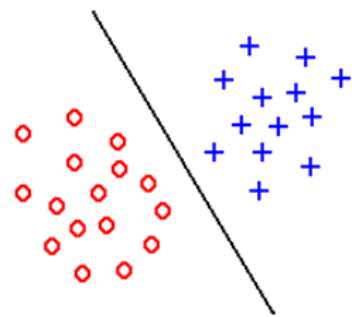
Lab1: Data manipulation demo

Lecture 2 Overview

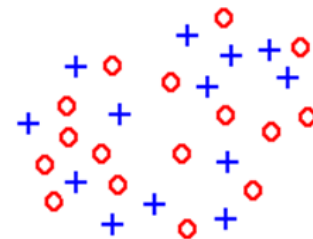
- Supervised ML problem setup
- Loss Function
- **Data Pre-processing**
- Model Evaluation
- Nearest Neighbours and KNN

Features quality

- **Feature** is an individual measurable property or characteristic of a phenomenon being observed.
- Good features makes it easy for classifier to decide (learn) between two different classes / concepts / labels OR **good features enhances inter class variations while minimize intra class variation.**



"Good" features

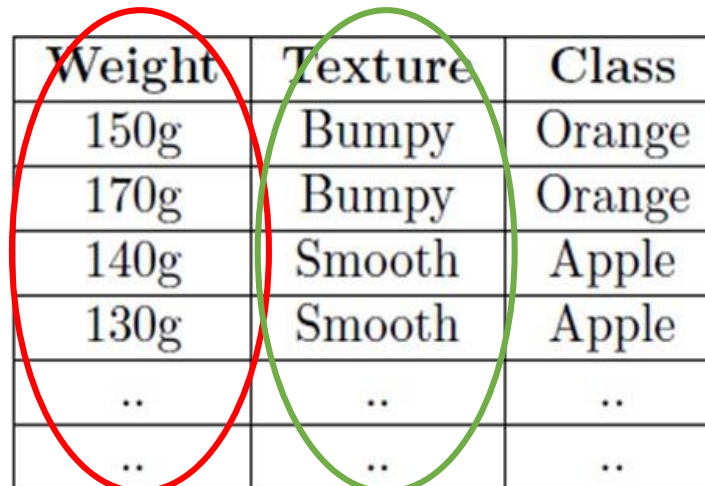


"Bad" features

Feature types

Mainly feature variable can have **two distinct types**:

- **Numerical** variable / feature : is a type of data that is expressed in terms of numbers rather than natural language descriptions
- **Categorical** variable / feature : is a type of data that can be stored into groups or categories



Weight	Texture	Class
150g	Bumpy	Orange
170g	Bumpy	Orange
140g	Smooth	Apple
130g	Smooth	Apple
..
..

Feature types

- **Numerical:**
 - **Continuous:** Observations can take any value between a certain set of real numbers.
 - **Discrete:** Observations can take a value based on a count from a set of distinct whole values. A discrete variable cannot take the value of a fraction between one value and the next closest value.

Feature types

- **Categorical:**

- **Ordinal:** Observations can take a value that can be logically ordered or ranked. The categories associated with ordinal variables can be ranked higher or lower than another, but do not necessarily establish a numeric difference between each category e.g. short, tall.
- **Nominal:** Observations can take a value that is not able to be organized in a logical sequence e.g. the name or colour of an object. A nominal variable may be numerical in form, but the numerical values have no mathematical interpretation. E.g. label 10 people as numbers 1, 2, 3, ..., 10, but any arithmetic with such values, e.g. $1 + 2 = 3$ would be meaningless.

Lecture 2 Overview

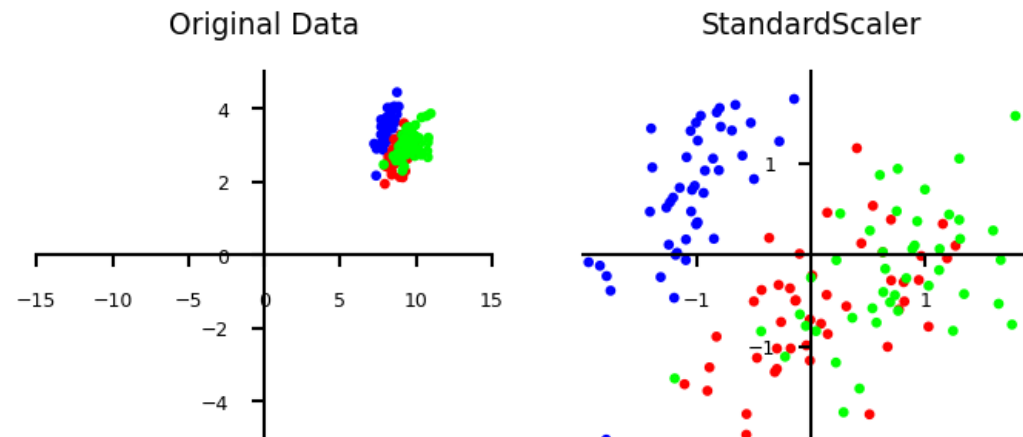
- Supervised ML problem setup
- Loss Function
- **Data Pre-processing**
 - Data transformations
 - Automatic Feature Selection
 - Missing value imputation
 - Handling imbalanced data
- Model Evaluation
- Nearest Neighbours and KNN

Data transformations

- Machine learning models make a lot of assumptions about the data
- In reality, these assumptions are often violated
- We build pipelines that transform the data before feeding it to the learners
 - Scaling (or other numeric transformations)
 - Encoding (convert categorical features into numerical ones)
 - Automatic feature selection
 - Feature engineering (e.g. binning, polynomial features,...)
 - Handling missing data
 - Handling imbalanced data
 - Dimensionality reduction (e.g. PCA)
 - ...

Scaling

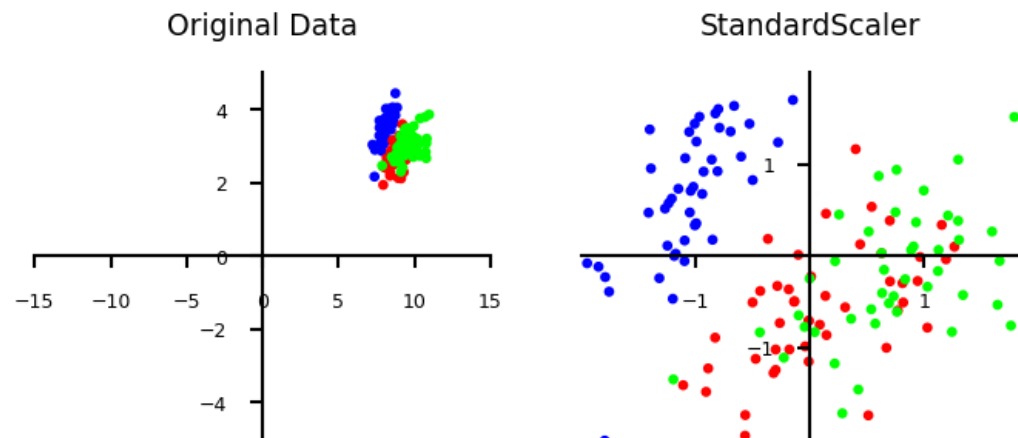
- Use when different numeric features have different scales (different range of values)
 - Features with much higher values may overpower the others
- Goal: bring them all within the same range
- Different methods exist



Standard scaling (standardization)

- Generally most useful, assumes data is more or less normally distributed
- Per feature, subtract the mean value μ , scale by standard deviation σ
- New feature has $\mu = 0$ and $\sigma = 1$, values can still be arbitrarily large

