

# Lecture #4: ML Quality Assurance

Modern machine learning for data analysis

Common Pitfalls

Avoidance Strategies



Michael Fu

<https://michaelfu1998-create.github.io/>



Dr. Kla Tantithamthavorn

Senior Lecturer in Software Engineering

<http://chakkrit.com> @klainfo



# Schedule <http://chakkrit.com/teaching/quantitative-research-methods>

Date	Location	Topics
<del>11 November 2024</del>	<del>G.13 Woodside Building (20 Exhibition Walk)</del>	<del>Design Science Paradigm</del>
<del>15 November 2024</del>	<del>G.13 Woodside Building (20 Exhibition Walk)</del>	<del>Statistical Analysis</del>
<del>25 November 2024</del>	<del>G.13 Woodside Building (20 Exhibition Walk)</del>	<del>Modern Regression Analysis</del>
<b>29 November 2024</b>	G.13 Woodside Building (20 Exhibition Walk)	<b>ML Quality Assurance</b>



All models are wrong,  
but some are useful.

George E. P. Box

quote fancy

# CLASSICAL MACHINE LEARNING



MONASH  
University

MONASH  
INFORMATION  
TECHNOLOGY

Data is pre-categorized  
or numerical

## SUPERVISED

Predict  
a category

### CLASSIFICATION

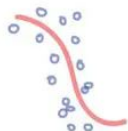
«Divide the socks by color»



Predict  
a number

### REGRESSION

«Divide the ties by length»



Data is not labeled  
in any way

## UNSUPERVISED

Divide  
by similarity

### CLUSTERING

«Split up similar clothing  
into stacks»



Identify sequences

Find hidden  
dependencies

### ASSOCIATION

«Find what clothes I often  
wear together»



### DIMENSION REDUCTION (generalization)

«Make the best outfits from the given clothes»



# Regression in Python | An Example

Preparing the dataset: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

```
# First, prepare the data for model building
import pandas
df = pandas.read_csv('diabetes.csv')
df.head()

#split dataset in features and target variable
feature_cols = df.columns.to_list()
feature_cols.remove('Outcome')

X = df[feature_cols] # Features
y = df.Outcome # Target variable

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=16)

# Check the distribution of Y-value (~35% positive)
y_train.value_counts()
0    375
1    201
Name: Outcome, dtype: int64
y_test.value_counts()
0    125
1     67
Name: Outcome, dtype: int64
```



# Classification in Python | An Example

# Building the model

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=16) # using the default parameters
logreg.fit(X_train, y_train) # fit the model with data
y_pred = logreg.predict(X_test)
```

# Build confusion matrix

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
```

```
array([[116,   9],
       [ 26,  41]])
```

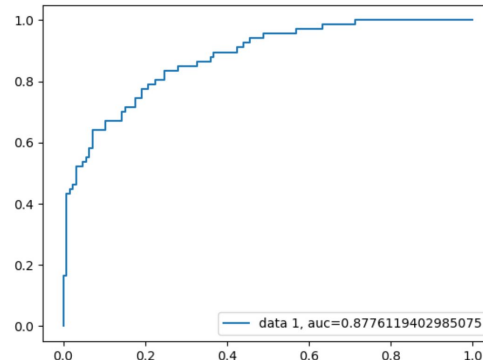
# calculate evaluation metrics

```
from sklearn.metrics import classification_report
target_names = ['without diabetes', 'with diabetes']
print(classification_report(y_test, y_pred, target_names=target_names))
```

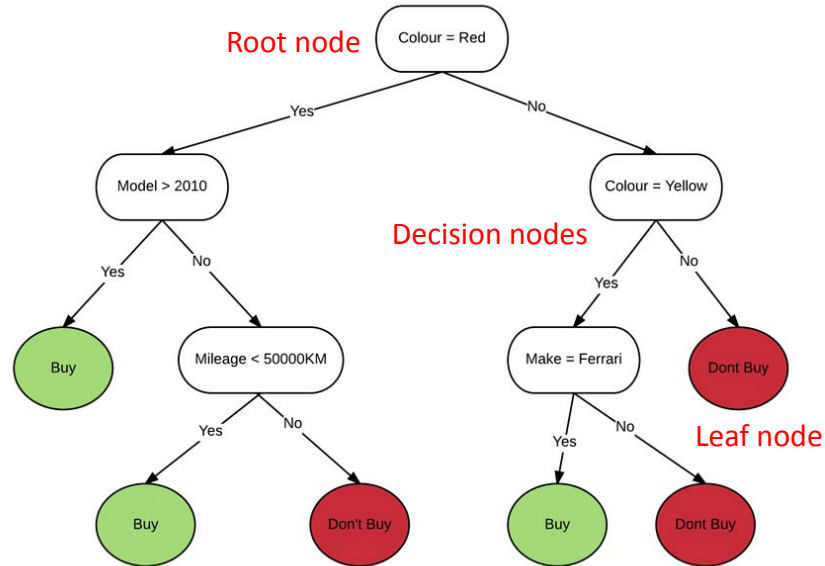
	precision	recall	f1-score
without diabetes	0.82	0.93	0.87
with diabetes	0.82	0.61	0.70
accuracy			0.82
macro avg	0.82	0.77	0.78
weighted avg	0.82	0.82	0.81

