Advanced Topics for Supervised Learning

Advanced Topics for Supervised Learning

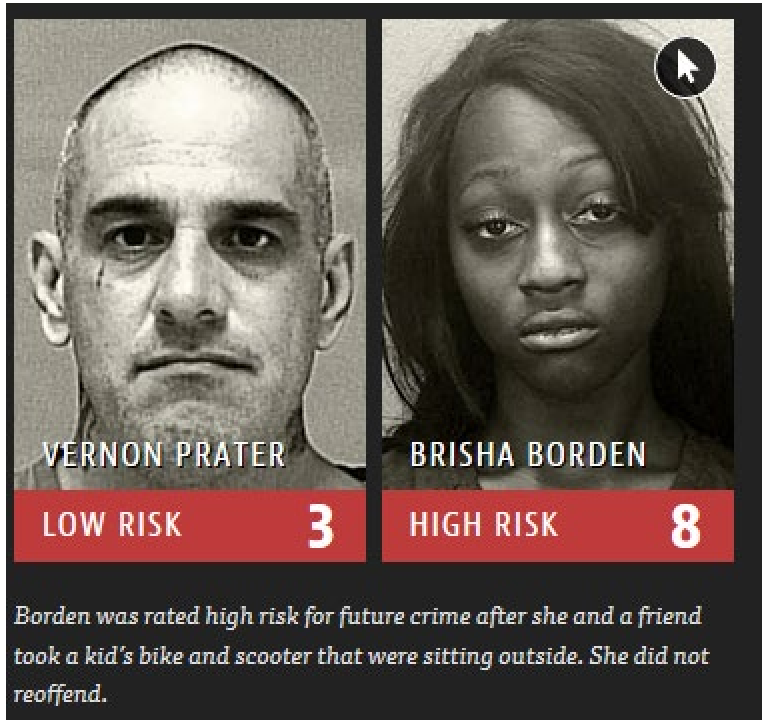
- Briscoe, E., & Feldman, J. (2011). Conceptual complexity and the bias/variance tradeoff. *Cognition*, *118*(1), 2-16.
- Guyon, I., Elisseeff, A., 2006. An Introduction to Feature Extraction, in: Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L. (Eds.), Feature Extraction, *Studies in Fuzziness and Soft Computing*. Springer Berlin Heidelberg, pp. 1–25
- Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16:321–357, 2002.
- Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: adaptive synthetic sampling approach for imbalanced learning. In 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 1322–1328. IEEE, 2008.
- Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.".

Advanced Topics for Supervised Learning

Algorithmic bias

- There are many examples where (supervised) machine learning has gone wrong:
  - A hiring algorithm that disproportionately favored men.
  - Online advertising that seemed to reproduce the phenomenon of redlining on a digital map.
  - A widely-used healthcare algorithm that exhibited significant racial bias.

- Machine learning algorithms are not perfect but suffer shortcomings
  - Restriction bias is the representational power of a machine learning algorithm.
  - Restriction bias tells us what our model can represent.
    - Example: the ID3 decision tree algorithm can only be applied to categorical data and not to numerical data
    - Some problems maybe not be representable using a programming language and thus maybe cannot be learned

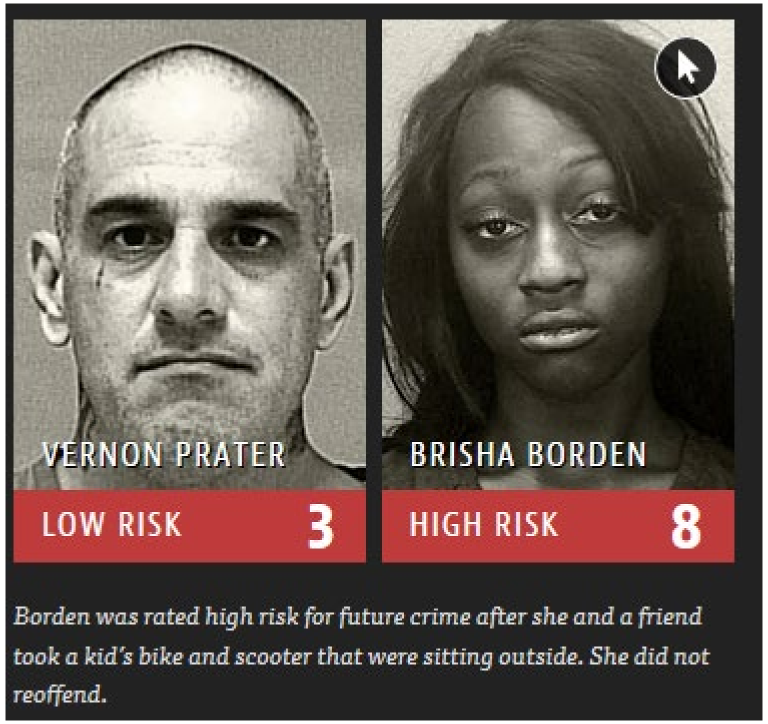


Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

Prior offenses:	Prior offenses:
• 2 armed robberies, • 1 attempted armed robbery	• 4 juvenile misdemeanours
Subsequent offenses:	Subsequent offenses:
• 1 grand theft	• None
Classified as low risk	Classified as high risk

Algorithmic bias

- Example where (supervised) machine learning has gone wrong and showed systematic bias:
  - [Amazon scraps secret AI recruiting tool that showed bias against women]
  - [Facebook algorithms based on ‘race, color, national origin, religion’]
  - [Racial bias in many facial-recognition systems]

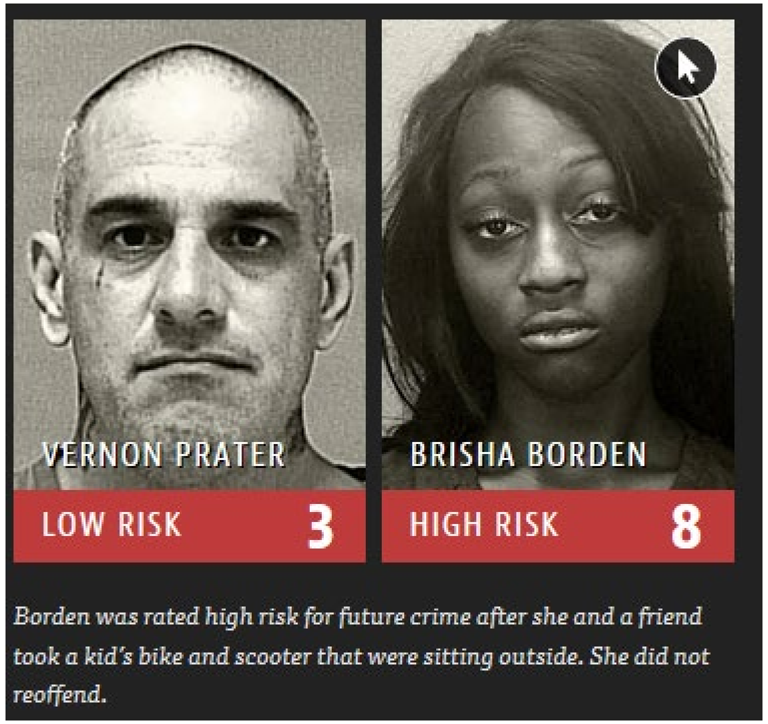


Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

Prior offenses: • 2 armed robberies, • 1 attempted armed robbery	Prior offenses: • 4 juvenile misdemeanours
Subsequent offenses: • 1 grand theft	Subsequent offenses: • None
Classified as low risk	Classified as high risk

Algorithmic bias

- The results of an algorithm may further be biased depending on the data used
  - Skewed dataset:
    - Lack of representation in the data can affect (supervised) machine learning's ability to learn from diverse sets of examples, which can result in biased model performance.
  - Tainted examples:
    - Unreliable labels or historical bias in the data have a direct impact on (supervised) machine learning's discriminatory behavior
  - Limited features:
    - Feature collection for certain groups may not be informative or reliable, which can occur under bad data collection practices.
    - Similar to a skewed dataset, this will impair (supervised) machine learning's ability to predict accurately for those groups.
  - Sample size:
    - Small datasets limit the ability of (supervised) machine learning's learning process and can result in bias.
  - Proxy features:
    - Features can indirectly leak information about the protected attributes, even in cases when that protected feature has been removed.
    - Zipcodes, sports activities, and university attended can be used by the model to indirectly infer race or gender; if the examples are then tainted by historical bias, even without direct access to that protected feature, the model will learn the pattern of discrimination from proxies

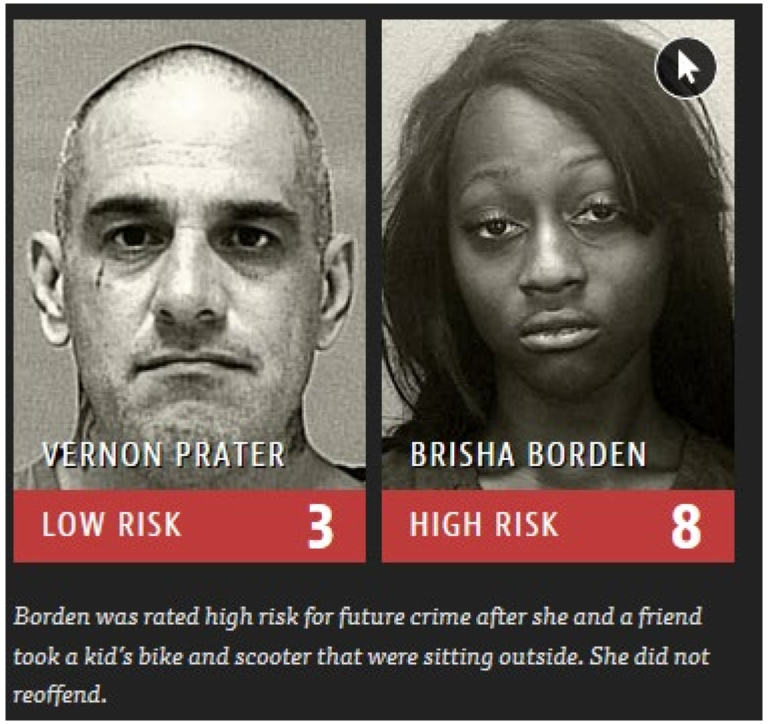


Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

Prior offenses: • 2 armed robberies, • 1 attempted armed robbery	Prior offenses: • 4 juvenile misdemeanours
Subsequent offenses: • 1 grand theft	Subsequent offenses: • None
Classified as low risk	Classified as high risk

Algorithmic bias

- Mitigation strategies for biases can occur at different points of the machine learning pipeline: pre-processing, in-processing, and post-processing.
  - Pre-processing strategies aim at reducing bias in the original data before you train your model.
    - Sampling the data to have more balanced and curated data or
    - Giving weights to rows representing unprivileged
    - Reduce the correlation between features, target, and protected attributes, and predictability of protected or sensitive attributes
    - Example literature for further, detailed, information:
      - Kamiran, Faisal and Calder s, Toon. Data preprocessing techniques for classification without discrimination. Knowledge and Information Systems, 33(1):1–33, 2012
      - Calmon, F., Wei, D., Vinzamuri, B., Ramamurthy, K. N., and Var shney, K. R. Optimized pre-processing for discrimination prevention. In Advances in Neural Information Processing Systems 30, 2017.
      - M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian. Certifying and removing disparate impact. In KDD, 2015.

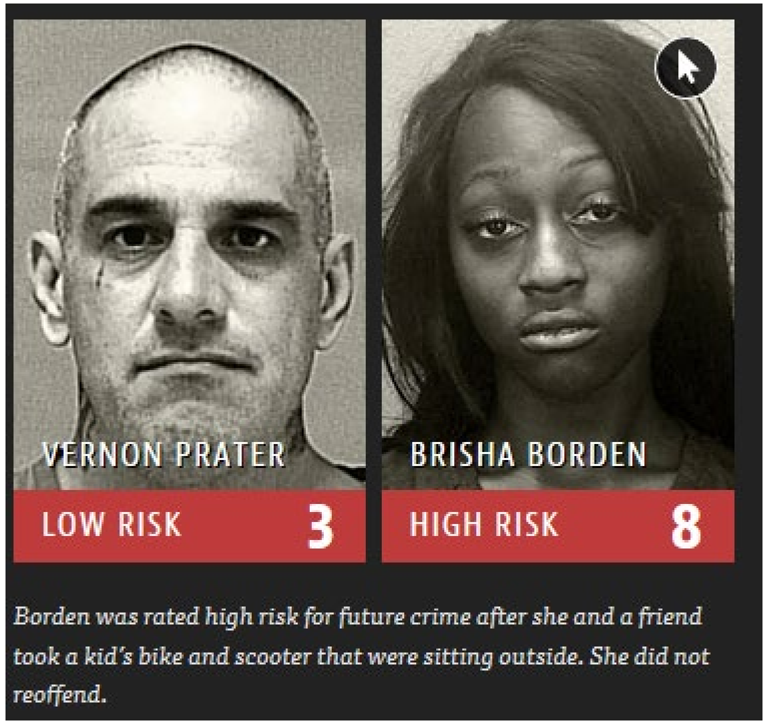


Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

Prior offenses: • 2 armed robberies, • 1 attempted armed robbery	Prior offenses: • 4 juvenile misdemeanours
Subsequent offenses: • 1 grand theft	Subsequent offenses: • None
Classified as low risk	Classified as high risk

Algorithmic bias

- Mitigation strategies for biases can occur at different points of the machine learning pipeline: pre-processing, in-processing, and post-processing.
  - In-processing strategies tackle bias during model training
    - Examples are classification with fairness constraints, which enforces fairness constraints in the optimization process, and fairness aware classifier
    - Example literature for further, detailed, information :
      - L. Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K. Vishnoi. Classification with fairness constraints: A meta-algorithm with provable guarantees. In Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* '19, 2019.
      - T. Kamishima, S. Akaho, H. Asoh, and J. Sakuma. Fairness-aware classifier with prejudice remover regularizer. Machine Learning and Knowledge Discovery in Databases, pages 35—50, 2012.