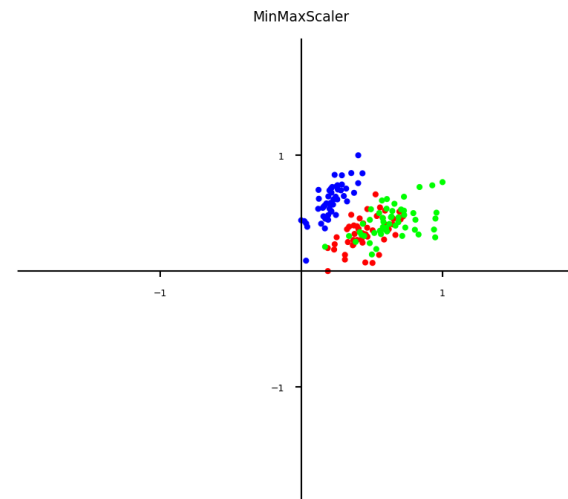
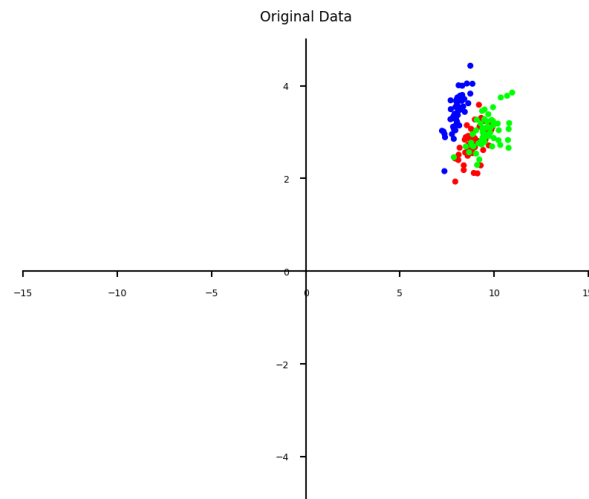
$$x_{new} = \frac{x - \mu}{\sigma}$$



Min-max scaling

- Scales all features between a given min and max value (e.g. 0 and 1)
- Makes sense if min/max values have meaning in your data
- Sensitive to outliers

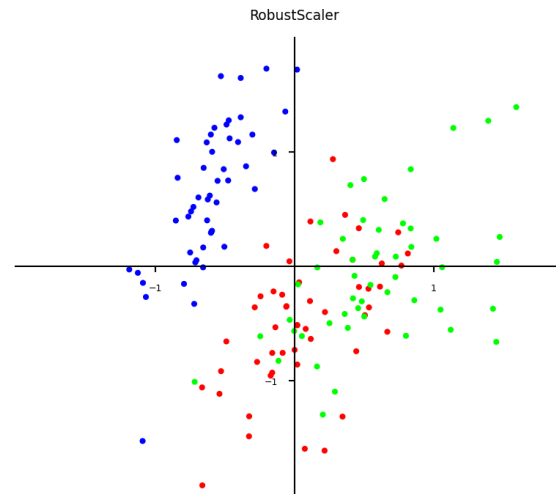
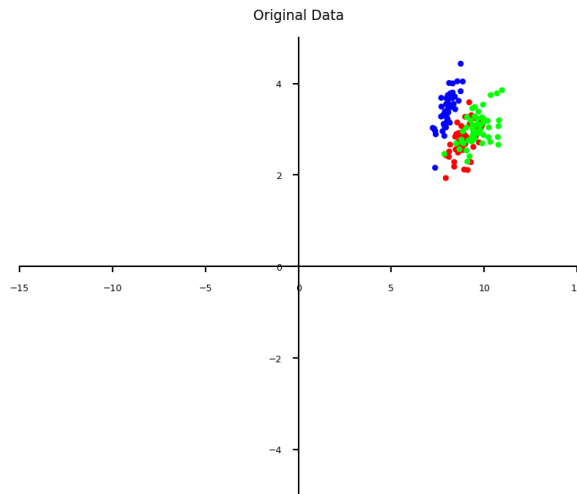
$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}} (\max - \min) + \min$$



Robust scaling

- Subtracts the median, scales between quantiles q_{25} and q_{75}
- New feature has median 0, $q_{25} = -1$ and $q_{75} = 1$
- Similar to standard scaler, but ignores outliers

$$x_{new} = \frac{(x - \mu)}{q_{75} - q_{25}}$$



Categorical feature encoding

- Many algorithms can only handle numeric features, so we need to encode the categorical ones

	boro	salary	vegan
0	Manhattan	103	0
1	Queens	89	0
2	Manhattan	142	0
3	Brooklyn	54	1
4	Brooklyn	63	1
5	Bronx	219	0

Ordinal encoding

- Simply assigns an integer value to each category in the order they are encountered
- Only really useful if there exist a natural order in categories
 - Model will consider one category to be 'higher' or 'closer' to another

	boro	boro_ordinal	salary
0	Manhattan	2	103
1	Queens	3	89
2	Manhattan	2	142
3	Brooklyn	1	54
4	Brooklyn	1	63
5	Bronx	0	219

One-hot encoding

- Simply adds a new 0/1 feature for every category, having 1 (hot) if the sample has that category
- Can explode if a feature has lots of values, causing issues with high dimensionality
- What if test set contains a new category not seen in training data?
 - Either ignore it (just use all 0's in row), or handle manually (e.g. resample)

	boro	boro_Bronx	boro_Brooklyn	boro_Manhattan	boro_Queens	salary
0	Manhattan	0	0	1	0	103
1	Queens	0	0	0	1	89
2	Manhattan	0	0	1	0	142
3	Brooklyn	0	1	0	0	54
4	Brooklyn	0	1	0	0	63
5	Bronx	1	0	0	0	219

Applying data transformations

- Data transformations should always follow a fit-predict paradigm
 - Fit the transformer on the training data only
 - E.g. for a standard scaler: record the mean and standard deviation
 - Transform (e.g. scale) the training data, then train the learning model
 - Transform (e.g. scale) the test data, then evaluate the model
- Only scale the input features (X), not the targets (y)

Applying data transformations

- If you fit and transform the whole dataset before splitting, you get data leakage
 - You have looked at the test data before training the model
 - Model evaluations will be misleading
- If you fit and transform the training and test data separately, you distort the data
 - E.g. training and test points are scaled differently

Lecture 2 Overview

- Supervised ML problem setup
- Loss Function
- **Data Pre-processing**
 - Data transformations
 - **Automatic Feature Selection**
 - Missing value imputation
 - Handling imbalanced data
- Model Evaluation
- Nearest Neighbours and KNN

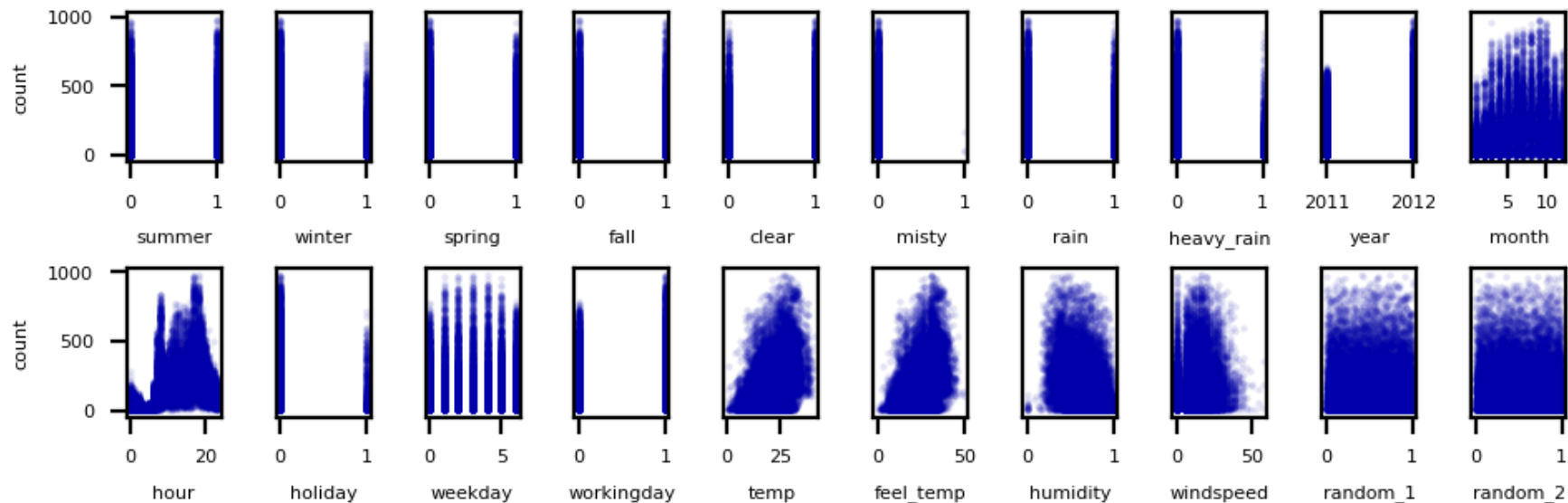
Automatic Feature Selection

It can be a good idea to reduce the number of features to only the most useful ones