# calculate AUC

```
import matplotlib.pyplot as plt
plt.cla()
y_pred_proba = logreg.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr, tpr, label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



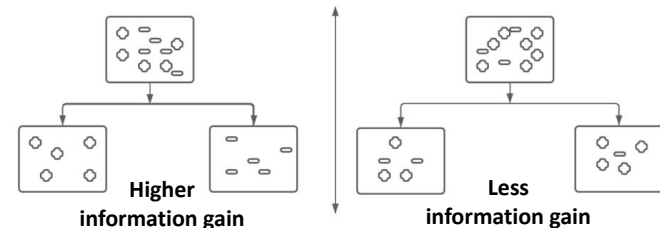
# Decision Trees (for regression or classification)



- **Root Node:** It represents the entire population or sample.
- **Decision Node:** A sub-node that split from other node
- **Leaf Node:** A Node that cannot be further split.
- **Pruning:** Decision trees can easily overfit. Pruning (i.e., removing sub-nodes) can be performed to avoid overfitting.

Predicting a response variable using a tree-like structure decisioning. To construct a tree, recursively splitting the dataset into leaf-nodes until every leaf-node cannot be split anymore.

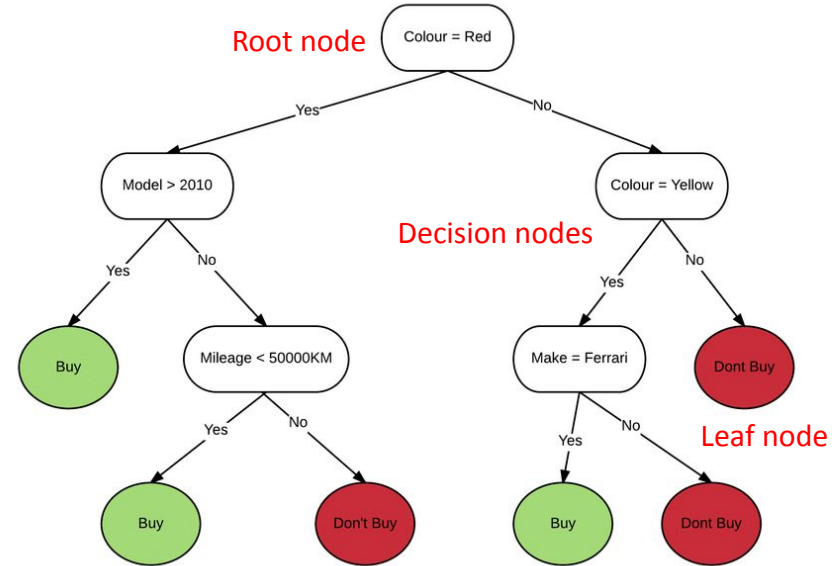
The features and rules used in splitting is determined by maximizing the information gain.



However, decision tree can be unstable because small variations in the data might result in a completely different tree being generated.

# Advantages of Decision Trees

1. **Simplicity and Interpretability:** Decision trees are easy to understand and interpret. The visual representation closely mirrors human decision-making processes.
2. **Versatility:** Can be used for both classification and regression tasks.
3. **No Need for Feature Scaling:** Decision trees do not require normalization or scaling of the data.
4. **Handles Non-linear Relationships:** Capable of capturing non-linear relationships between features and target variables.





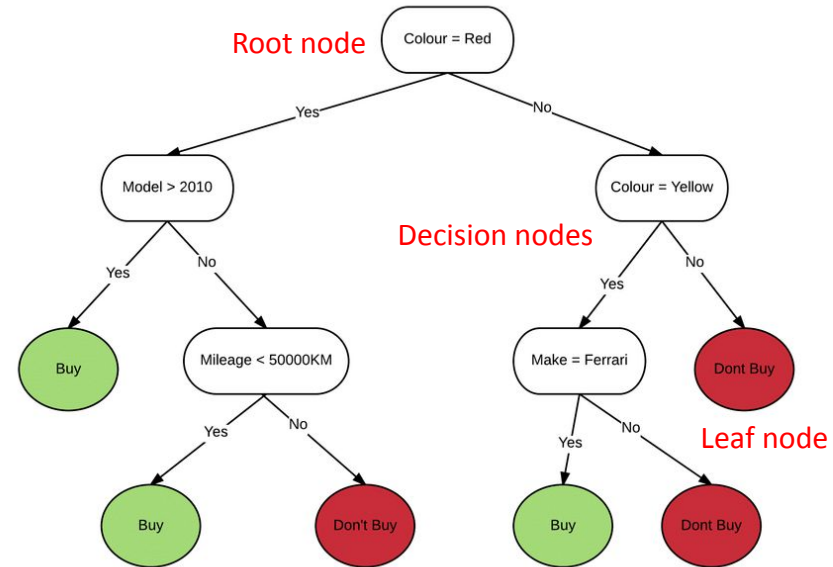
# Disadvantages of Decision Trees

1. **Overfitting:** Decision trees can easily overfit the training data, especially if they are deep with many nodes.
2. **Instability:** Small variations in the data can result in a completely different tree being generated.
3. **Bias towards Features with More Levels:** Features with more levels can dominate the tree structure.

## Pruning for overfitting

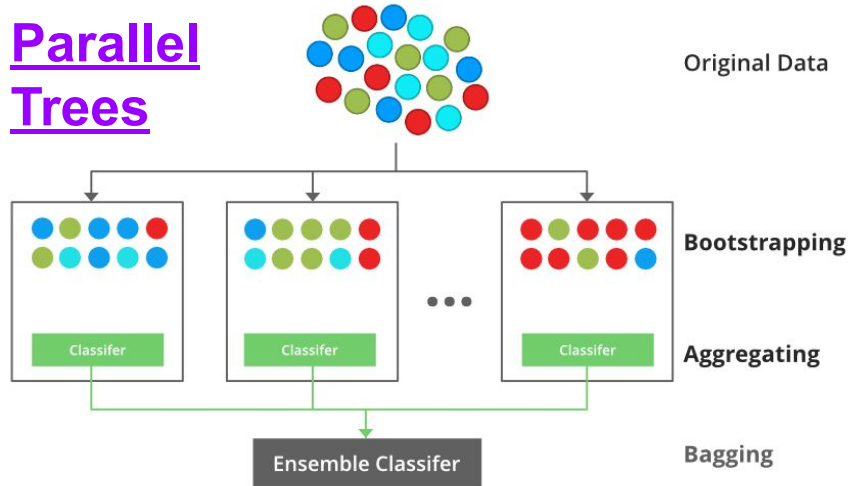
Pruning reduces the size of the tree by removing nodes that provide little power in classifying instances. There are two main types of pruning:

1. **Pre-pruning (Early Stopping):** Stops the tree from growing once it meets certain criteria (e.g., maximum depth, minimum number of samples per leaf).