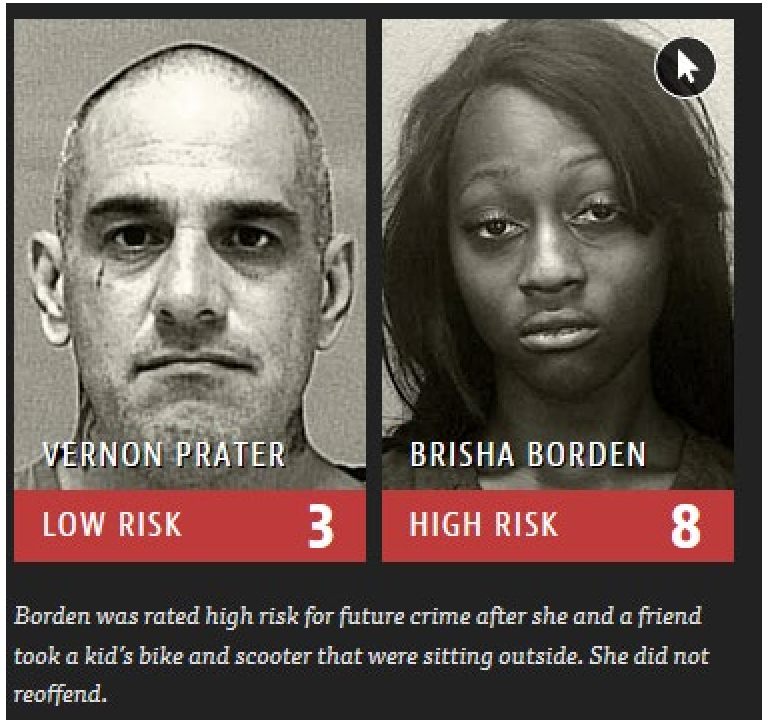


Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

Prior offenses: • 2 armed robberies, • 1 attempted armed robbery	Prior offenses: • 4 juvenile misdemeanours
Subsequent offenses: • 1 grand theft	Subsequent offenses: • None
Classified as low risk	Classified as high risk

Algorithmic bias

- Mitigation strategies for biases can occur at different points of the machine learning pipeline: pre-processing, in-processing, and post-processing.
  - Post-processing strategies reduce bias by modifying model predictions
    - Change model predictions for privileged and unprivileged groups to reduce bias in model predictions based on specific fairness metrics
    - Example literature for further, detailed, information :
      - Pleiss, G., Raghavan, M., Wu, F., Kleinberg, J., and Weinberger, K. Q. (2017). On fairness and calibration. In Advances in Neural Information Processing Systems, pages 5680–5689.
      - Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In Advances in Neural Information Processing Systems, 2016.



Borden was rated high risk for future crime after she and a friend took a kid's bike and scooter that were sitting outside. She did not reoffend.

Prior offenses: • 2 armed robberies, • 1 attempted armed robbery	Prior offenses: • 4 juvenile misdemeanours
Subsequent offenses: • 1 grand theft	Subsequent offenses: • None
Classified as low risk	Classified as high risk

Main Challenges of Machine Learning

- The main task is to select a machine learning algorithm (classifier) and train it on some data

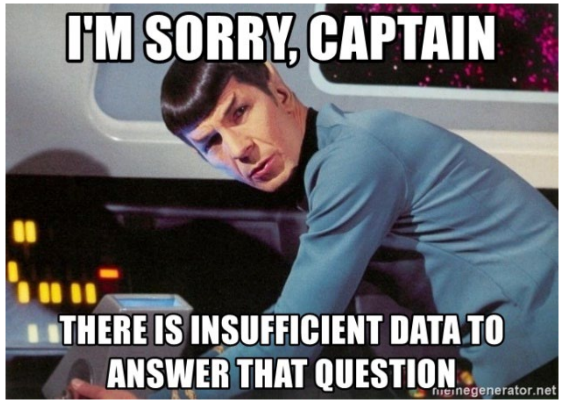
- The two things that can go wrong are
  - “bad data” and
  - “bad algorithm”

- We discuss those two aspects in the next slides

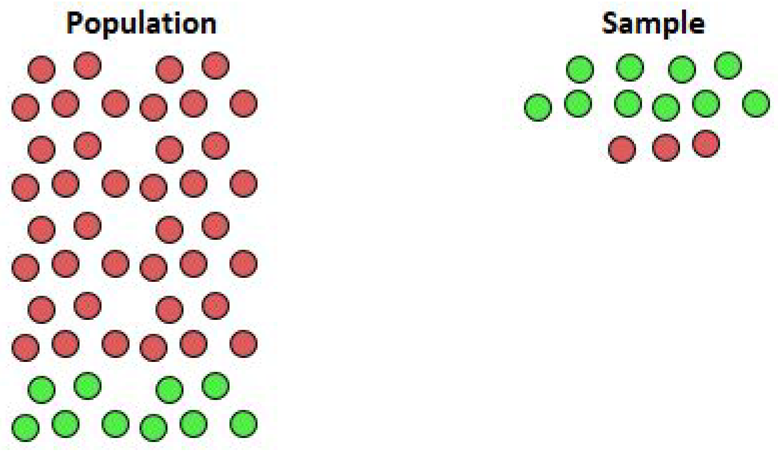
Advanced Topics for Supervised Learning

Main Challenges of Machine Learning

- "Bad data"
  - Insufficient quantity of training data
    - It takes a lot of data for most machine learning algorithms to work properly
    - Even for very simple problems one typically needs thousands of examples
    - For complex problems, such as, image or speech recognition one may need millions of examples (unless you can reuse parts of an existing model)

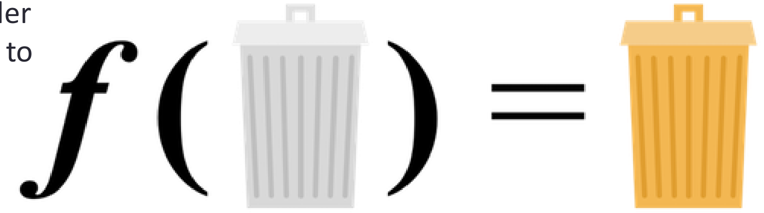


  - Non-representative training data
    - It is crucial to use a training set that is representative of the cases we want to generalize to
    - If the sample is too small, we will have sampling noise (i.e., nonrepresentative data as a result of chance)
    - Even very large samples can be nonrepresentative of the sampling method is flawed
      → this is called "sampling bias"



Main Challenges of Machine Learning

- “Bad data”
  - Poor data quality
    - If the training data is full of errors, outliers, and noise (e.g., due to poor-quality measurements), it will make it harder for the machine learning algorithm to detect the underlying patterns, and the machine learning model is less likely to perform well (“garbage in, garbage out”)
    - It is often well worth the effort to spend time cleaning up your training data
    - Data scientists spend a significant part of their time doing just preparing and cleaning data



  - Irrelevant Features
    - A critical part of the success of a machine learning project is producing a good set of features to train on.
    - This process, called feature engineering and includes
      - feature selection (selecting the most useful features to train on among existing features)
      - feature extraction (combining existing features to produce a more useful one)
      - feature engineering (creating new features by gathering new data)

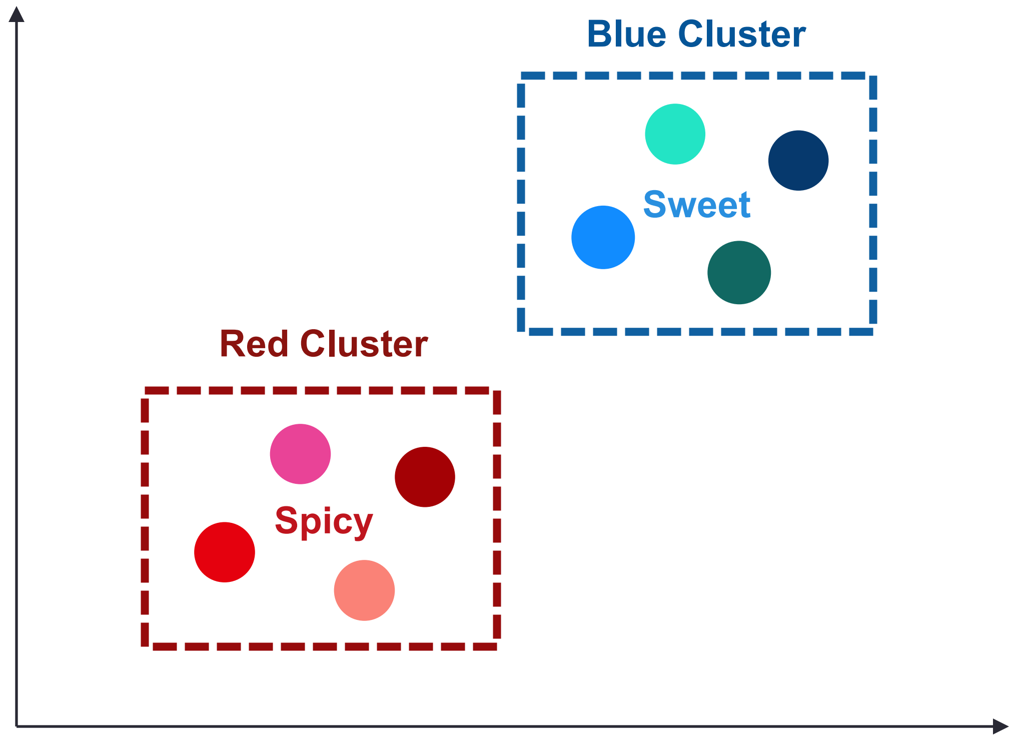
Advanced Topics for Supervised Learning

Curse of Dimensionality

- The “curse of dimensionality”
  - Refers to problems that occur when a data set has too many features, also called “high-dimensional data”
  - Examples:
    - If we’re recording 60 features for each of our observations, we’re working in a space with 60 dimensions.
    - If we’re analyzing grayscale images sized 50x50, we’re working in a space with 2,500 dimensions.
    - If the images are RGB-colored, the dimensionality increases to 7,500 dimensions (one dimension for each color channel in each pixel in the image)

Curse of Dimensionality

- The "curse of dimensionality" is a common problem when our data has too many features
  - Classifiers often rely on distance measures (e.g. k-nearest neighbors)
  - As long as we only have only a few dimensions, classification and clustering usually work well



Our 2 Color Based Clusters of Candy Flavor

Curse of Dimensionality

- Unless we somehow label the data, the computer does *not* know that there is a reddish ("spicy candy") and a bluish ("sweet candy") cluster in the data:

	Reddish	Bluish
🔴	1	0
🔴	1	0
🔴	1	0
🔴	1	0
🔵	0	1
🟢	0	1
🟢	0	1
🔵	0	1

Perfect clusters

Curse of Dimensionality

- If we do not label the observations, they become harder to cluster
  - Every candy is its color
  - The algorithm does not know the relationship between the colors
  - For example, unlike humans, it does not know that pink is closer to red than to turquoise
  - Given this set of features, the algorithm concludes that there are eight clusters and they are all equally similar to each other

	Red	Maroon	Pink	Flamingo	Blue	Turquosie	Seaweed	Ocean
●	**1**	0	0	0	0	0	0	0
●	0	**1**	0	0	0	0	0	0
●	0	0	**1**	0	0	0	0	0
●	0	0	0	**1**	0	0	0	0
●	0	0	0	0	**1**	0	0	0
●	0	0	0	0	0	**1**	0	0
●	0	0	0	0	0	0	**1**	0
●	0	0	0	0	0	0	0	**1**

High Dimensional Data Makes Trouble for Clustering

Curse of Dimensionality

- Classification problems are often within data sets which have > 50 variables ("features")
  - Most of this data is usually in some way relevant
  - Nevertheless, much data is unimportant, redundant, or noisy

- Solution: Dimensionality reduction (next slides)
  - Feature extraction "compresses" the available information in a smaller number of features
    - Here some features are transformed so that new features are created out of the existing features
  - Feature selection chooses features that are most suitable for classification and sort out less important ones
    - Here the features are maintained but one only selects a subset of those features for the analysis