- Simpler models that generalize better (less overfitting)
 - Curse of dimensionality (e.g. kNN)
 - Even models such as RandomForest can benefit from this
 - Sometimes it is one of the main methods to improve models (e.g. gene expression data)
- Faster prediction and training
 - Training time can be quadratic (or cubic) in number of features
- Easier data collection, smaller models (less storage)
- More interpretable models: fewer features to look at

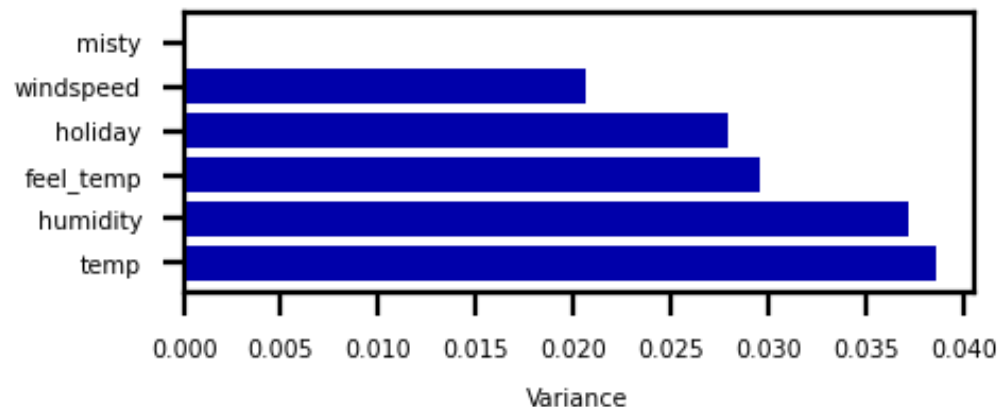
Automatic Feature Selection

- The Bike Sharing Demand dataset shows the amount of bikes rented in Washington DC
- Some features are clearly more informative than others (e.g. *temp*, *hour*)
- Some are correlated (e.g. *temp* and *feel_temp*)



Unsupervised feature selection

- Variance-based
 - Remove (near) constant features
 - Choose a small variance threshold
 - Scale features before computing variance!
 - Infrequent values may still be important



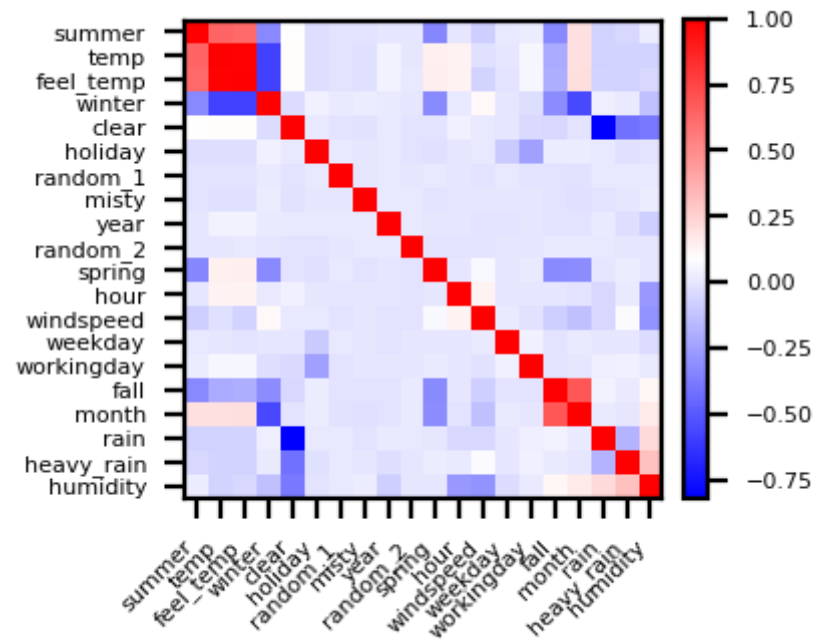
Unsupervised feature selection

- Covariance-based
 - Remove features (x_i) that are highly correlated (have high correlation coefficient ρ)
 - The small differences may actually be important
 - You don't know because you don't consider the target

$$\rho(X_1, X_2) = \frac{\text{cov}(X_1, X_2)}{\sigma(X_1)\sigma(X_2)} = \frac{\frac{1}{N-1} \sum_i (X_{i,1} - \overline{X_1})(X_{i,2} - \overline{X_2})}{\sigma(X_1)\sigma(X_2)}$$

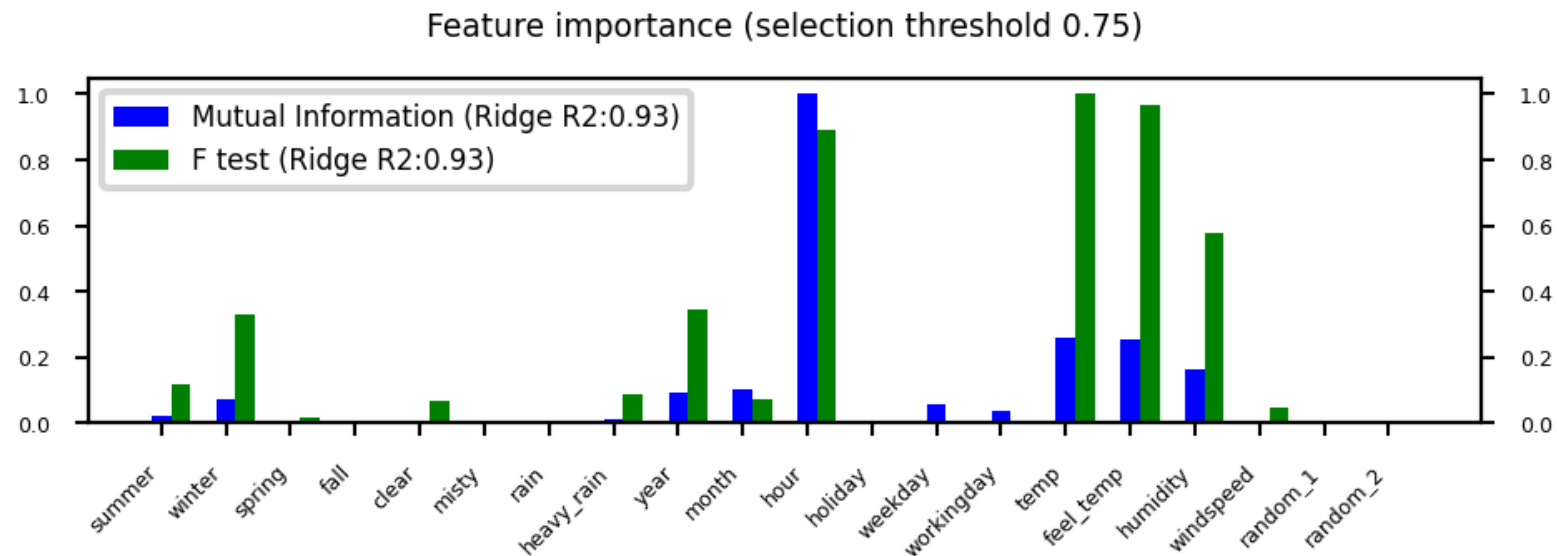
Unsupervised feature selection

- Covariance-based
 - Should we remove *feel_temp* ? Or *temp* ? Maybe one correlates more with the target?