2. **Post-pruning:** Removes branches from a fully grown tree that do not provide significant power.



# Random Forest

## Parallel Trees



**Random Forest** combines the outputs of multiple decision trees using majority votes (classification) or averaging (regression).

\* Robust to noises

## Advantages

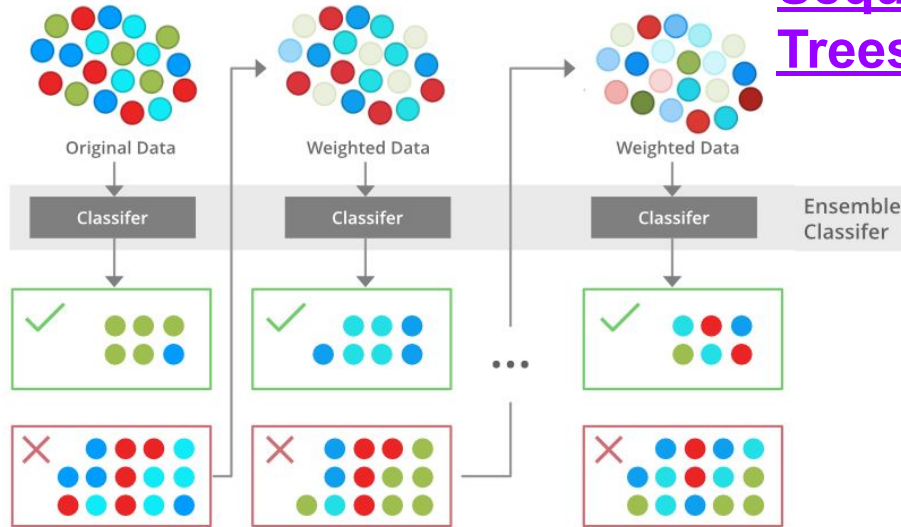
1. **Reduced Overfitting:** Averaging multiple uncorrelated trees reduces variance and prevents overfitting.
2. **High Flexibility:** Handles both regression and classification with high accuracy; robust against missing data.

## Disadvantages

1. **Time-Consuming:** Slower due to processing multiple trees for large datasets.
2. **Resource-Intensive:** Requires more memory and computational power.
3. **Complex Interpretation:** Harder to interpret compared to a single decision tree.

# XGBoost (Extreme Gradient Boosting)

## Sequential Trees



**XGBoost** build multiple trees where weights of the wrong predictions are passed to the next tree to focus on improving the wrong predictions.

\* Performs best

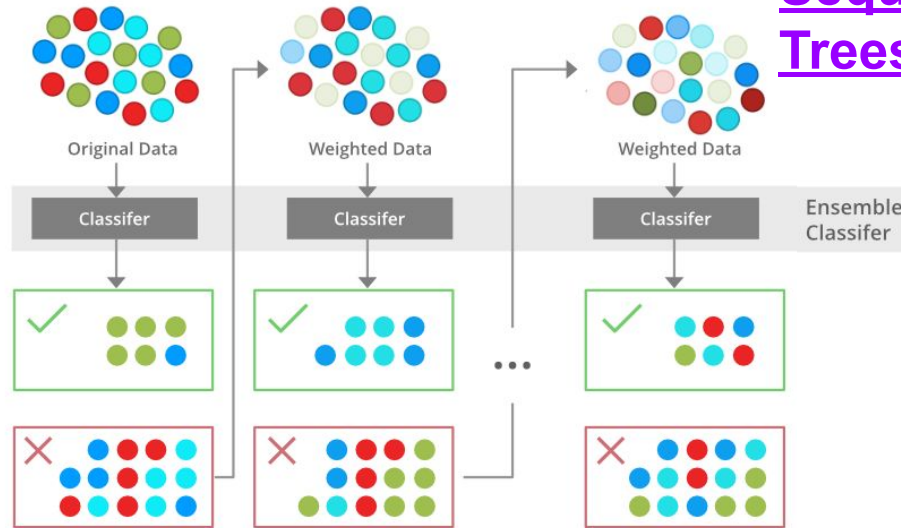
## Key Differences between XGB and RF

**Random Forest:** Trees are built **independently in parallel** (not sequentially), and their predictions are averaged for the final output.

**Boosting (XGBoost):** Trees are built **sequentially**, with each tree learning from the mistakes of the previous one, creating a more refined and accurate model over time.

# XGBoost (Extreme Gradient Boosting)

## Sequential Trees



**XGBoost** build multiple trees where weights of the wrong predictions are passed to the next tree to focus on improving the wrong predictions.

\* Performs best

## Advantages

1. Higher Accuracy: Consistently outperforms other algorithms on structured/tabular data.
2. Regularization for overfitting: Built-in L1 and L2 regularization helps prevent overfitting.

## Disadvantages

1. Complexity from Tree Dependency: Difficult to train and interpret compared to simpler models such as RF.
2. Resource-Intensive: Requires significant memory and computational power.
3. Hyperparameter Tuning: Needs extensive tuning for optimal performance, which can be time-consuming.

# Random Forest and XGBoost

# Random Forest