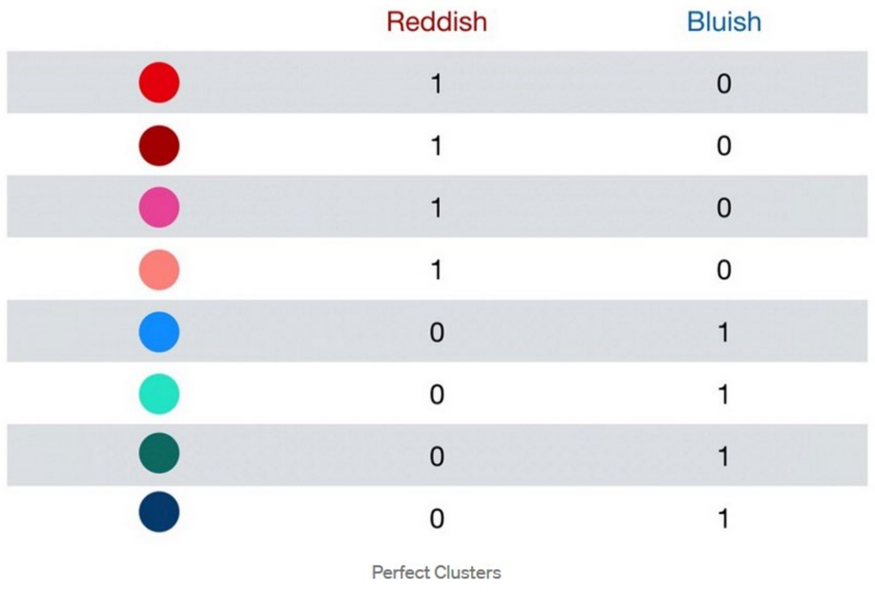
Feature Extraction

- Feature extraction is often theory-driven
  - Decide based on background and domain knowledge and depending on the goal of the analysis which features should be extracted
  - Theory and domain knowledge thereby helps to find out what to look out for

- Automatic feature extraction rarely works, since most of the data is too domain-specific

- Often considered to be more art than science – experience and theory help to focus the learner on the relevant information

- Example: Reducing to “reddish” and “bluish”



	Reddish	Bluish
●	1	0
●	1	0
●	1	0
●	1	0
●	0	1
●	0	1
●	0	1
●	0	1

Perfect Clusters

Feature Selection

- Even a well-defined data set may contain irrelevant, redundant, or noisy information
- Thus, it is important to <u>select</u> the most important features

- Why should we do this?
  - Increase classification speed
  - Increase classifier accuracy
  - Reduce storage requirements
  - Gain insights into the influence of each feature

- There are two types of feature selection methods
  - Filter methods
  - Wrapper methods

Feature Selection: Filter Methods

- Use statistical measures to quantify the expressiveness of features
  - For example, correlation coefficients, statistical tests, entropy measures
  - These measures can then be used to rank features according to their importance and choose the most appropriate features based on the rank
- Advantages of filter methods
  - Low computational complexity
  - Reasons for selection of a single feature are obvious
- Disadvantages of filter methods
  - No feature interaction considered
  - Not all measures are appropriate for all classification problems and feature types

Feature Selection: Wrapper Methods

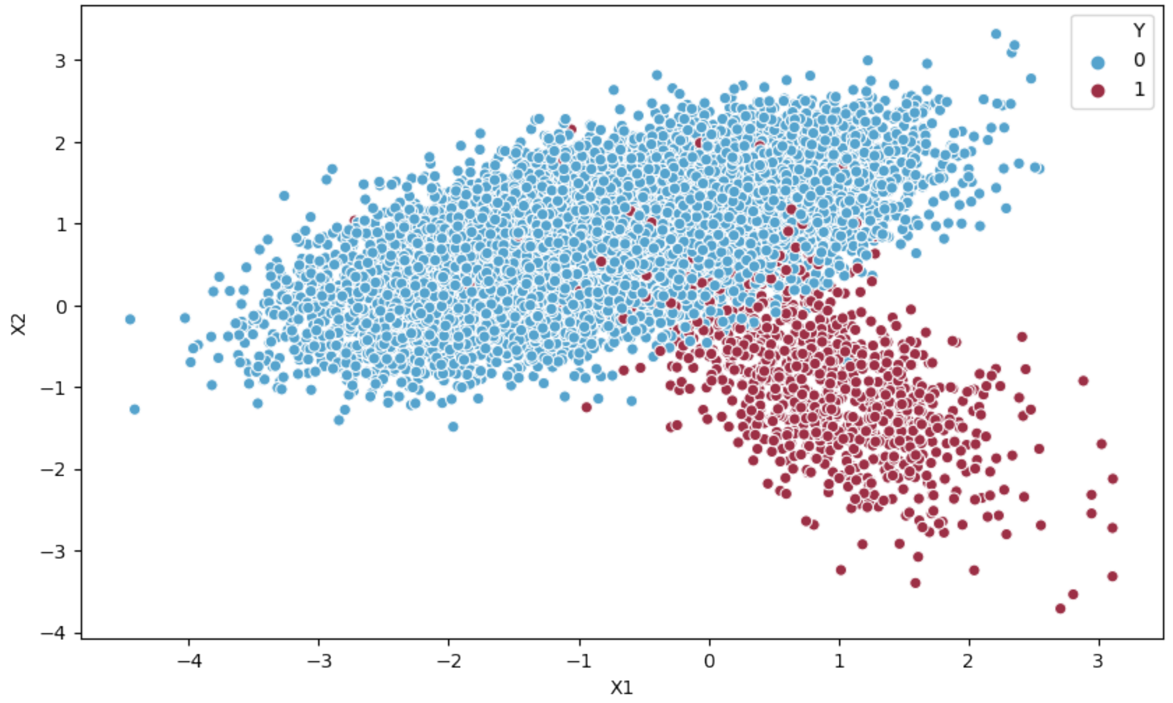
- Wrapper methods use the actual classification performance (e.g. accuracy, precision, recall, F1, AUC) to evaluate the quality of a feature set
- Most wrapper selectors use greedy algorithms to expand or reduce the feature set iteratively
- Examples:
  - Forward selection: Start with an empty set and add features that improve the chosen performance index (e.g. AUC)
  - Backward elimination: Start with all the features and eliminate the least useful ones
- Advantages of wrapper methods
  - Feature combinations are evaluated
  - Suitable for any classification problem
  - Robust against overfitting
- Disadvantages wrapper methods
  - High computational complexity
  - Not all feature combinations might be tested

Advanced Topics for Supervised Learning

Imbalanced Classes

- In realistic settings, the occurrence of different classes within the target variable is often imbalanced
  - For example, in a cancer screening scenario 99% of the sample data have no cancer and only 1% have cancer

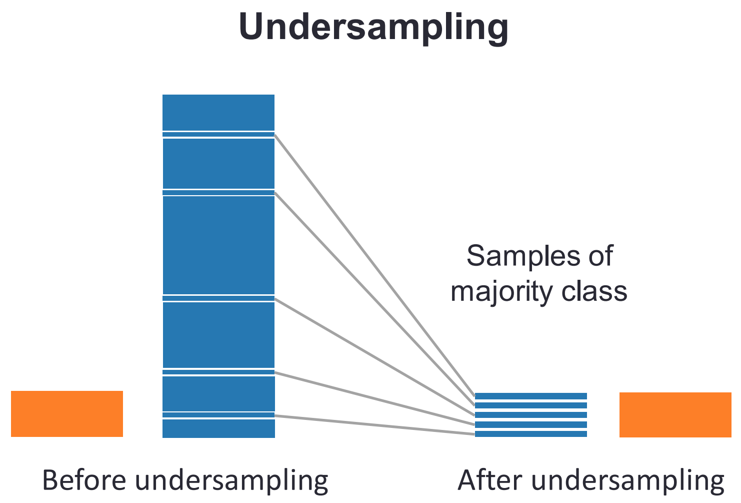
- Problem:
  - The estimation is usually worse for the class, which occurs more seldom
  - The seldom class might be the class with higher misclassification costs (e.g. decide no cancer, although the true class is cancer)



Adapted from: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

Imbalanced Classes: Undersampling

- Solution: undersampling or oversampling and synthetic oversampling
  - Undersampling:
    - remove examples from the over-represented class
      - In the example below: the "blue" data is undersampled and the "orange" data is not changed
    - https://imbalanced-learn.org/stable/under_sampling.html

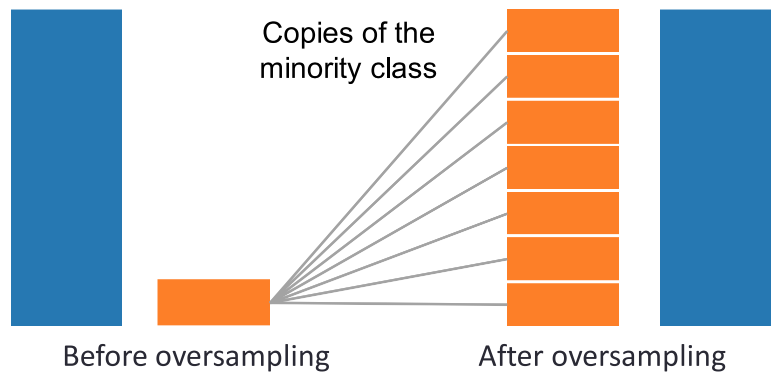


Note:
🟧 Positive class (e.g., cancer)
🟦 Negative class (e.g., no cancer)

Imbalanced Classes: Oversampling

- Solution: under- or oversampling and synthetic oversampling
  - Oversampling
    - Oversampling: clone data from the underrepresented class
      - In the example below: the “orange” data is oversampled and the “blue” data is not changed
    - Python: https://imbalanced-learn.org/stable/over_sampling.html#

Oversampling



Note:

🟧 Positive class (e.g., cancer)

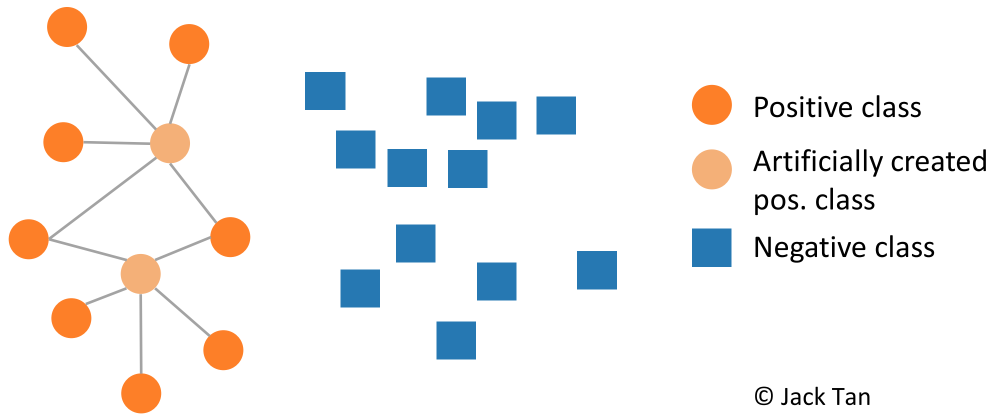
🟦 Negative class (e.g., no cancer)

Imbalanced Classes: Synthetic Oversampling

- Solution: under- or oversampling and synthetic oversampling
  - Synthetic oversampling:
    - Python: https://imbalanced-learn.org/stable/over_sampling.html#
    - Oversample the minority class by creating synthetic minority cases based on the k-nearest neighbors of the underrepresented class instances
    - There are two popular methods to over-sample minority classes:
      - (i) the Synthetic Minority Oversampling Technique (SMOTE) and
        (ii) the Adaptive Synthetic (ADASYN) sampling method.
  - In the example below:
    - the "red" data is the minority class and therefore synthetic observations are cerated
    - the "blue" data is not changed

SMOTE

(Synthetic Minority Oversampling Technique)



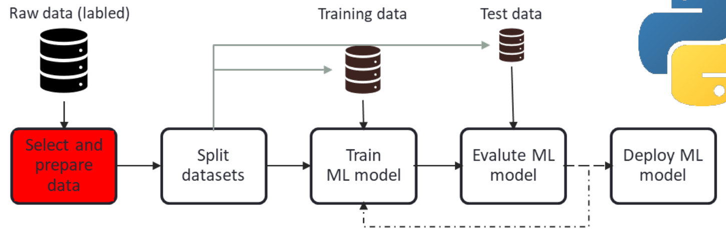
© Jack Tan

Remember the Running Example

- As a healthcare data analyst, your job is to identify patients or sufferers that have a higher chance of a particular disease, for example, diabetes
- These predictions will help you to treat patients before the disease occurs
- As an analyst, you have first to define the problem that you want to solve using classification and then identify the potential features that predict the labels accurately
    - Features are the columns or attributes that are responsible for prediction.
    - In diabetes prediction problems, health analysts will collect patient information, such as the number of pregnancies the patient has had, their BMI, insulin level, and age.