Supervised feature selection: overview

- Univariate: F-test and Mutual Information
- Model-based: Random Forests, Linear models, kNN
- Wrapping techniques (black-box search)
- Permutation importance



Lecture 2 Overview

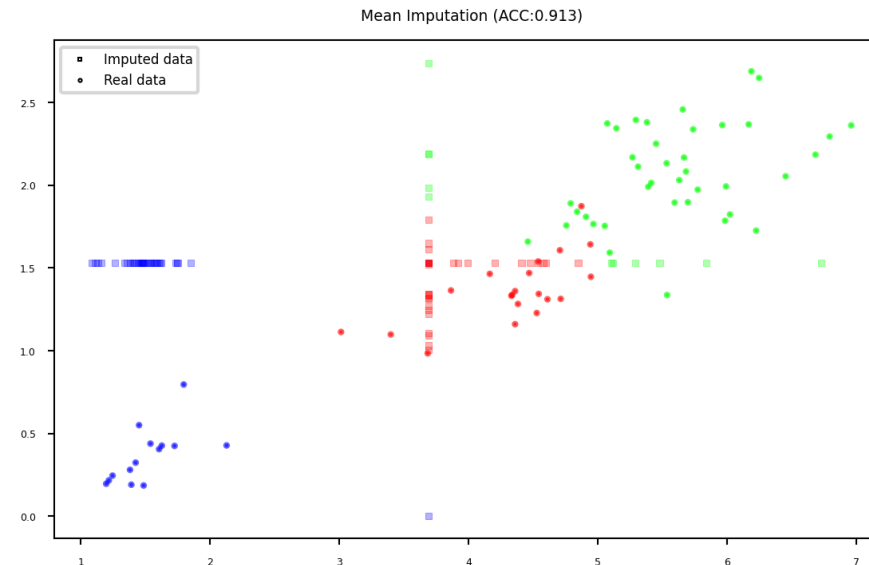
- Supervised ML problem setup
- Loss Function
- **Data Pre-processing**
 - Data transformations
 - Automatic Feature Selection
 - **Missing value imputation**
 - Handling imbalanced data
- Model Evaluation
- Nearest Neighbours and KNN

Missing value imputation

- Data can be missing in different ways:
 - Missing Completely at Random (MCAR): purely random points are missing
 - Missing at Random (MAR): something affects missingness, but no relation with the value
 - E.g. faulty sensors, some people don't fill out forms correctly
 - Missing Not At Random (MNAR): systematic missingness linked to the value
 - Has to be modelled or resolved (e.g. sensor decay, sick people leaving study)
- Missingness can be encoded in different ways: '?', '-1', 'unknown', 'NA',...
- Also labels can be missing (remove example or use semi-supervised learning)

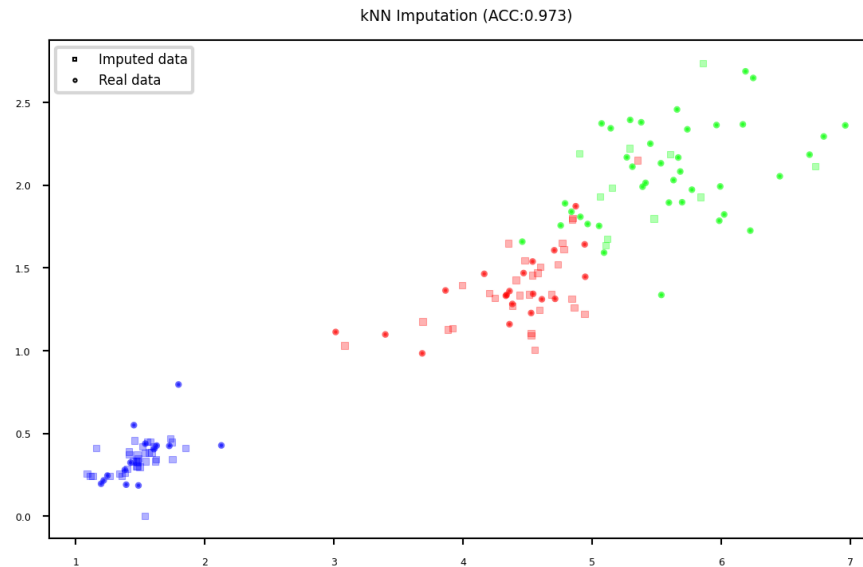
Mean imputation

- Replace all missing values of a feature by the same value
 - Numerical features: mean or median
 - Categorical features: most frequent category
 - Constant value, e.g. 0 or 'missing' for text features
- Optional: add an indicator column for missingness



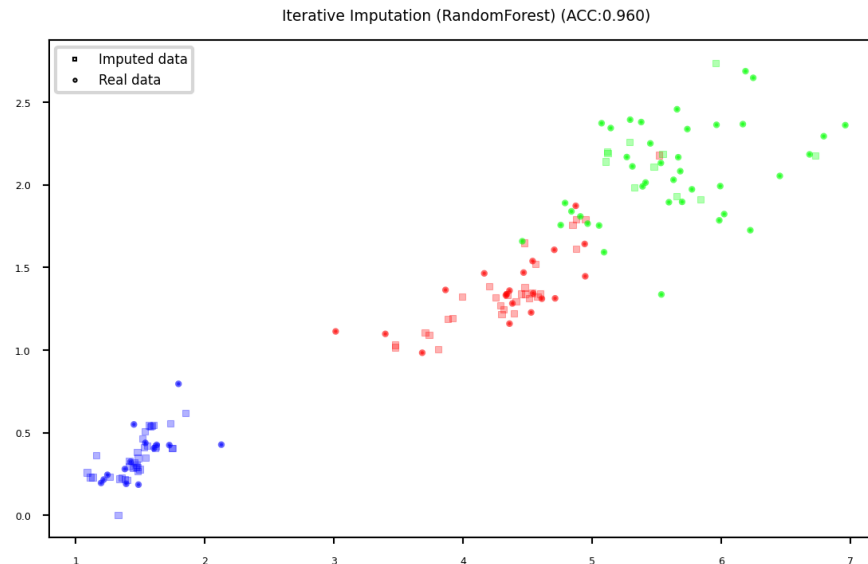
kNN imputation

- Use special version of kNN to predict value of missing points
- Uses only non-missing data when computing distances



Iterative (model-based) Imputation

- Known as Multiple Imputation by Chained Equations (MICE)
- Iterative approach
 - Do first imputation (e.g. mean imputation)
 - Train model (e.g. RandomForest) to predict missing values of a given feature
 - Train new model on imputed data to predict missing values of the next feature
 - Repeat m times in



t at a time

Lecture 2 Overview

- Supervised ML problem setup
- Loss Function
- **Data Pre-processing**
 - Data transformations
 - Automatic Feature Selection
 - Missing value imputation
 - **Handling imbalanced data**
- Model Evaluation
- Nearest Neighbours and KNN

Handling imbalanced data

- Problem:
 - You have a majority class with many times the number of examples as the minority class
 - Or: classes are balanced, but associated costs are not (e.g. FN are worse than FP)
- We already covered some ways to resolve this:
 - Add class weights to the loss function: give the minority class more weight
 - In practice: set `class_weight='balanced'`
 - Change the prediction threshold to minimize false negatives or false positives