```
from sklearn.ensemble import RandomForestClassifier
randomForest = RandomForestClassifier(random_state=16)
randomForest.fit(X_train, y_train)
```

# calculate AUC

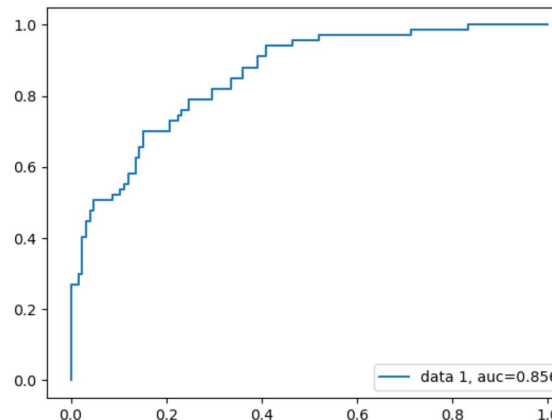
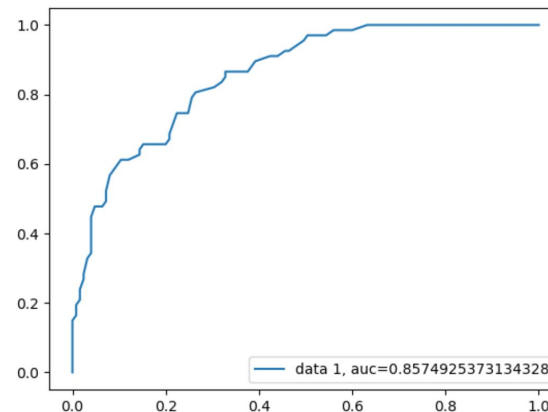
```
import matplotlib.pyplot as plt
plt.cla()
y_pred_proba = randomForest.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr, tpr, label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