- Terminology:
    - Classification is the process of categorizing customers into two or more categories.
    - The classification model predicts the categorical class label, such as whether the customer is potential or not.
    - In the classification process, the model is trained on available data, makes predictions, and evaluates the model performance.
    - Developed models are called classifiers.

Imbalanced Classes: Python Example



Select and prepare data

```
# Import libraries
import pandas as pd

# read the dataset
diabetes = pd.read_csv("diabetes.csv")

# Show top 5-records
diabetes.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting dataset in two parts: feature set and target label
feature_set = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree','skin']

X = diabetes[feature_set]
y = diabetes.label
```

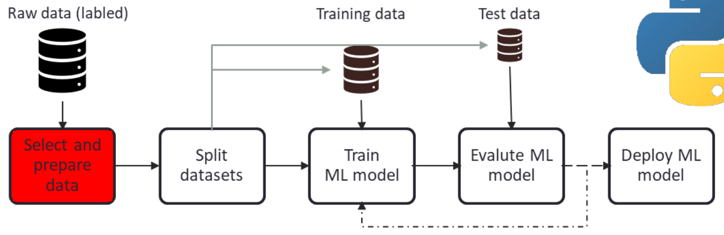
```
diabetes.label.value_counts()
```

```
0      500
1      268
Name: label, dtype: int64
```

- **Explanation:**
  - The dataset can be described as imbalanced, because we have 500 observations where the outcome is not present (i.e., 0) and 268 observations where the outcome is present (i.e., 1)

Imbalanced Classes: Python Example

INFORMATION SYSTEMS AND SERVICES
UNIVERSITY OF BAMBERG



Oversampling

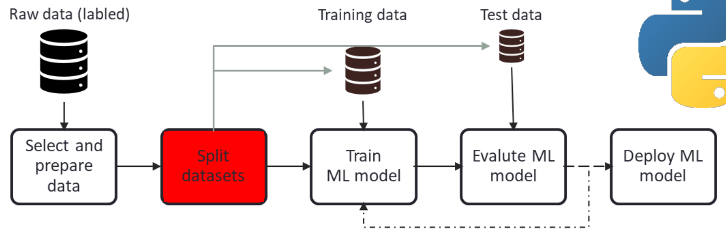
```python
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import RandomUnderSampler
from sklearn.metrics import SMOTE

ros = RandomOverSampler(random_state=0)
# rus = RandomUnderSampler(random_state=0) #Alternative for undersampling

X_oversampled, y_oversampled = ros.fit_resample(X, y)
# X_oversampled, y_oversampled = SMOTE().fit_resampled(X,y) #Alternative for SMOTE
```

```python
y_oversampled.value_counts()
```

```
1     500
0     268
Name: label, dtype: int64
```

- **Explanation:**
  - In this example, we use oversampling
  - After applying oversampling, we have 500 observations for the presence and absence of the outcome

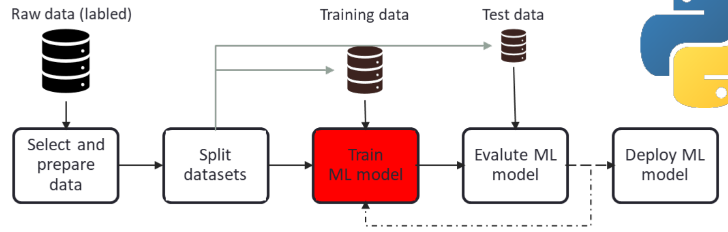
Imbalanced Classes: Python Example



Split dataset

```
# Partition data into training and testing set
from sklearn.model_selection import train_test_split
X_train_OS,X_test_OS, y_train_OS, y_test_OS = train_test_split(X_oversampled, y_oversampled,
test_size=0.3, random_state=42)
```

Imbalanced Classes: Python Example



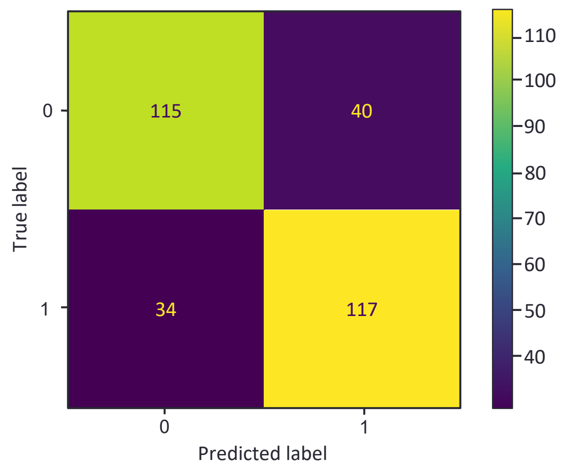
Train decision tree model

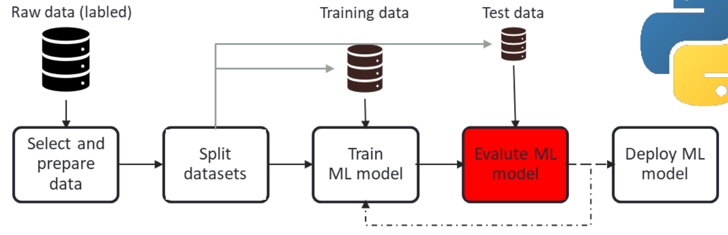
```
# Import Decision Tree model
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier object
clf_tree_OS = DecisionTreeClassifier(random_state = 42, min_samples_leaf = 0.04)

# Train the model using training dataset
clf_tree_OS = clf_tree.fit(X_train_OS,y_train_OS)
```

Imbalanced Classes: Python Example

Evaluate Decision tree model (test data)

```
# Predict the response for test dataset
y_pred_test = clf_tree.predict(X_test_OS)

# Import the confusion matrix
from sklearn.metrics import plot_confusion_matrix

# Plot Confusion matrix
plot_confusion_matrix(clf_tree_OS , X_test_OS, y_test_OS, values_format='d')
```





```
# Oversampling
# Import metrics module for performance evaluation
from sklearn.metrics import accuracy_score

# Calculate model accuracy
print("Accuracy:",accuracy_score(y_test_OS, y_pred_test_OS))
```

```
Accuracy: 0.7533333333333333
```

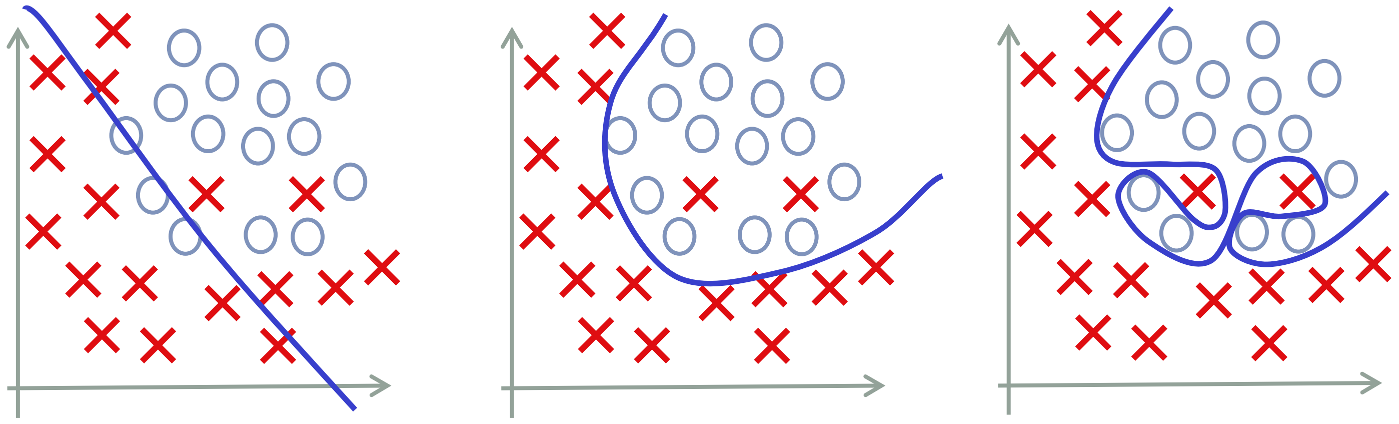
- **Explanation:**
  - The accuracy bevor oversampling was 0.71 (see chapter on decision trees)
  - The oversampling has improved the accuracy of the test data

Advanced Topics for Supervised Learning

"Bad algorithms" with under- or over-fitted models

- Usual goal when generalizing from data: maximize the accuracy of future classifications from the same data source
  - But: the accuracy is not maximized by learning the training data as precisely as possible
  - Instead: extremely fitting the training data tends to generalize poorly to future data, because thereby random aspects of the sample (e.g., noise) are also fitted, in addition to the regular data
  - Thereby, the broader regularities in the data are missed
  - Example: a student who memorizes every last detail of the teacher's lecture can only do so at the expense of the broader ideas in the lesson

- Fitting training data too closely in this sense—fitting noise as well real trends—is often referred to as <u>overfitting</u>
- Fitting it too loosely—missing real trends as well as noise—is called <u>underfitting</u>
- The goal is to find an <u>appropriate-fitting</u> model


Under-fitting Appropriate-fitting Over-fitting

“Bad algorithms” with under- or over-fitted models (contd.)

- Overfitting describes the problem of overfitting a model to the training data.
  - The "learned" classifier is optimized for the data of the training set but delivers significantly worse results on the population of the data.
  - Due to this overfitting, the ability to generalize is lost.
  - On unknown data, divergent patterns are not recognized correctly.
  - After all, too specific rules have been generated.

- Real-world data is usually impure and error-prone.
  - In this context, one also likes to speak of noisy data.
  - Incomplete and/or incomplete and/or erroneous data can be reworked by suitable preprocessing.

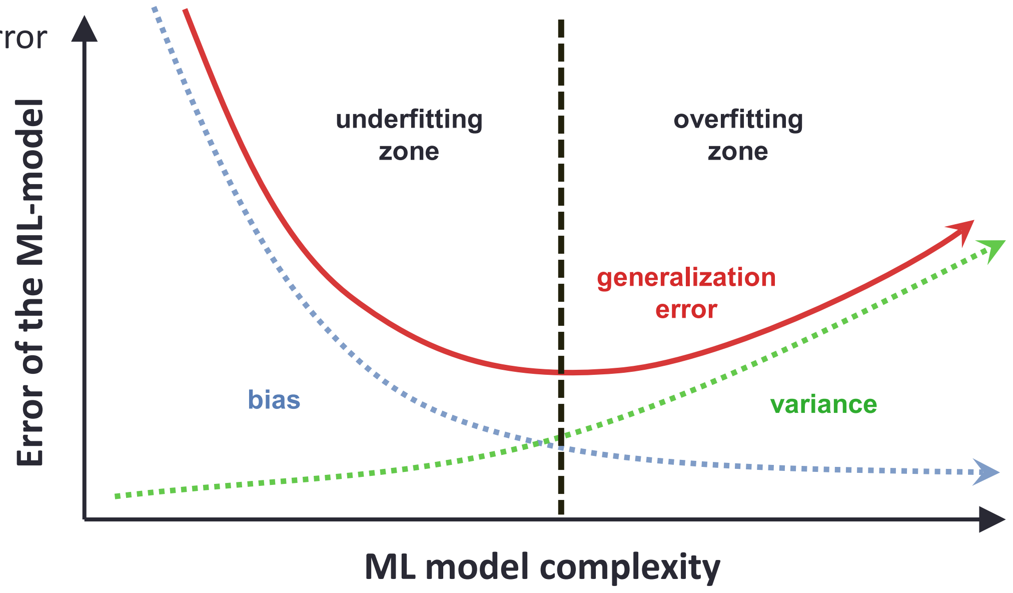
- Overfitting can also occur when training data selection is too small or is not a representative sample of the base dataset.
  - In cases, it is inadmissible to infer the validity of the rules in all data sets. to be inferred.

- Overfitting can occur with all classification algorithms.
  - The difficulty lies in detecting the effect of overfitting promptly.
  - This must be decided on a case-by-case basis.
  - Incorrectly classified data sets can usually only be detected and corrected with great, mostly manual effort.