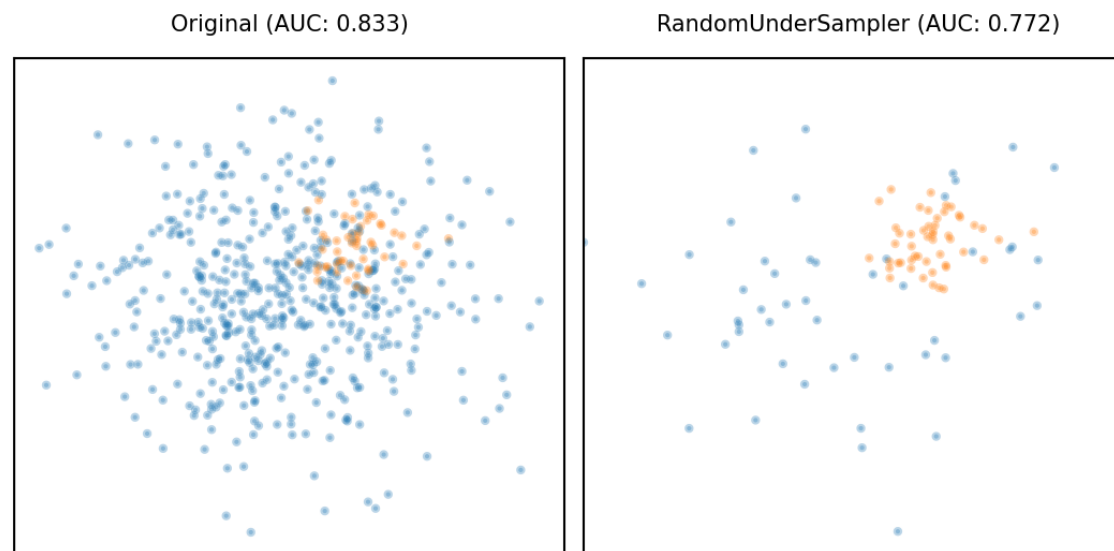
Handling imbalanced data

- There are also things we can do by preprocessing the data
 - Resample the data to correct the imbalance
 - Random or model-based
 - Generate synthetic samples for the minority class
 - Build ensembles over different resampled datasets
 - Combinations of these

Random Undersampling

- Copy the points from the minority class
- Randomly sample from the majority class (with or without replacement) until balanced
 - Optionally, sample until a certain imbalance ratio (e.g. 1/5) is reached
 - Multi-class: repeat with every other class

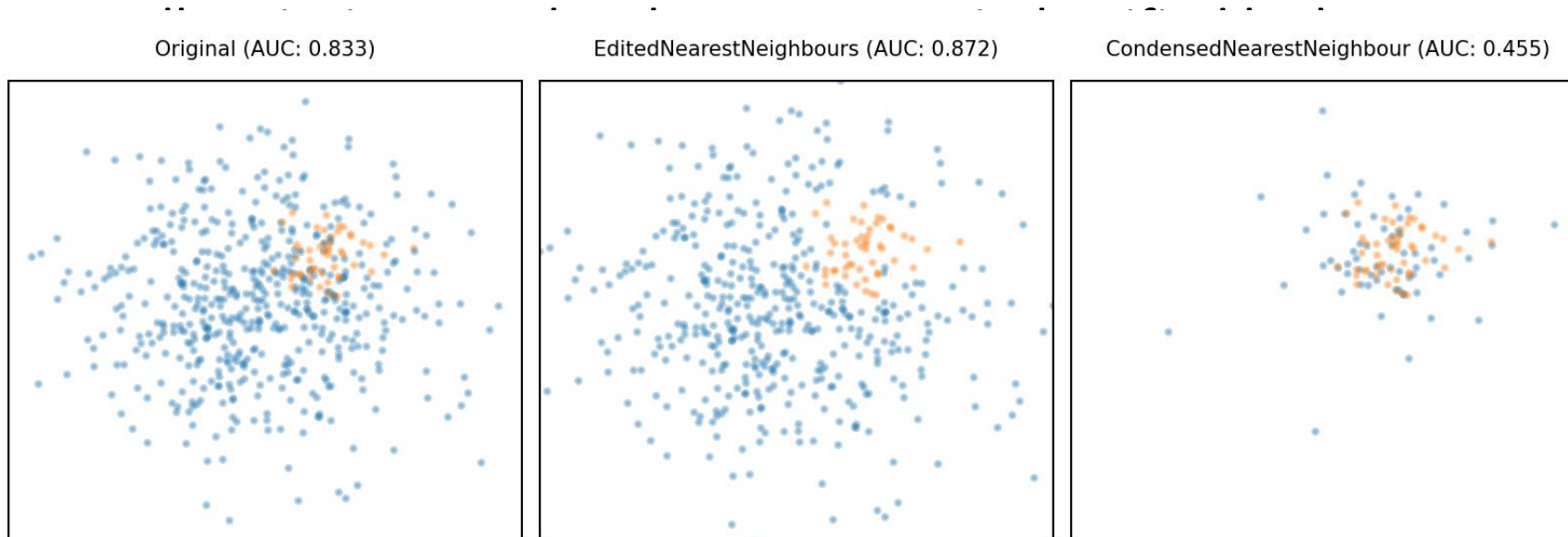
- Preferred for l₂ models with similar performance



r models with

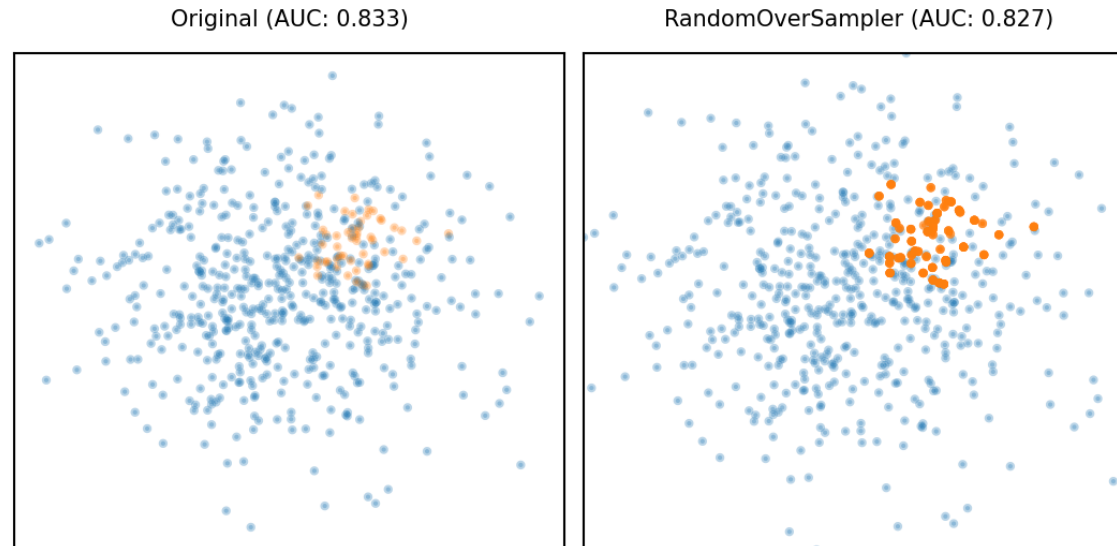
Model-based Undersampling

- Edited Nearest Neighbors
 - Remove all majority samples that are misclassified by kNN (mode) or that have a neighbor from the other class (all).
 - Remove their influence on the minority samples
- Condensed Nearest Neighbors
 - Re
 - Fo



Random Oversampling

- Copy the points from the majority class
- Randomly sample from the minority class, with replacement, until balanced
 - Optionally, sample until a certain imbalance ratio (e.g. 1/5) is reached
- Makes models more expensive to train, doesn't always improve performance
- Similar to giving more data (expensive)