# XG Boost

```
from xgboost import XGBClassifier
xgboost = XGBClassifier()
xgboost = xgboost.fit(X_train, y_train)
```

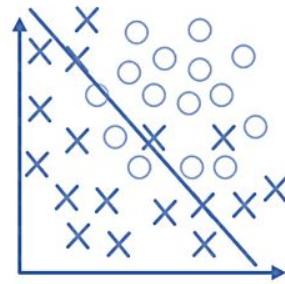
# calculate AUC

```
import matplotlib.pyplot as plt
plt.cla()
y_pred_proba = xgboost.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr, tpr, label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



# Overfitting and Underfitting

## Under-fitting



**Under-fitting**  
(too simple to  
explain the variance)

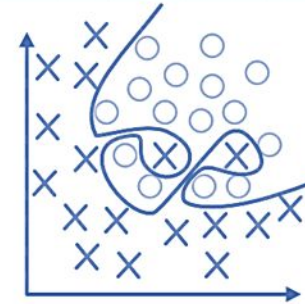
A model cannot capture the data trend. Underfitting would occur, for example, when fitting a linear model to non-linear data.

The model would have poor predictive performance as it cannot capture the data trend.

### How to:

- Estimate the performance using cross validation technique
- Choose a more advanced algorithm

## Over-fitting



**Over-fitting**  
(forcefitting--too  
good to be true)

A model describes noises instead of the underlying relationship (overly complex).

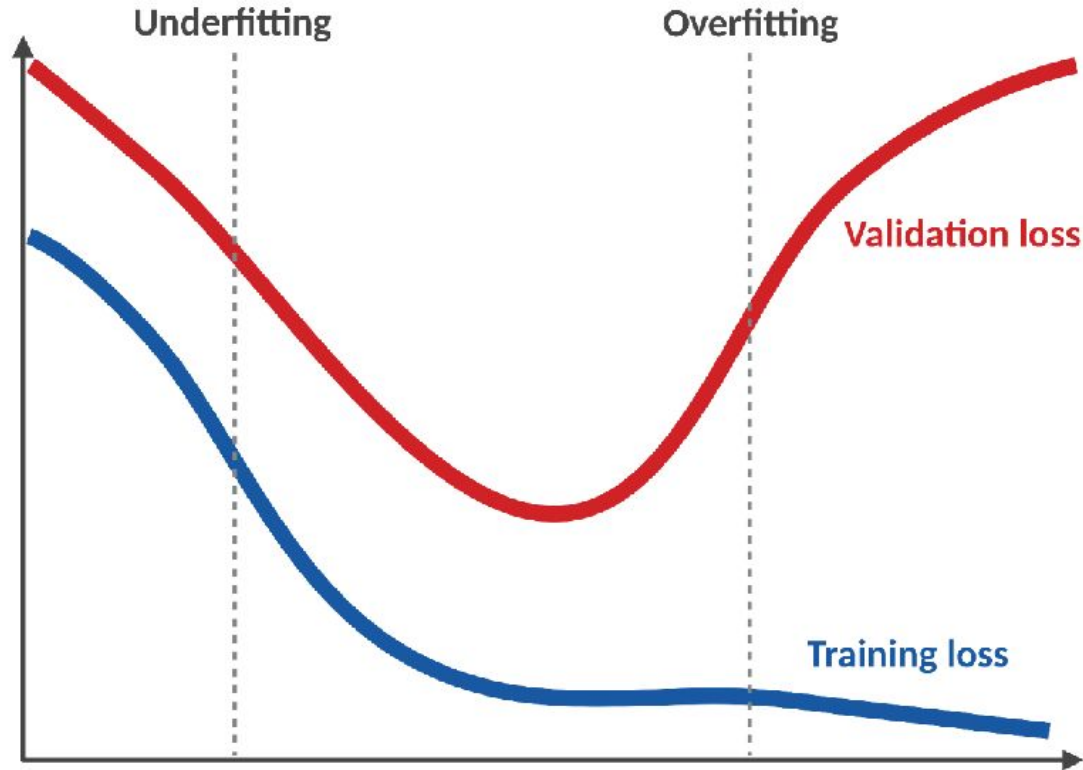
The model have poor performance, as it overreacts to minor fluctuations in the training data

### How to:

- Estimate the performance using cross validation technique
- Reduce the number of features (Dimensional reduction)



# Overfitting and Underfitting (How to spot them?)



# Machine Learning Quality Assurance | 12 Checklists

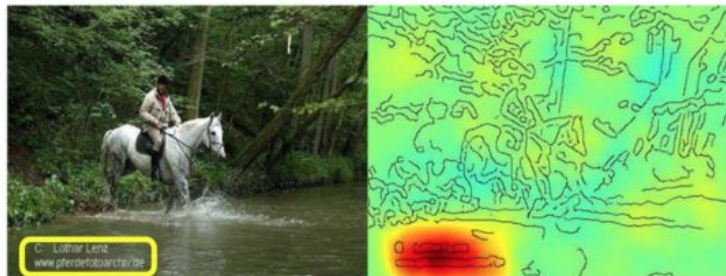
1. Assure no spurious correlations
2. Assure no data mismatch between testing and use case
3. Assure no testing data leakage
4. Assure no testing data leakage in time-series problems
5. Using proper performance metrics for an imbalanced dataset
6. Assure no duplicate data rows between training and testing
7. Avoid training a model using too many features
8. Avoid training a model using highly-correlated features
9. Avoid training a model using severely imbalanced columns
10. Avoid re-balancing the dataset while including the testing dataset.
11. Avoid hyperparameter tuning on the testing dataset.
12. Avoid using a naïve cross-validation method

*An experience report on defect modelling in practice: pitfalls and challenges Chakkrit Tantithamthavorn, and Ahmed E. Hassan In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2018, Gothenburg, Sweden, May 27 - June 03, 2018, 2018*

# #1: Assure no spurious correlations

If your dataset contains anything other than the signal (i.e. noise) that is correlated with your labels, your model will learn to exploit it for prediction.

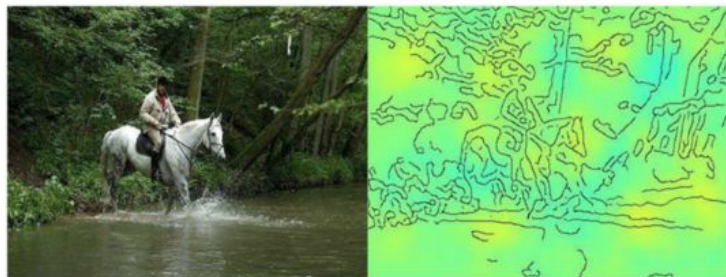
Horse-picture from Pascal VOC data set



Source tag  
present



Classified  
as horse



No source  
tag present



Not classified  
as horse

## How to avoid:

Check the correlation between each features and target prediction

For features with high correlation, check

- Do they make sense as causal relationships?
- Do they fit established theory?
- Can you find a mechanism for causation?
- Is there a direct link, or are mediator variables involved?

If not, we may exclude the feature out.

We may also consider mixed effects models that can consider the features as random effects, e.g., mixed effects logistic regression