Generalization error

- Generalization error of a supervised machine learning model consists of two main components
  - Bias:
    - The bias term shows, on average, how much the supervised machine learning model is different from the actual underlying model in the data
    - We want to reduce the bias, because high bias leads to underfitting
  - Variance:
    - The variance term shows how much the supervised machine learning model is inconsistent over different training sets
    - We want to reduce variance, because high variance leads to overfitting

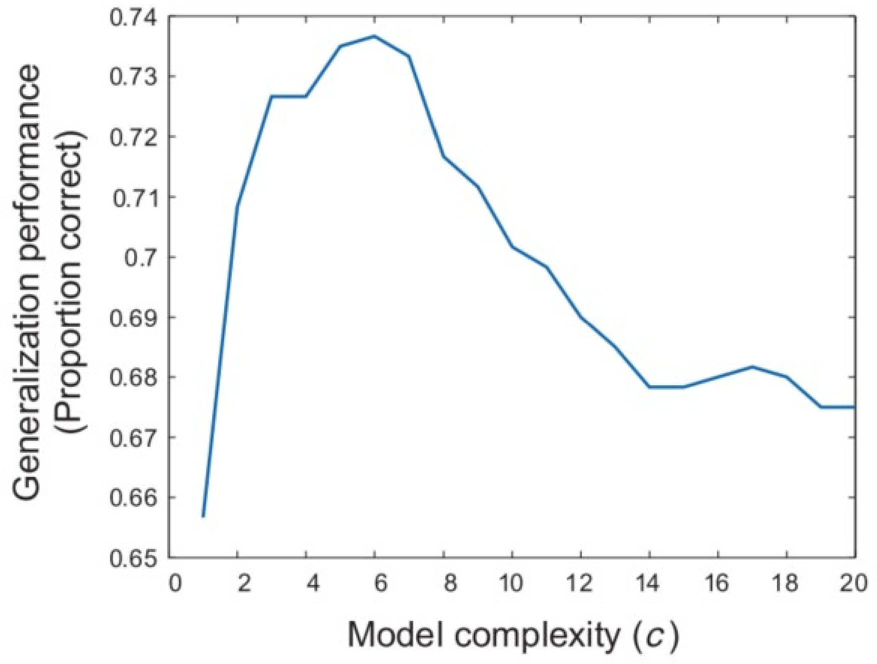
- As a data scientist, you can adjust the complexity of a machine learning (ML) model
  - When the ML model complexity increases, the variance increases while the bias decreases.
  - Conversely, when the ML model complexity decreases, variance decreases, and bias increases.
  - The goal is to find the ML model complexity that achieves the lowest generalization error



Overfitting

- The critical variable influencing overfitting is the complexity of the ML models
  - More complex ML models (e.g., decision trees) can fit the training data more closely than less complex models (e.g., linear regression)
  - Less complex ones may lack the flexibility to fit it very well

- Accuracy first rises to some optimum, but then declines as overfitting sets in
  - Exactly where the optimal performance lies depends on the nature of the patterns to be learned
  - At one extreme (simple ML model), the model imposes a strong expectation or ‘‘bias’’ on the data, sacrificing fit.
  - At the other extreme (complex model), the models are more flexible (i.e., exhibit greater variance), risking overfitting



Overfitting

- Strategies to reduce overfitting:
  - Removal of incorrect training data (“data cleaning”)
  - Selection of suitable size of the training set
  - Classifier specific measures, e.g., decision trees can use pruning or a minimum number of observations per node
- Next, we discuss how cross-validation can help to detect if a model suffers from overfitting or underfitting

You can learn all this in an interactive online course!

https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/the-bias-variance-tradeoff?ex=1

Generalization Error

MACHINE LEARNING WITH TREE-BASED MODELS IN PYTHON

Elie Kawerk
Data Scientist

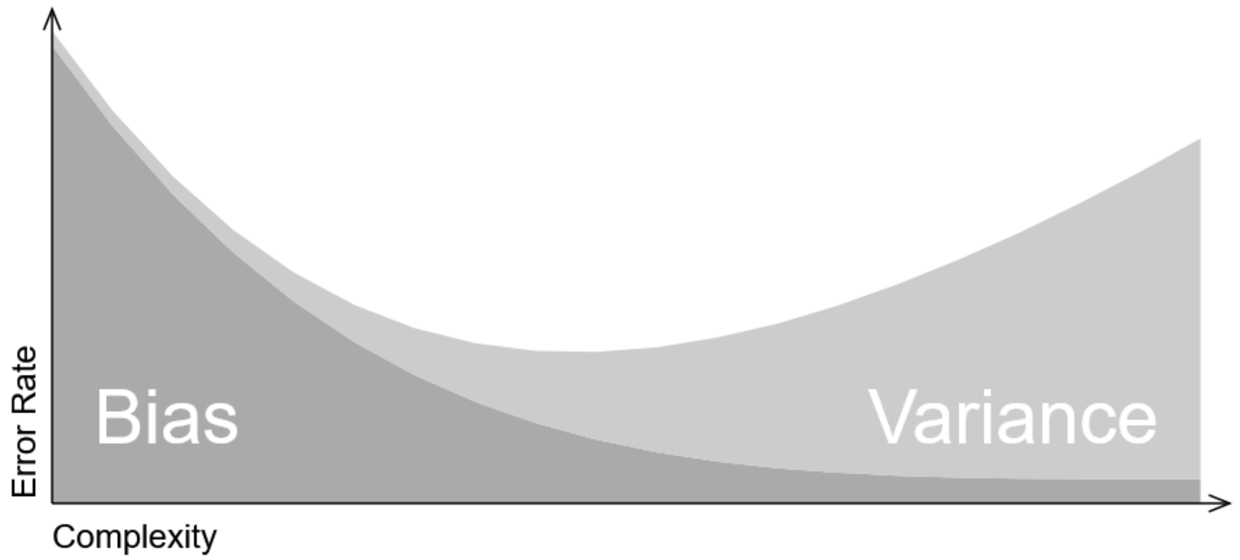
datacamp

Model Tuning and the Bias-Variance Tradeoff: A Visual Introduction

On the other hand, when a model is less complex, error due to variance is low. Error due to variance increases as complexity increases.

Overall model error is a function of error due to bias plus error due to variance. The ideal model minimizes error from each.

You can actually show mathematically that error sue to bias and error due to variance are distinct.



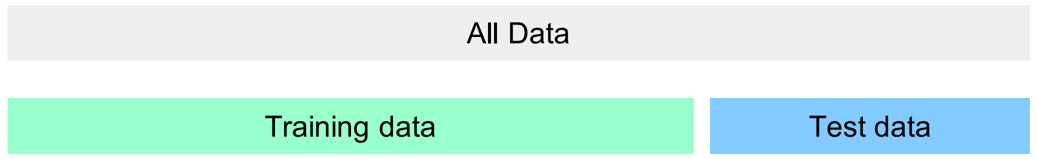
An intuitive visualization for bias-variance tradeoff: http://www.r2d3.us/visual-intro-to-machine-learning-part-2/

Advanced Topics for Supervised Learning

Holdout method

- Training and test data split
  - We are interested in measuring how well our trained machine learning model generalizes to new, previously unseen data
  - So far, we train a model using the **train set** and the calculation of the machine learning model performance is performed using the **test set**
  - This has several potential drawbacks
    - The calculation of machine learning model performance can be highly variable (i.e., **high variance/overfitting)**, depending on precisely which observations are included in the training set and which observations are included in the test set
    - The machine learning model will perform well if randomly all "complicated" observations are in the training data and the "easy" observations are in the test data
    - We do not know how sensitive our model is to the "random" selection of training data

- One method to overcome these drawbacks is cross-validation
  - **Non-exhaustive cross-validation**
    - Does not learn and test on all possible splits
    - E.g., k-fold cross-validation (next slide)
  - **Exhaustive cross-validation**:
    - Learn and test all ways to divide the labeled sample
    - Not covered here

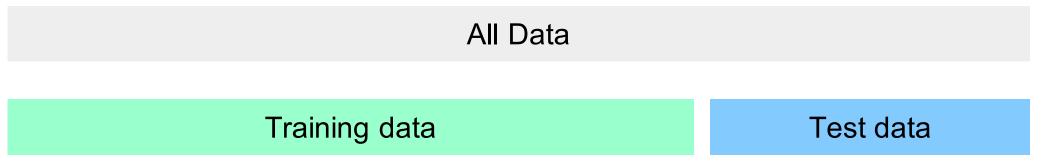


Holdout method (contd.)

- Remember that train_test_split splits the dataset randomly.
  - If we are "lucky" and all difficult to predict examples end up in the training dataset, there will be only "easy" examples in the test data.
  - In this case, the accuracy for the test data set would be unrealistically high.
  - If, on the other hand, we are "unlucky," all examples that are difficult to classify will end up in the test data, and the accuracy will thus be unrealistically low.

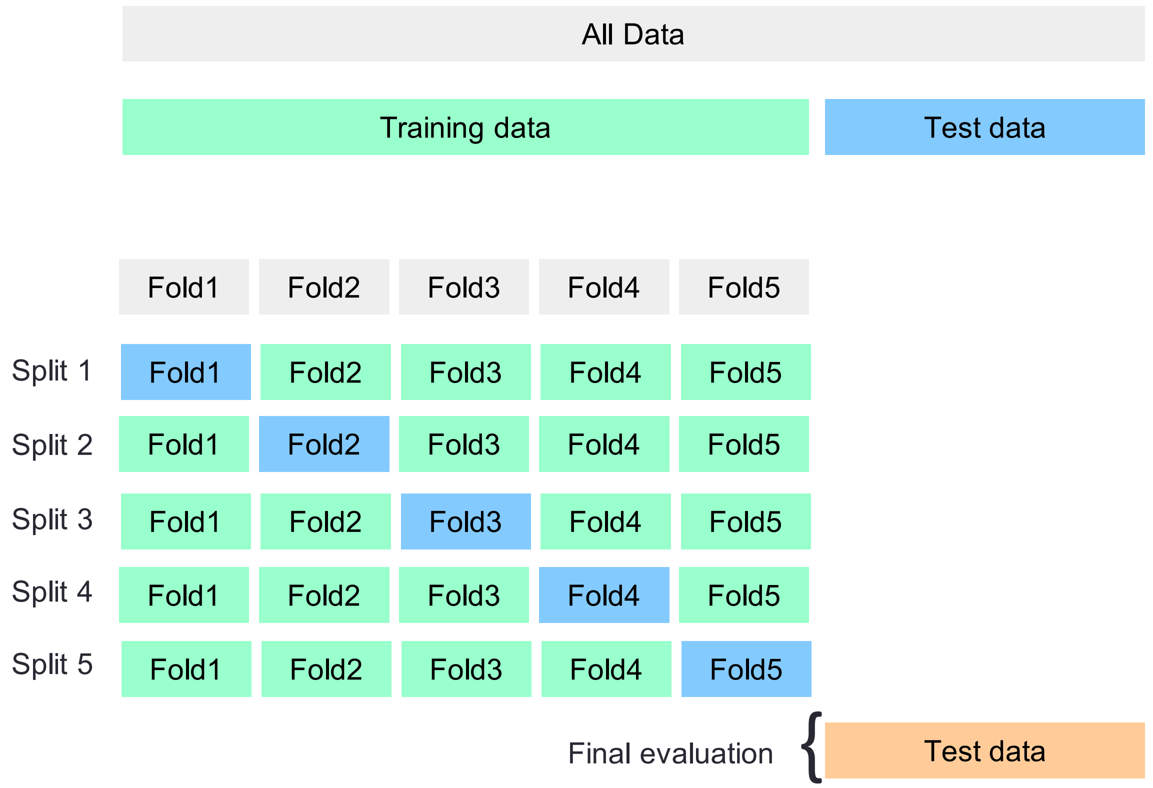
- Cross-validation is a statistical method of evaluating generalization performance that is more stable and thorough than using a split into a training and a test set.

- In cross-validation, the data is instead split repeatedly, and multiple models are trained.



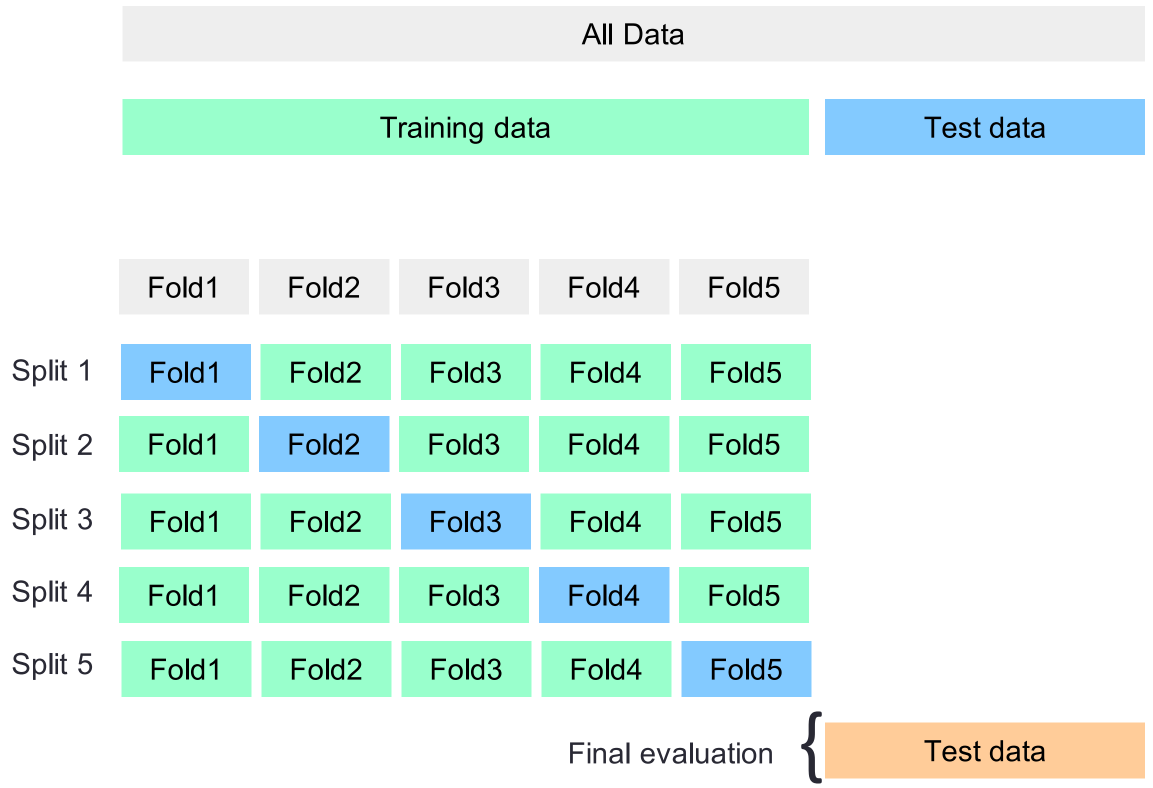
K-fold Cross Validation

- K-fold cross-validation (CV)
  - This approach involves randomly, dividing the set of observations into k folds, of approximately equal size.
  - The first fold is treated as a test set, and the model is fit on the remaining k − 1 folds.
    - K is normally between 5 and 10
  - This procedure is repeated k times; each time, a different group of observations is treated as a test set.
  - This process results in k estimates of machine learning model performance
  - The overall k-fold CV performance estimate is computed by averaging these evaluation scores values (e.g. accuracy)
  - Other statistics, such as the standard deviation of the evaluation scores (e.g., accuracy) or the range of the evaluation scores, can help to interpret if the machine learning model suffers from overfitting or underfitting



K-fold Cross Validation (contd.)