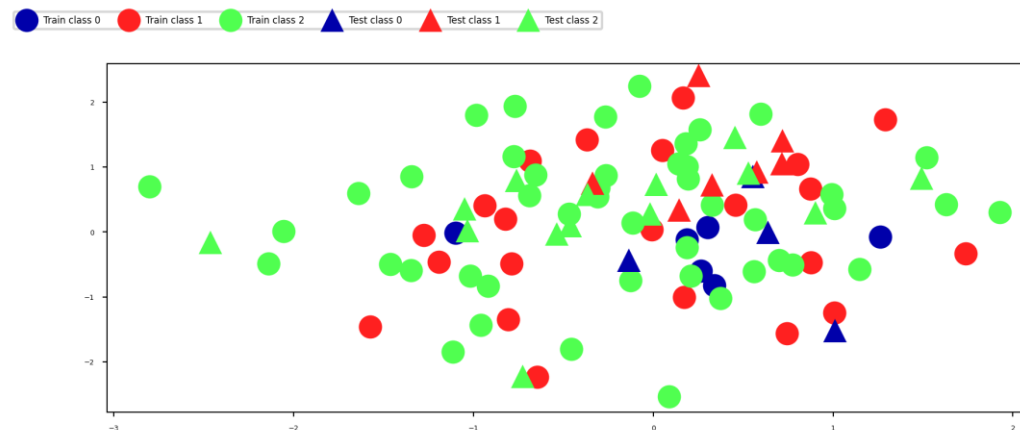
d more

Lecture 2 Overview

- Supervised ML problem setup
- Loss Function
- Data Pre-processing
- **Model Evaluation**
- Nearest Neighbours and KNN

Performance estimation techniques

- Always evaluate models as if they are predicting future data
- We do not have access to future data, so we pretend that some data is hidden
- Simplest way: the holdout (simple train-test split)
 - Randomly split data (and corresponding labels) into training and test set (e.g. 75%-25%)
 - Train (fit) a model on the training data. score on the test data

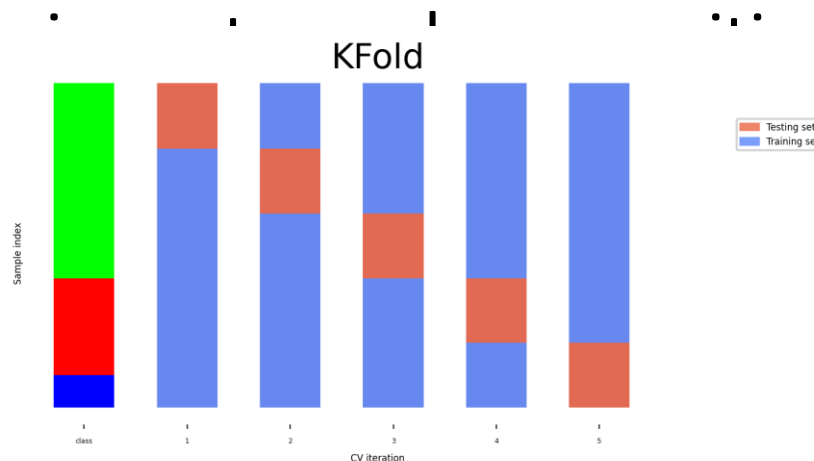


K-fold Cross-validation

- Each random split can yield very different models (and scores)
 - e.g. all easy (of hard) examples could end up in the test set
- Split data into k equal-sized parts, called folds
 - Create k splits, each time using a different fold as the test set
- Compute k evaluation scores, aggregate afterwards (e.g. take the mean)

• Examine the score variance across different splits (unstable) models are

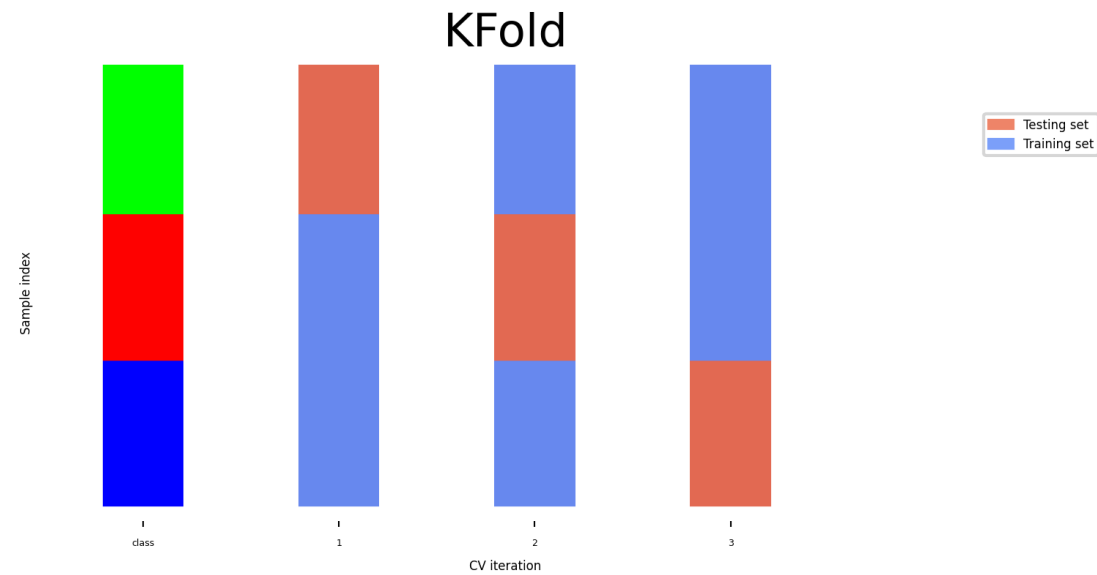
• Large k gives better estimate



... (unstable) models are (a), but is expensive

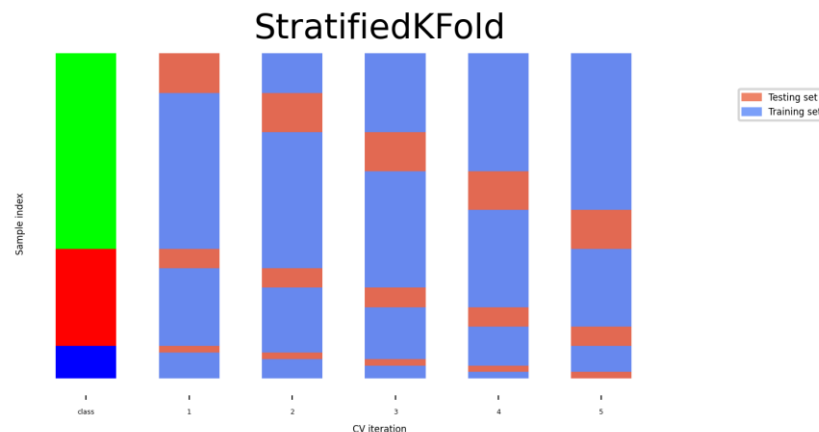
K-fold Cross-validation

- Can you guess the model's performance ?



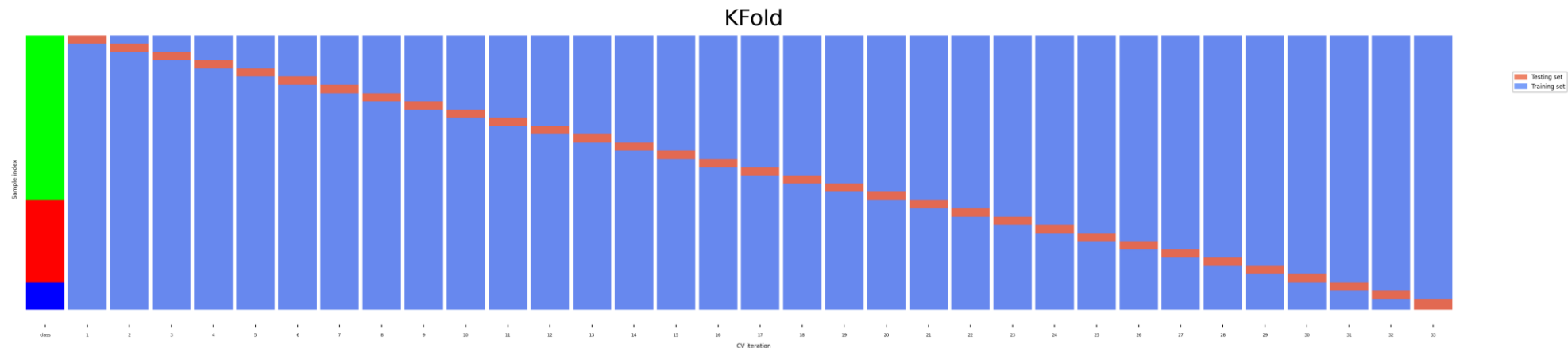
Stratified K-Fold cross-validation

- If the data is unbalanced, some classes have only few samples
- Likely that some classes are not present in the test set
- Stratification: proportions between classes are conserved in each fold
 - Order examples per class
 - Separate the samples of each class in k sets (strata)
 - Combine corresponding strata into folds