```
m <- glmer(remission ~ featureA + featureB + featureC + featureD + (1 | confoundingFeature), data = df, ...)
```

## #2: Assure no data mismatch between testing and use case

Mismatch between testing dataset and the real use case may lead to a model with poor performance.

### How to avoid:

Check if the testing data has a similar characteristics as the real usage scenario.

For example, a model trained using **ImageNet** dataset may not performed well with **ObjectNet** dataset

Chairs



**ImageNet**

(simple and straight images of objects)

Chairs by rotation



Chairs by background



Chairs by viewpoint



**ObjectNet**

(Objects with different rotations, backgrounds, and viewpoints)

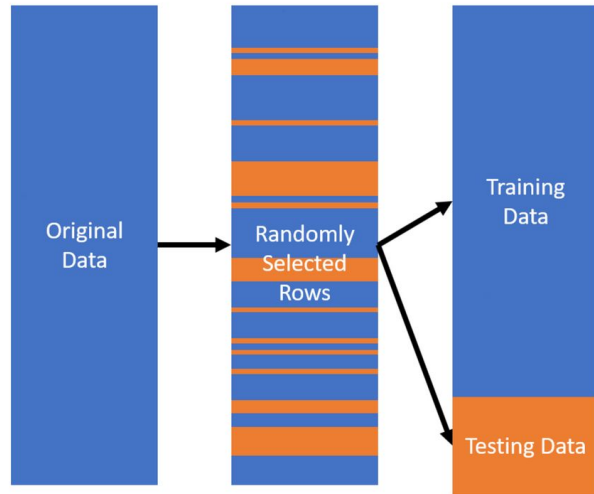
# #3: Assure no testing data leakage

Using testing data to train the model will mislead the model performance evaluation.

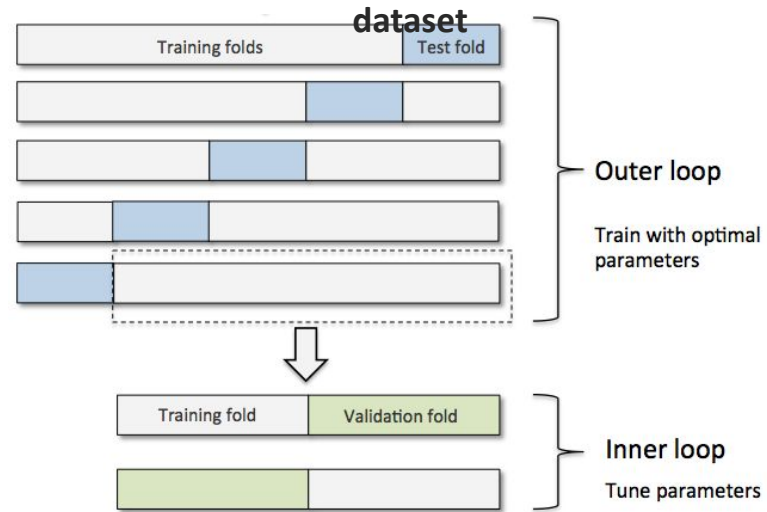
## How to avoid:

Apply cross-validation technique properly.

### Clearly splitting training and testing dataset



### If required, split the validation data from the training

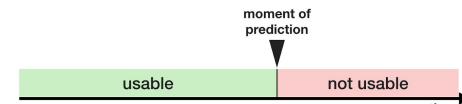


# #4: Assure no testing data leakage in time-series problems

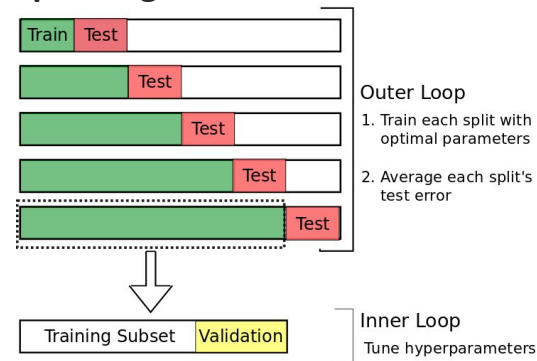
In time-series prediction problem (e.g., predicting stock price), training the model using future data and testing using the past data is strictly prohibited.

## How to avoid:

- Apply a time-wise cross validation
  - Expanding window (mimic scenario where more data is collected over time)
  - Sliding window (all splits have the same training size)
- Avoid extracting features based on the future data



## Expanding window cross-validation



## Information leakage when perform random



## Sliding window cross-validation



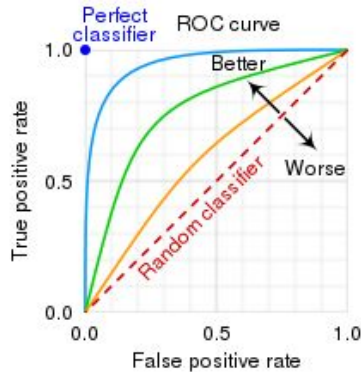


# #5: Using proper performance metrics for an imbalanced dataset

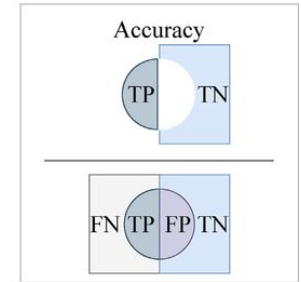
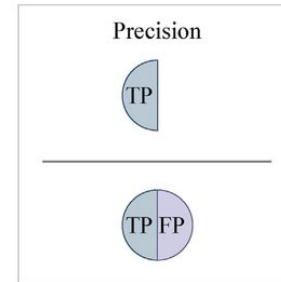
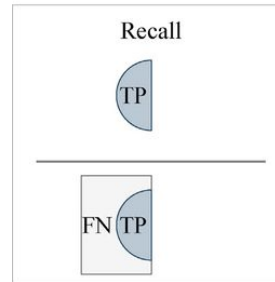
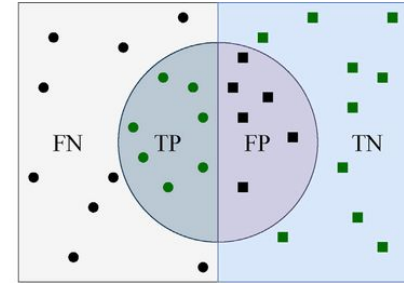
In an imbalanced prediction problem (e.g., identifying cancer patient), using metrics like accuracy may mislead the benefit and usefulness of the model in real usage scenario.

**How to avoid:** By using

- AUC
- MCC



AUC = 1 indicate that the model can perfectly predict the outcome