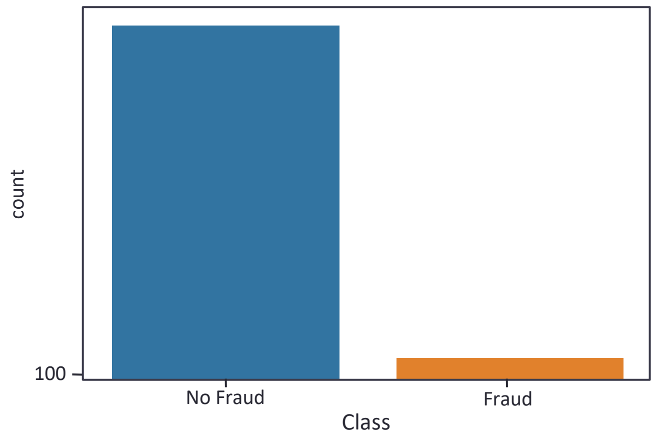
- The most commonly used version of cross-validation is k-fold cross-validation, where k is a user-specified number, usually 5 or 10.
  - When performing five-fold cross-validation, the data is first partitioned into five parts of (approximately) equal size, called folds. Next, a sequence of models is trained.
  - The first model is trained using the first fold as the test set, and the remaining folds (2–5) are used as the training set.
  - The model is built using the data in folds 2–5, and then the accuracy is evaluated on fold 1.
  - Then, another model is built, this time using fold 2 as the test set and the data in folds 1, 3, 4, and 5 as the training set.
  - This process is repeated using folds 3, 4, and 5 as test sets. For each of these five splits of the data into training and test sets, we compute the accuracy. In the end, we have collected five accuracy values.



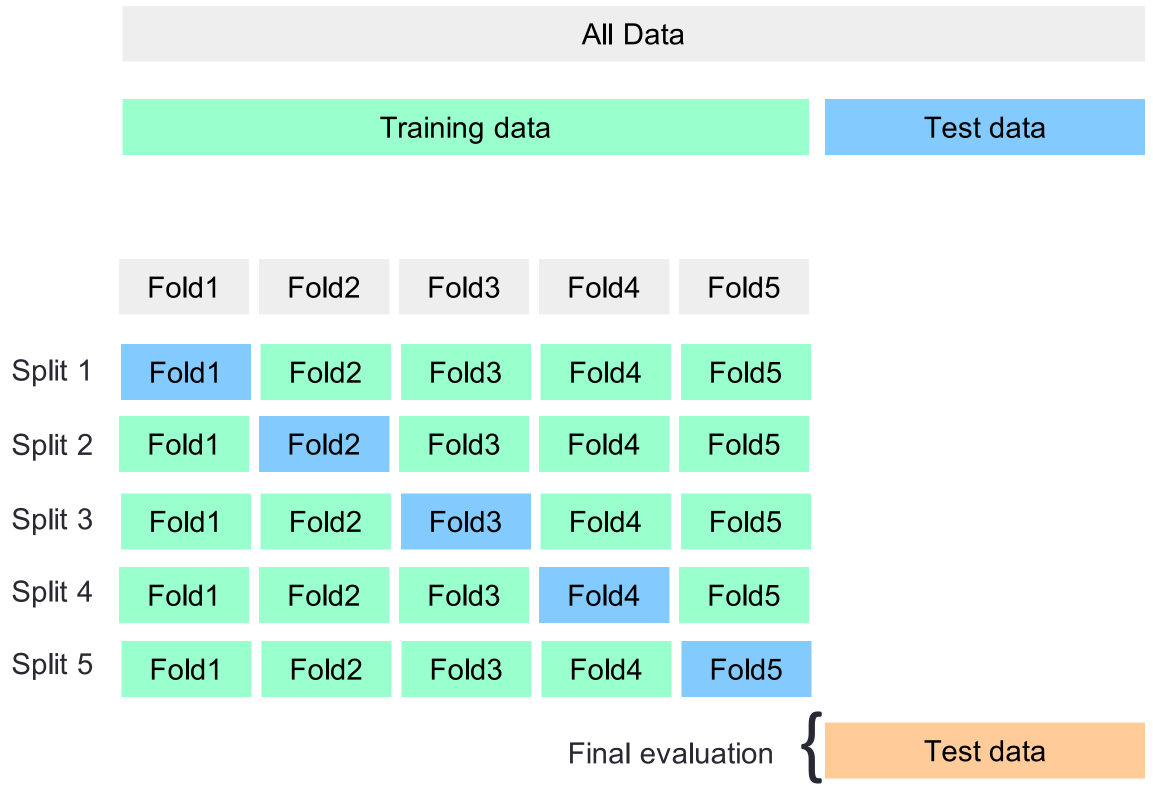
Additional info: https://scikit-learn.org/stable/modules/cross_validation.html

Stratified K-fold Cross-Validation

- Inbalanced classes and cross-validation
  - In many data sets, we only have a small percentage of the data, where the to be predicted class exists (see slides on imbalanced data)
    - For instance: fraud detection
    - There is often only a small number of observations of fraud

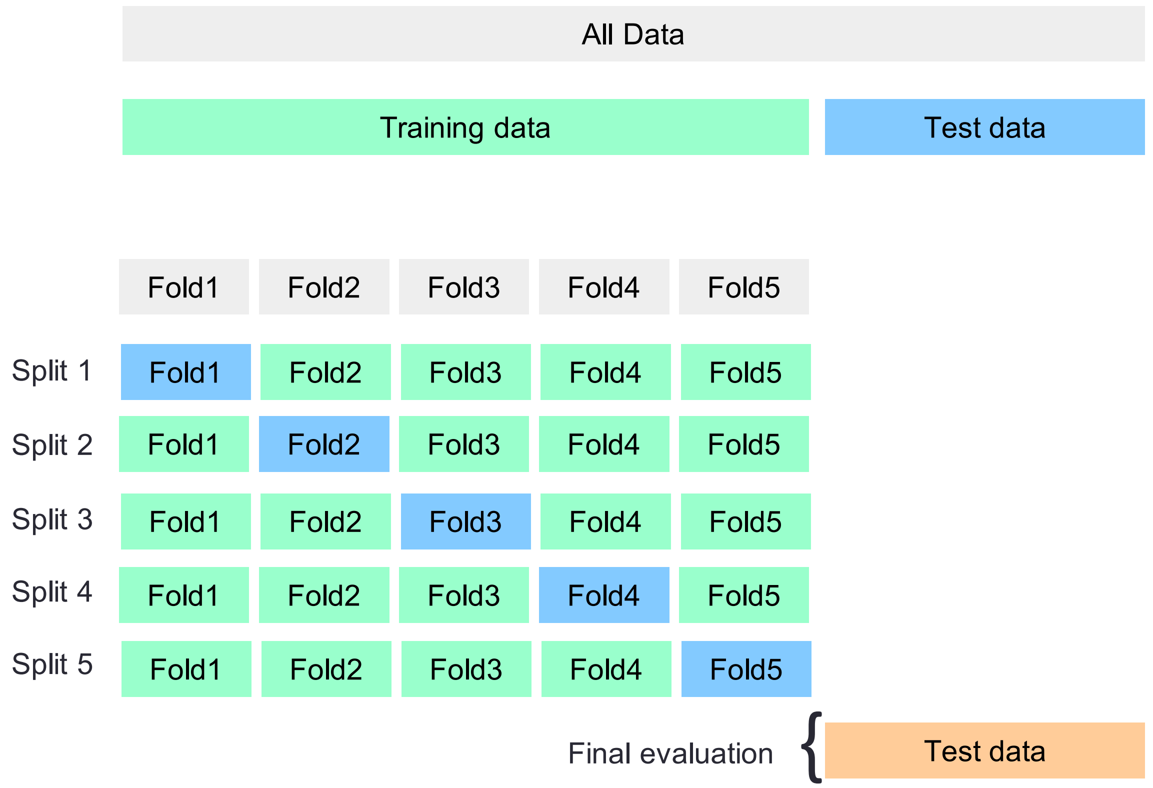


  - An unbalanced cross-validation is not very accurate
    → stratified k-fold cross-validation can be used
- Stratified k-fold cross-validation
  - The data is spitted, such that the proportions between classes are the same in each fold as they are in the whole dataset



Stratified K-fold Cross-Validation (contd.)

- For example, if 90% of a sample belong to class A and 10% of a sample belong to class B, then stratified cross-validation ensures that in each fold, 90% of a sample belong to class A and 10% a of sample belong to class B.
  - It is usually a good idea to use stratified k-fold cross-validation instead of k-fold cross-validation to evaluate a classifier, because it results in more reliable estimates of generalization performance
  - In the case of only 10% of a sample belonging to class B, using standard k-fold cross-validation it might easily happen that one fold only contains samples of class A.
  - Using this fold as a test set would not be very informative about the overall performance of the classifier.



Comparing performance of CV, Train and Test

- Using cross-validation to estimate the generalization error
  - We have several evaluations of the ML model‘s performance
    - Performance (e.g., accuracy) of the ML model on the training set
    - Performance (e.g., accuracy) of the ML model on the test set
    - Performance (e.g., accuracy) of the ML model on cross-validation

- If the performance of the cross-validation is smaller than the performance of the training set
  - The ML model might suffer from high variance
  - The ML model seems to suffer from overfitting
  - It might be good to decrease the complexity of the ML model

- If the performance of the cross-validation is nearly equal performance of the training set but smaller than the desired performance
  - The ML model suffers from high bias
  - The ML model seems to suffer from underfitting
  - It might be good to increase the complexity of the ML model

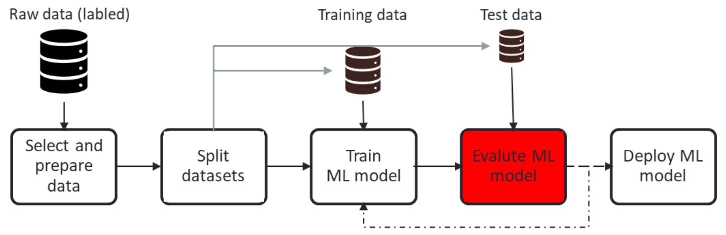
Cross-validation in Scikit-Learn

- Cross-validation is implemented in scikit-learn using the cross_val_score function from the model_selection module
  - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score

- Tip 1:
  - It is important to keep in mind that cross-validation is not a way to build a model that can be applied to new data.
  - Cross-validation does not return a model.
  - When calling cross_val_score, multiple models are built internally, but the purpose of cross-validation is only to evaluate how well a given algorithm will generalize when trained on a specific dataset.

- Tip 2:
  - Scikit-Learn's cross-validation features expect a utility function (where greater values are better) rather than a cost function (where lower values are better),
  - The scoring function for regression is the opposite of the mean squared error (MSE) (i.e., a negative value), which is why the preceding code computes -scores before calculating the square root.