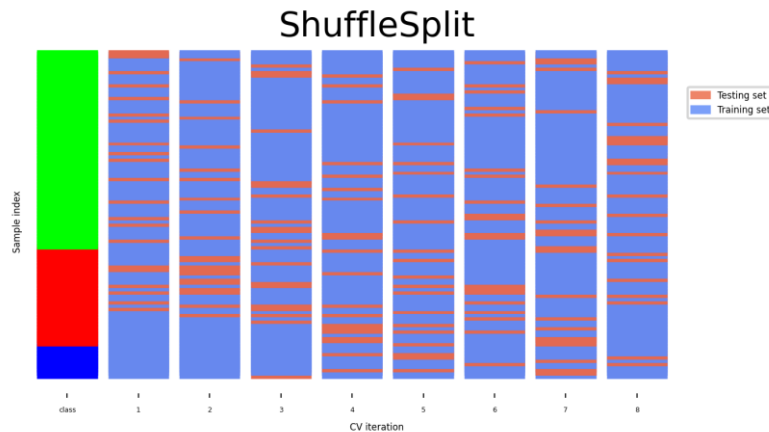
Leave-One-Out cross-validation

- k fold cross-validation with k equal to the number of samples
- Completely unbiased (in terms of data splits), but computationally expensive
- Actually generalizes less well towards unseen data
 - The training sets are correlated (overlap heavily)
 - Overfits on the data used for (the entire) evaluation
 - A different sample of the data can yield different results
- Recommended only for small datasets



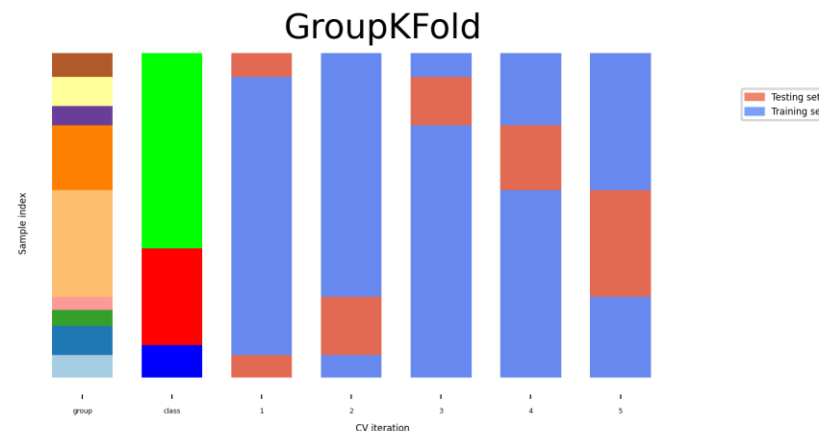
Shuffle-Split cross-validation

- Shuffles the data, samples (`train_size`) points randomly as the training set
- Can also use a smaller (`test_size`), handy with very large datasets
- Never use if the data is ordered (e.g. time series)



Cross-validation with groups

- Sometimes the data contains inherent groups:
 - Multiple samples from same patient, images from same person,...
- Data from the same person may end up in the training *and* test set
- We want to measure how well the model generalizes to *other* people
- Make sure that data from one person are in *either* the train or test set
 - This is called *grouping* or *blocking*
 - Leave-one-subject-out cross-validation: test set for each subject/group



Choosing a performance estimation procedure

No strict rules, only guidelines:

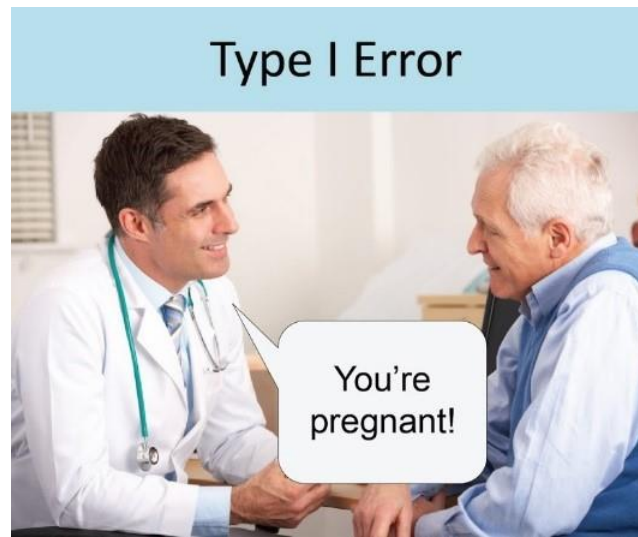
- Always use stratification for classification
- Use holdout for very large datasets (e.g. $>1.000.000$ examples)
 - Or when learners don't always converge (e.g. deep learning)
- Choose k depending on dataset size and resources
 - Use leave-one-out for very small datasets (e.g. <100 examples)
 - Use cross-validation otherwise
 - Most popular (and theoretically sound): 10-fold CV
 - Literature suggests 5x2-fold CV is better
- Use grouping or leave-one-subject-out for grouped data
- Use train-then-test for time series

Evaluation Metrics for Classification

- Each algorithm optimizes a given loss function (on the training data)
- The choice of function is limited by what can be efficiently optimized
- However, we evaluate the resulting model with a score that makes sense in the real world
 - Percentage of correct predictions (on a test set)
 - The actual cost of mistakes (e.g. in money, time, lives,...)
- We also tune the algorithm's hyperparameters to maximize that score

Binary classification

- We have a positive and a negative class
- 2 different kind of errors:
 - False Positive (type I error): model predicts positive while true label is negative
 - False Negative (type II error): model predicts negative while true label is positive
- They are not always equally important
 - Which side do you want to err on for a medical test?



Binary classification

- We can represent all predictions (correct and incorrect) in a confusion matrix
 - n by n array (n is the number of classes)
 - Rows correspond to true classes, columns to predicted classes
 - Count how often samples belonging to a class C are classified as C or any other class.
 - For binary classification, we label these true negative (TN), true positive (TP), false negative (FN), false positive (FP)

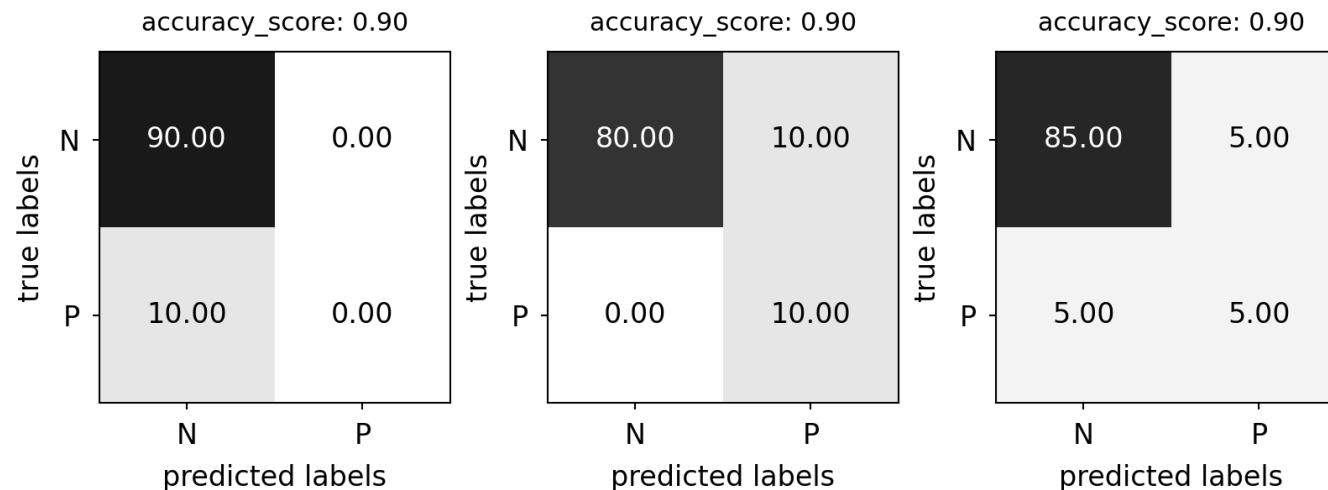
	Predicted Neg	Predicted Pos
Actual Neg	TN	FP
Actual Pos	FN	TP