## #6: Assure no duplicate data rows between training and testing

Having duplicate data rows in both training and testing dataset can be considered as data leakage.

### How to avoid:

- Include other dimensions (features) to provide more information for the model training
- Check and remove the duplicated data rows to avoid leakage.

**Training Dataset**

ID	Size	Suburb
1	200	Carlton
2	150	Melbourne
3	170	Melbourne
4	340	Fitzroy

**Testing Dataset**


ID	Size	Suburb
5	200	Carlton
6	180	Melbourne
7	280	Fitzroy
8	320	Fitzroy

# #7: Avoid training a model using too many features

The model that trained with too many features (no. features > no. rows) may predict a single data points using single feature, or making special case just for a single data point.

## How to avoid:

- Find a larger datasets
- Perform dimensional reduction to reduce the number of features (e.g., Principal component analysis, correlation analysis)

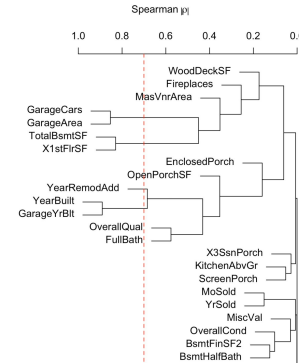
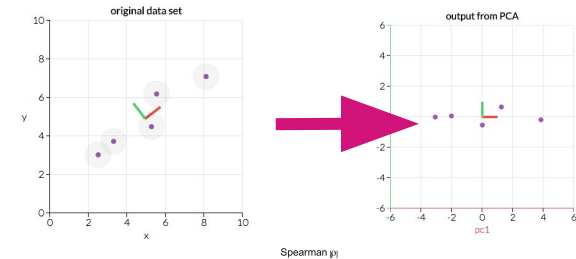
 **Do not** use one-hot encoding

Index	Animal	One-Hot code	Index	Dog	Cat	Sheep	Lion	Horse
0	Dog		0	1	0	0	0	0
1	Cat		1	0	1	0	0	0
2	Sheep		2	0	0	1	0	0
3	Horse		3	0	0	0	0	1
4	Lion		4	0	0	0	1	0

 **Use label encoding**

State (Nominal Scale)	State (Label Encoding)
Maharashtra	3
Tamil Nadu	4
Delhi	0
Karnataka	2
Gujarat	1
Uttar Pradesh	5

\*also make sure you treat this as an ordinal variables



Apply Dimensional reduction

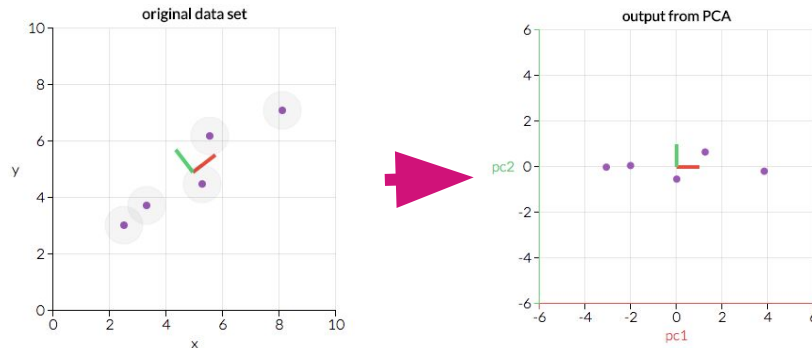
- Principal Component Analysis (PCA) or
- Correlation analysis (Spearman's Rho)

# #8: Avoid training a model using highly-correlated features

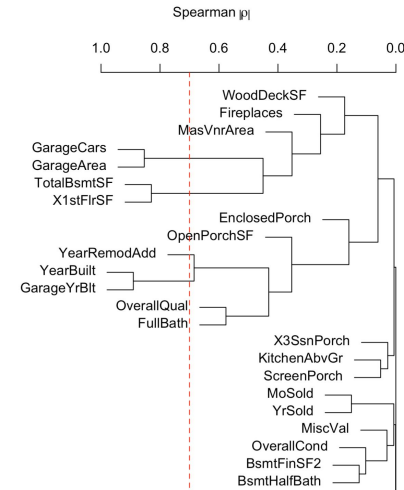
Having too many features or highly correlated features may interfere the model's prediction.

## How to avoid:

- Perform dimensional reduction to reduce the number of features (e.g., Principal component analysis, correlation analysis)



Principal Component Analysis (PCA)



Correlation analysis based on Spearman Rho

# #9: Avoid training a model using severely imbalanced columns (non-explanatory features)

Training a model using imbalanced columns may lead to overfitting (i.e., having too many features that does not provide predictive performance).

## How to avoid:

- Exclude imbalanced columns

ID	Size	Suburb	Type (Y)
1	400	Collingwood	Large
2	150	Melbourne	Small
3	170	Melbourne	Small
4	240	Melbourne	Small
5	220	Melbourne	Small
6	400	Melbourne	Large
7	450	Melbourne	Large
...	...	...	...