Cross Validation: Python Example



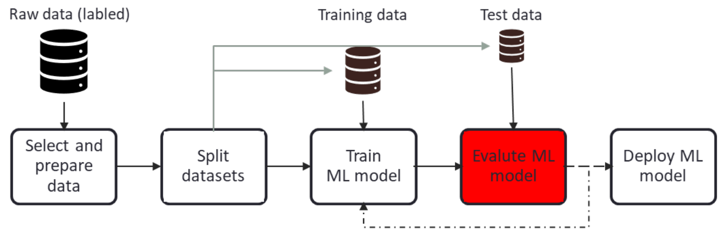
Cross Validation

```
from sklearn.model_selection import cross_val_score

scores =  cross_val_score(clf_tree_OS, X_train_OS,y_train_OS, scoring='accuracy')
print("Cross-validation scores: {}". format(scores))
print("Average cross-validation score: {:.2f}".format(scores.mean()))
print("Standard deviation: {:.2f}".format(scores.std()))
```

```
Cross-validation scores: [0.76428571 0.73571429 0.77142857 0.70714286 0.77857143]
Average cross-validation score: 0.75
Standard deviation: 0.03
```

- Using the mean cross-validation we can conclude that we expect the model to be around 75% accurate on average
- Looking at all five scores produced by the five-fold cross-validation, we can also conclude that there is a relatively high variance in the accuracy between folds, ranging from 70% accuracy to 77% accuracy.
- This could imply that the model is very dependent on the particular folds used for training.

Cross Validation: Python Example



```
# accuracy scores
y_pred_train_OS = clf_tree_OS.predict(X_train_OS)

print(“Accuracy of training data:", accuracy_score(y_train_OS, y_pred_train_OS))
print(“Accuracy of test data:", accuracy_score(y_test_OS, y_pred_test_OS))
print(“Accuracy of cross-validation:", cross_val_score(clf_tree_OS, X_train_OS,y_train_OS, scoring='accuracy').mean())
```

```
Accuracy of training data: 0.7985714285714286
Accuracy of test data: 0.7533333333333333
Accuracy of cross-validation: 0.7514285714285714
```

- The accuracy of the cross validation is worse than the performance of the training set, thus the model seems to suffer from high variance (i.e. overfitting).

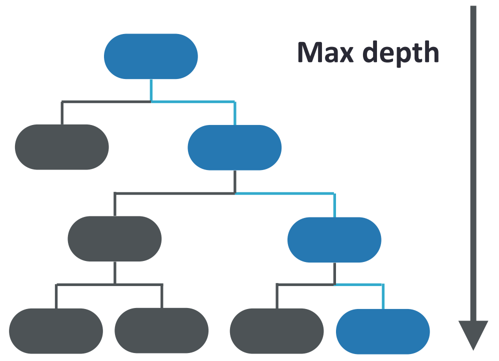
Summary

- Using cross-validation can help to estimate the generalization error, i.e., how well our machine learning model performs
- Often, the result is that we either need to increase or decrease the complexity of our machine learning model
- The complexity of our machine learning model can be adjusted through hyperparameters
  - Trying out different hyperparameters is called hyperparameter-tuning
  - In the next slides, we discuss this important topic

Advanced Topics for Supervised Learning

Hyperparameters

- Hyperparameters
  - Are not learned from data
  - Are set before training
  - A hyperparameter is a parameter that is set before the learning process begins. These parameters are tunable and can directly affect how well a model trains.
  - Example hyperparameter for Classification and Regression Trees: max_depth



- Machine learning models can be optimized by trying different hyperparameter
  - One option would be to fiddle with the hyperparameters manually until we find a great combination of hyperparameter values
  - This is very tedious work, and we may not have time to explore many combinations.

Finding the best Hyperparameters

- Problem: search for a set of optimal hyperparameters for a machine learning algorithm
- Solution: find a set of optimal hyperparameters that results in an optimal machine learning model
- Optimal model: yields an optimal performance score
  - E.g., accuracy in classification
  - E.g., R^2 in regression

- Cross validation should be used to estimate how well the model performance

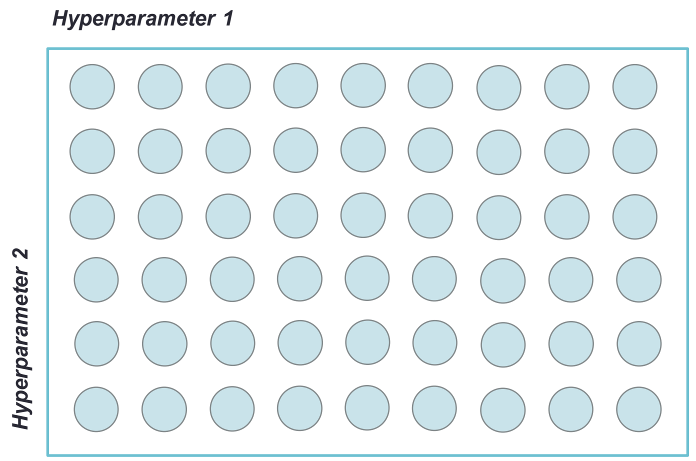
- Multiple approaches to hyperparameter tuning exist
  - Grid Search
  - Random Search
  - Bayesian Optimization (not covered here)
  - Genetic Algorithms (not covered here)

Advanced Topics for Supervised Learning

Grid search

- Grid search:
  - Search exhaustively over a given set of hyperparameters, once per set of hyperparameters
  - The number of models equals the number of distinct values per hyperparameter multiplied across each hyperparameter (for the example on the right: 3*3 = 9)

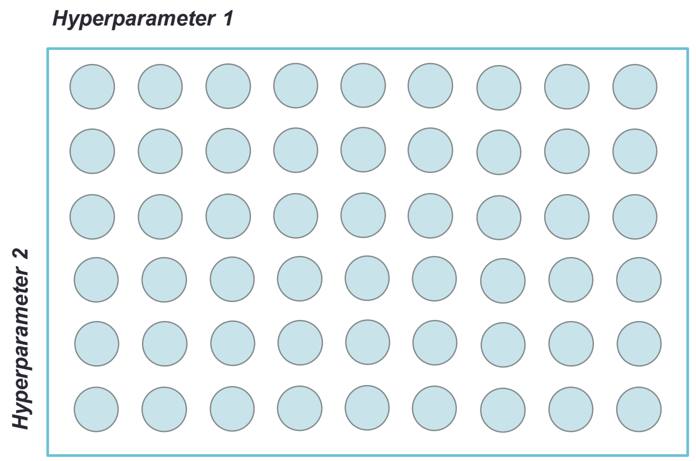
- Grid search procedure
  - Manually set a grid of discrete hyperparameter values
    - E.g., max depth of 2, 3, or 4
      max_depth = {2,3,4},
    - E.g, minimum samples in a leaf of 5% or 10%
      min_samples_leaf = {0.05, 0.1, 0.2}
  - Set a metric for scoring model performance (e.g., calculate accuracy)
  - Search exhaustively through the grid
    - Here: we have two hyperparameters with 3 possible values
      →grid search tries all 9 different hyperparameter configurations
  - For each set of hyperparameters, evaluate each model's CV (cross-validation) score.
  - The optimal hyperparameters are those of the model achieving the best CV (cross-validation) score.



Grid search

- Grid search limitations
  - The number of ML models you must build with every additional new parameter grows very quickly
  - For large data sets with many hyperparameters, we need a long time for finding the best hyperparameters

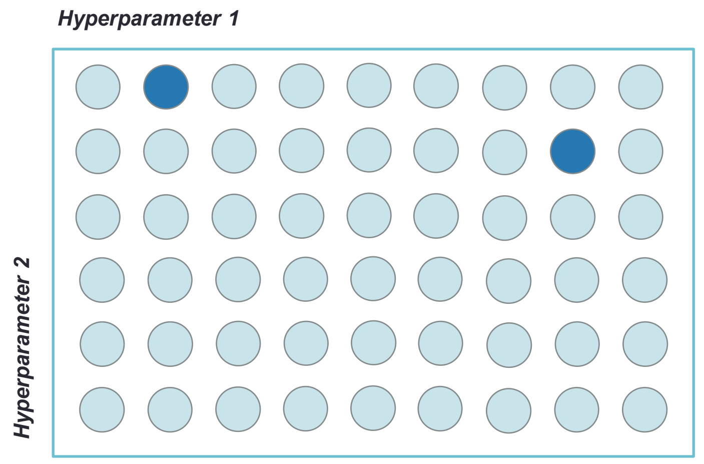
- Solution: hyperparameter tuning with random search
  - Grid search is one of the most popular methods for hyperparameter tuning
  - However, grid search can get computationally expensive if you are searching over a large hyperparameter space and dealing with multiple hyperparameters.
  - A possible solution is a random search



Advanced Topics for Supervised Learning

Hyperparameter tuning: Random Search

- Random search
  - Doesn't try all the hyperparameter values
  - A fixed number of hyperparameter settings is sampled from specified probability distributions
  - The optimal hyperparameters are those of the model achieving the best CV (cross-validation) score



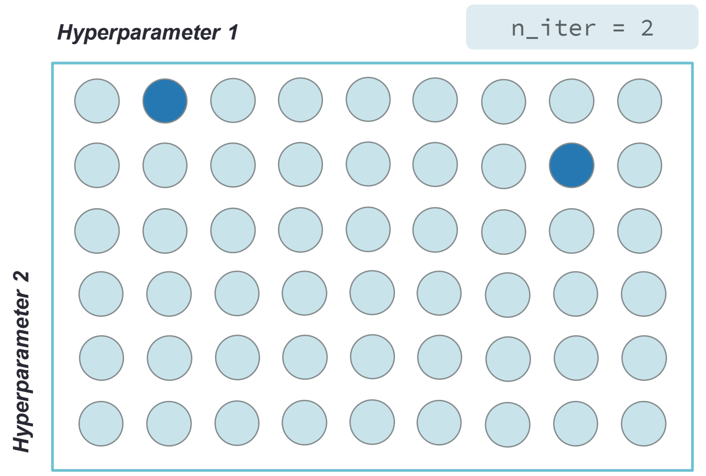
Key:		
	○	not tested hyperparameter combination
	●	tested hyperparameter combination

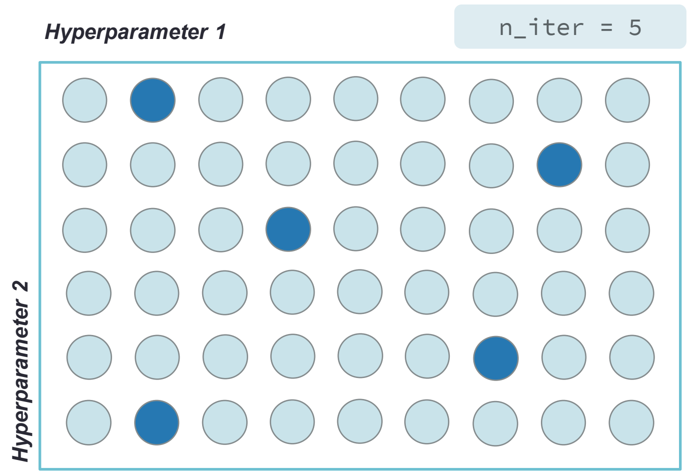
Hyperparameter tuning: Random Search

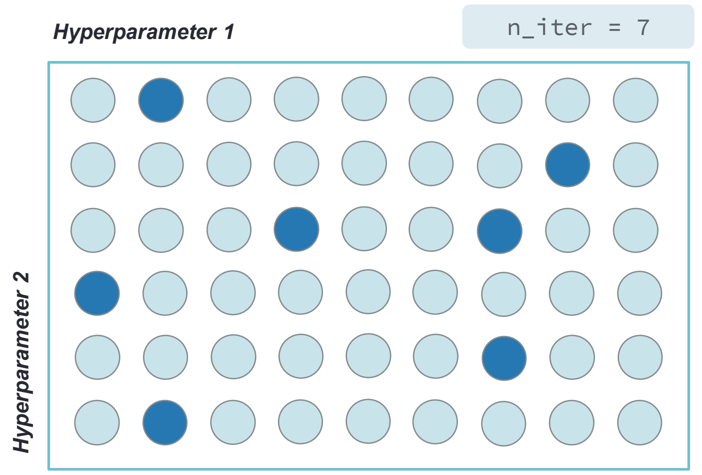
- The number of interations (n_iter) determines how many random combinations of the hyperparameter are tested

Key:

○	not tested hyperparameter combination
●	tested hyperparameter combination

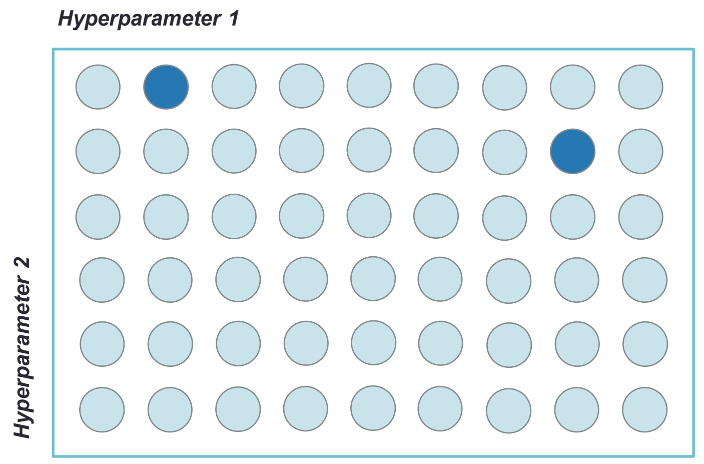






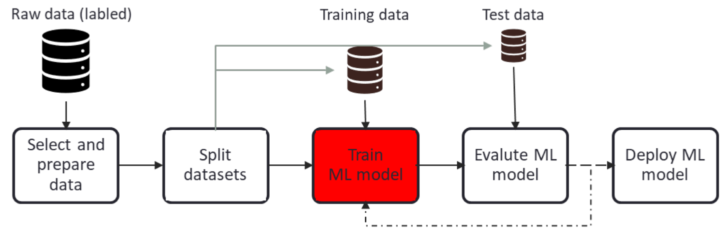
Hyperparameter tuning: Random Search

- Random Search Limitations
  - Parameter space to explore can be massive
  - Randomly jumping throughout the space looking for a "best" result becomes a waiting game
  - Results may not be the best solution