Binary classification

Accuracy:

- Accuracy can be computed based on the confusion matrix
- Not useful if the dataset is very imbalanced

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$



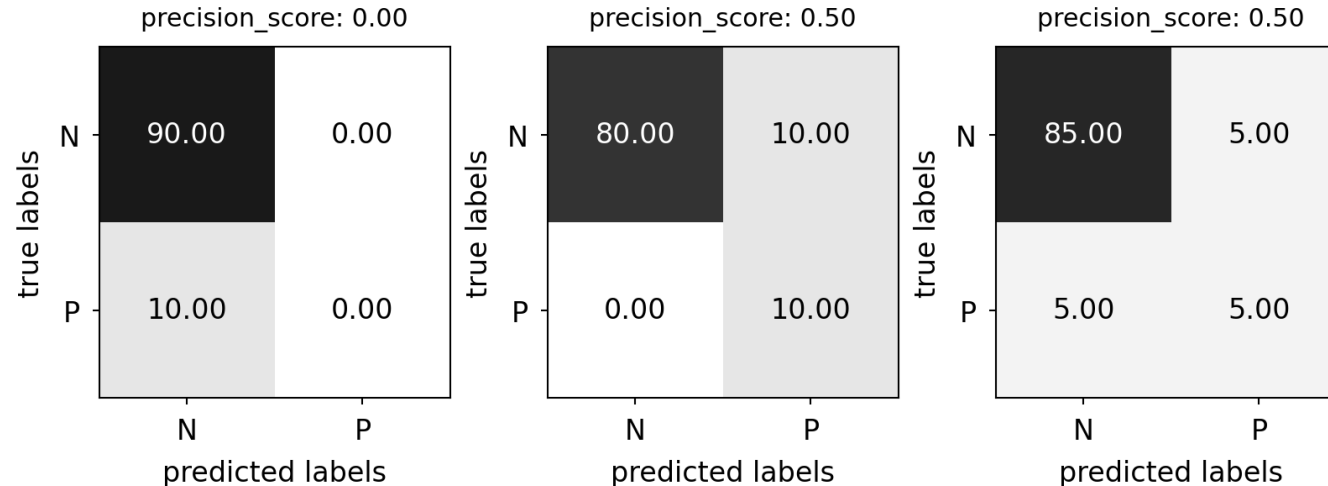
3 models: very different predictions, same accuracy:

Binary classification

Precision:

- Use when the goal is to limit FPs
 - Clinical trials: you only want to test drugs that really work
 - Search engines: you want to avoid bad search results

$$\text{Precision} = \frac{TP}{TP + FP}$$

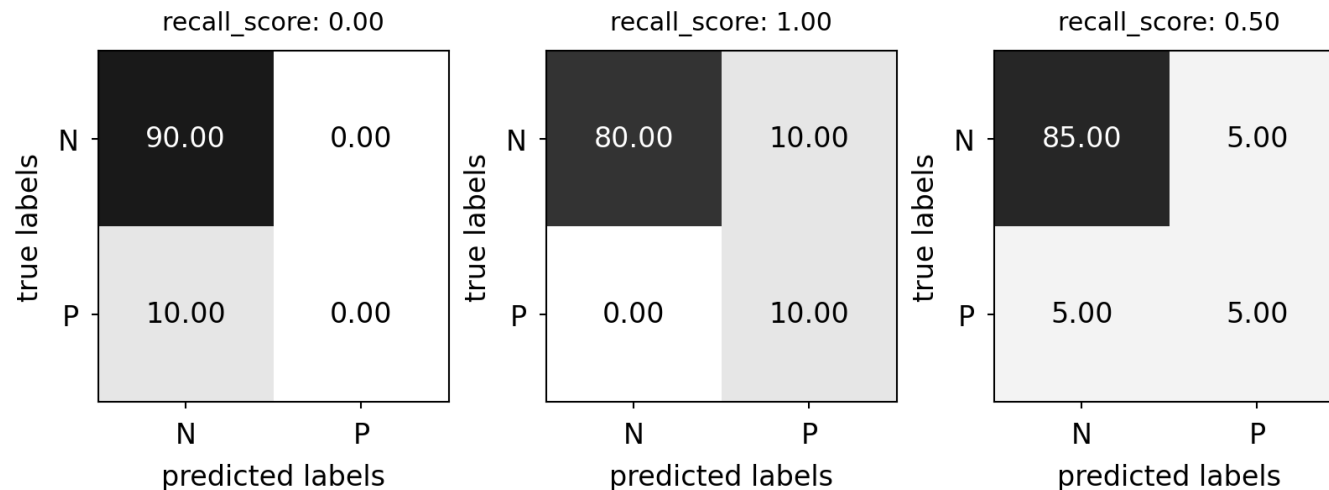


Binary classification

Recall:

- Use when the goal is to limit FNs
 - Cancer diagnosis: you don't want to miss a serious disease
 - Search engines: You don't want to omit important hits
- Also know as sensitivity, hit rate, true positive rate (TPR)

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

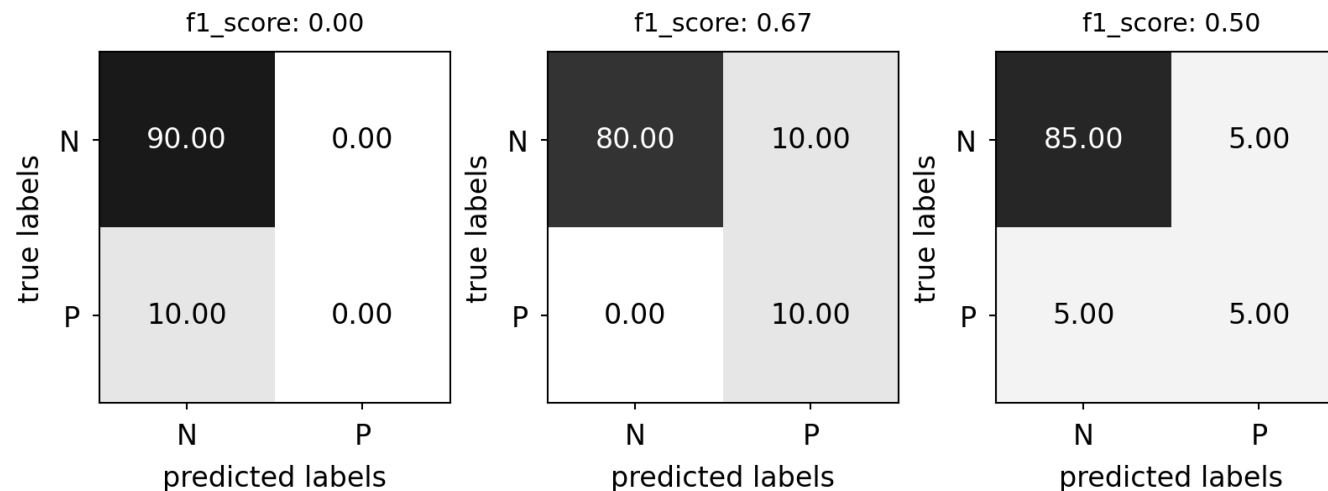


Binary classification

F1-score:

- Trades off precision and recall:

$$F1 = 2 * \frac{\textit{precision} * \textit{recall}}{\textit{precision} + \textit{recall}}$$



Lab1: Data manipulation demo