```

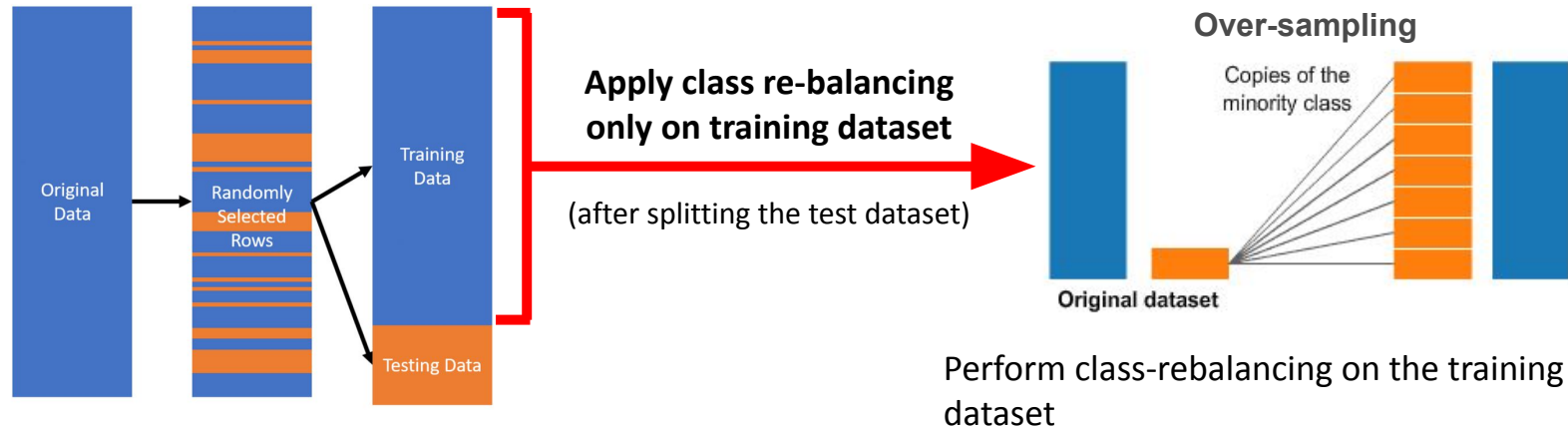
if suburb =
  Collingwood:
    Type is Large
else:
  if size >= 400:
    Type is large
  else:
    Type is small
  
```

# #10: Avoid re-balancing on the testing dataset.

Testing dataset should be treated as unseen data. We should preserve the characteristics of the dataset.

## How to avoid:

- Split the dataset into training and testing first.  
Then, only rebalance the training dataset.



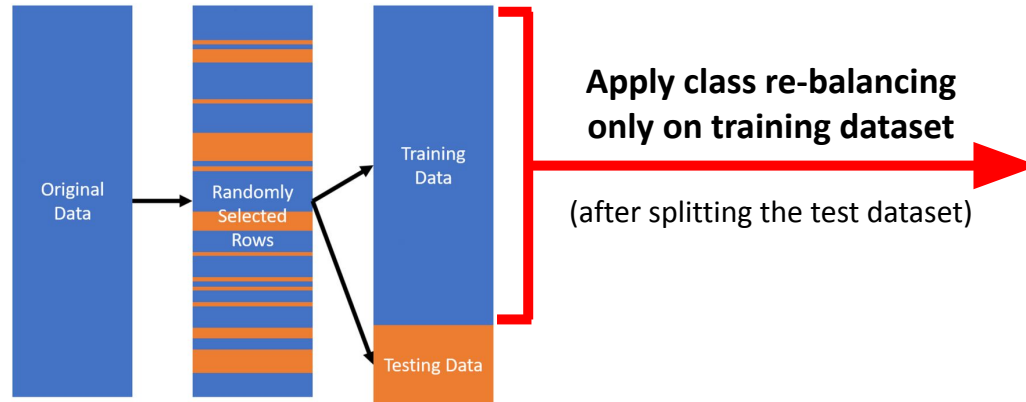


# #11: Avoid hyperparameter tuning on the testing dataset.

Testing dataset should be treated as unseen data. We should preserve the characteristics of the dataset.

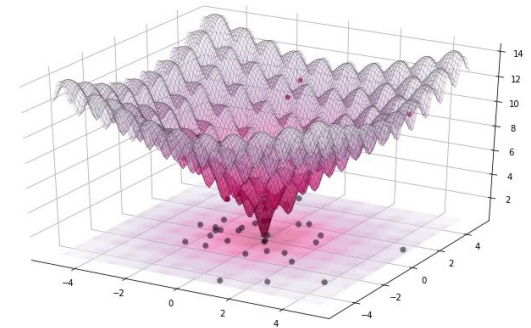
## How to avoid:

- Split the dataset into training and testing first.
- Then, only tuning the hyperparameter based on the training dataset.



## Hyper parameter tuning

- Grid search
- Random search
- Differential Evolution

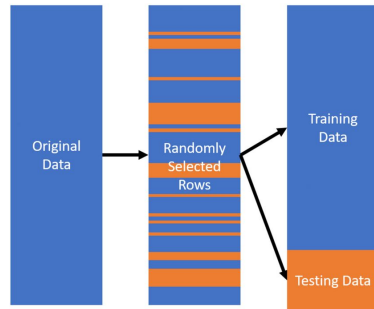



# #12: Avoid using a naïve cross-validation method

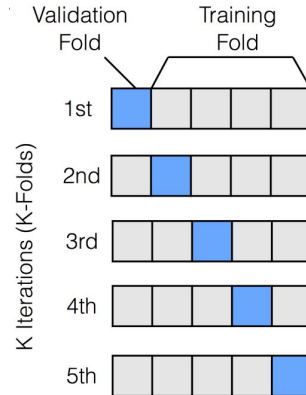
Measuring naïve cross-validation split methods may not accurately reflect the model prediction performance.


## How to avoid:

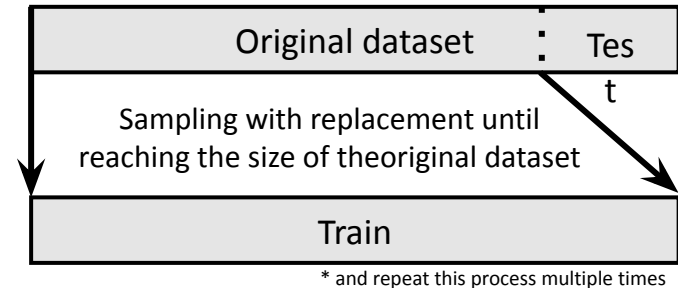
- Split the dataset into training and testing first.  
Then, only tuning the hyperparameter based on the training dataset.




 Single random split



 K-fold cross validation



 Bootstrap sampling validation