Key:		
	○	not tested hyperparameter combination
	●	tested hyperparameter combination

Grid Search Cross Validation: Python Example

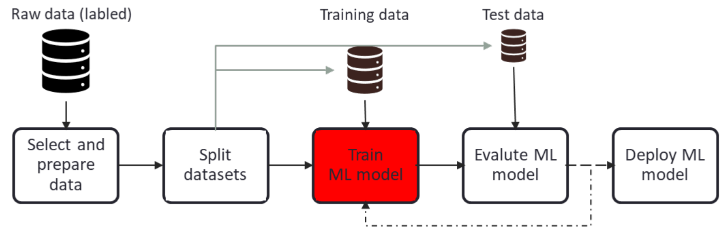


```python
from sklearn.model_selection import GridSearchCV

# Define the grid of hyperparameters
params_tree_clf = {
'max_depth': [1,2,3,4,5,6,7,8],
'min_samples_leaf': [0.0001,0.001,0.01,0.04, 0.06, 0.08, 0.1, 0.2],
'max_features': [0.2,0.4,0.6,0.8, 1],
'criterion' :['gini', 'entropy']
}
```

```python
# Instantiate a 10-fold CV-grid search object
grid_clf_tree_OS = GridSearchCV(DecisionTreeClassifier(random_state=42),param_grid=params_tree_clf,scoring='accuracy',
cv=5,
n_jobs=-1)

# Fitd CV-grid search object to the training data
grid_clf_tree_OS.fit(X_train_OS,y_train_OS)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
                         'max_features': [0.2, 0.4, 0.6, 0.8, 1],
                         'min_samples_leaf': [0.0001, 0.001, 0.01, 0.04, 0.06,
                                              0.08, 0.1, 0.2]},
             scoring='accuracy')
```

Grid Search Cross Validation: Python Example



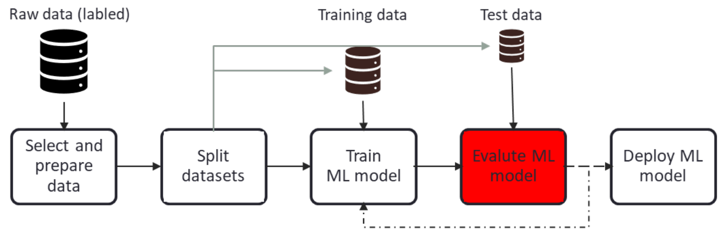
```
# Print the best hyperparameters
print('Best hyperparameters:\n', grid_clf_tree_OS.best_params_)
```

```
Best hyperparameters:
 {'criterion': 'entropy', 'max_depth': 8, 'max_features': 0.6, 'min_samples_leaf': 0.0001}
```

```
# Print the best CV score
print(grid_clf_tree_OS.best_score_)
```

```
0.8014285714285714
```

Grid Search Cross Validation: Python Example



```
#Training the model according to the new parameter
clf_tree_OS_GS = DecisionTreeClassifier(criterion='entropy', max_depth=8, max_features=0.6,
min_samples_leaf=0.0001, random_state=42)

clf_tree_OS_GS.fit(X_train_OS, y_train_OS)

#Accuracy scores
#train
print(“Accuracy of training data:", accuracy_score(y_train_OS,
clf_tree_OS_GS.predict(X_train_OS)))
#test
print(“Accuracy of test data:", accuracy_score(y_test_OS, clf_tree_OS_GS.predict(X_test_OS)))
#CV
print(“Accuracy of grid search cross-validation:", grid_clf_tree_OS.best_score_)
```

```
Accuracy of training data: 0.9128571428571428
Accuracy of test data: 0.7666666666666667
Accuracy of grid search cross-validation: 0.8014285714285714
```

Advanced Topics for Supervised Learning

Challenges of Machine Learning

- Bad Data
    - The curse of dimensionality
    - Imbalanced data
- Bad Algorithms

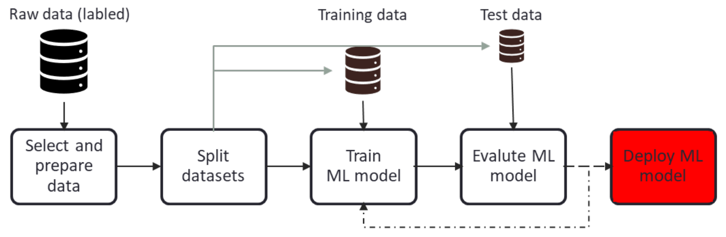
Cross-Validation

Hyperparameter Tuning

- Grid search
- Random search

Deployment

Deploying ML model

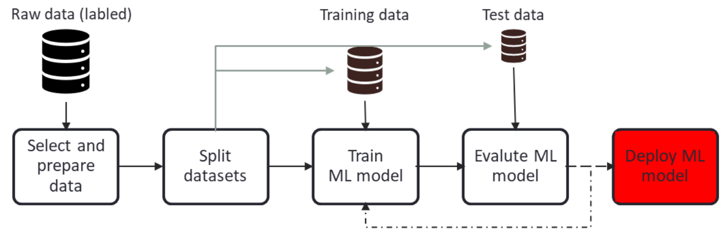


- There are various ways to deploy an ML model to the production environment

- 1) Export scikit-learn model to the production environment
  - Develop locally on your device
  - Export the trained machine learning model, e.g., using the joblib library or
  - Import the trained model into the production environment
  - Make predictions by calling the predict() method

  - Example:
    - The user can type in some in a form on a website and can submit the data
    - This will send a query containing the data to the web server, which will forward it to the web application, and finally, your code will simply call the predict() method

Grid Search Cross Validation: Python Example



Exporting

```python
import joblib
```

```python
joblib.dump(clf_tree_OS_GS, 'clf_tree_OS_GS.pkl')
```

```
['clf_tree_OS_GS.pkl']
```

Importing

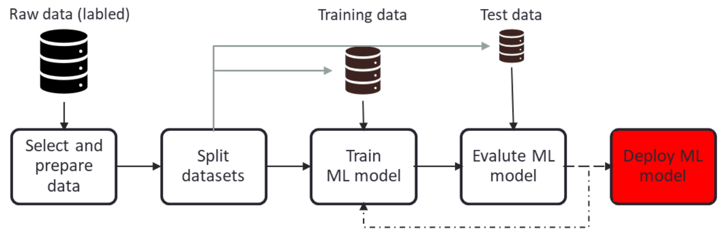
```python
myImportedModel = joblib.load('clf_tree_OS_GS.pkl')
```

Using Imported model

```python
print(myImportedModel.predict(X_test))
```

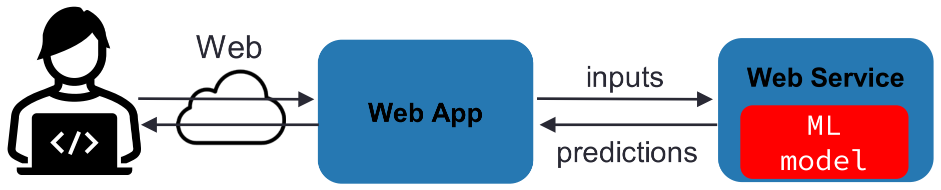
```
[0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 1 1 1 0 1 1
 0 0 1 0 1 1 1 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 0
 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0
 0 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 1
 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
 0 0 0 1 0 0 0 1 0]
```

Deploying ML model

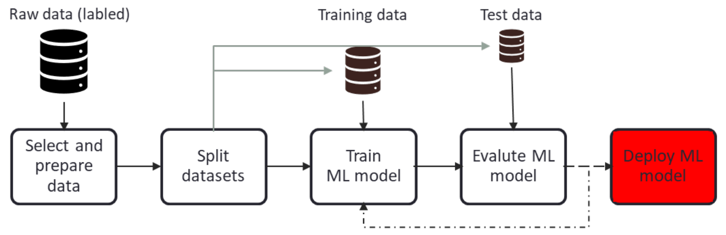


- There are various ways to deploy an ML model to the production environment

- 2) Wrap the model within a dedicated web service that your web application can query through a REST API
  - Using REST API makes it easier to upgrade a machine learning model to new versions without interrupting the main application.
  - It also simplifies scaling, since you can start as many web services as needed and load-balance the requests coming from your web application across these web services.
  - It allows your web application to use any language, not just Python (as in the first scenario).



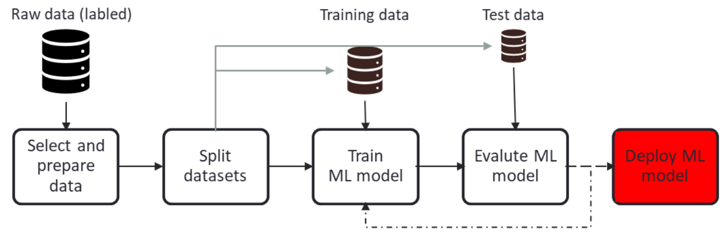
Deploying ML model



- There are various ways to deploy an ML model to the production environment

- 3) Deploy your model on the cloud (e.g., Google Cloud, Amazon Web Service, Microsoft Azure)
  - Example for deploying with Google Cloud:
    - Save your model using joblib and upload it to Google Cloud Storage (GCS)
    - In the Google Cloud AI Platform, you can create a new model version, pointing it to the GCS file
    - Could platforms thereby provide a simple web service
    - Cloud platforms can automatically take care of load balancing and scaling
    - The predict() method can be called by a JSON request containing the input data and returns JSON responses containing the predictions.
    - The request can be called from any authorized application

Deploying ML model



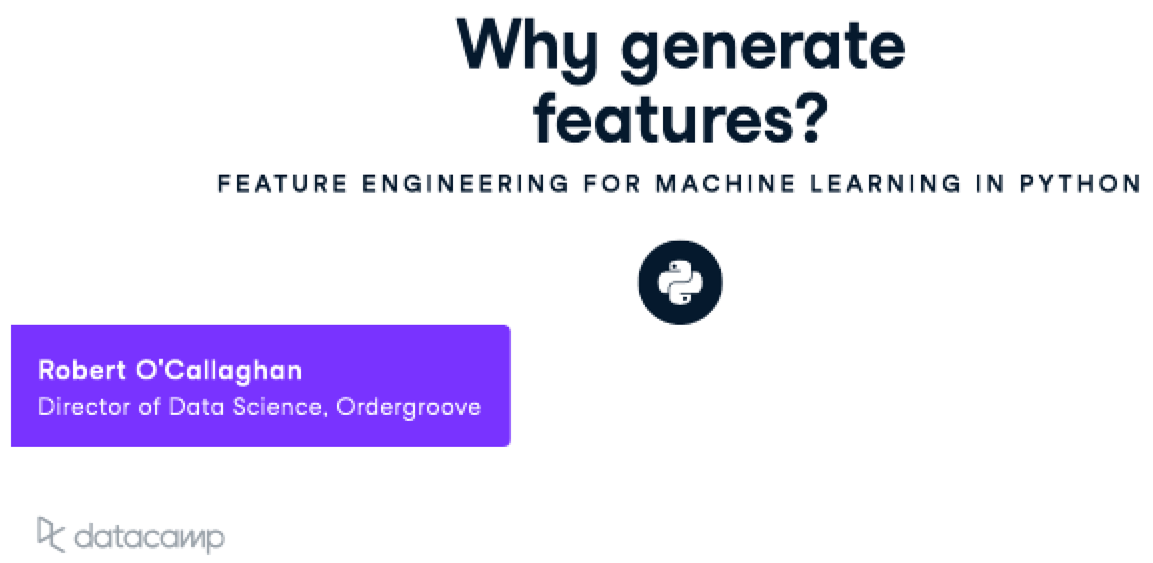
- ML models tend to “rot” over time:
  - If the model was trained with last year’s data, it may not be adapted to today’s data
- After deployment:
  - Monitoring the performance
    - Automatic if dependent variable gets measured
    - With human raters if performance is not easily accessible
  - React to changes in the performance
- If the data keeps evolving, it is necessary to update your datasets and to retrain the ML model regularly

Advanced Topics for Supervised Learning: Learning objectives

- After this lecture, students shall be able to…
    - explain how data and algorithms can impact supervised machine learning
    - apply cross-validation and hyperparameter tuning to improve supervised machine learning models
    - decide how to deploy a supervised machine learning model

You can learn all this in an interactive online course!

https://app.datacamp.com/learn/courses/feature-engineering-for-machine-learning-in-python



Why generate features?

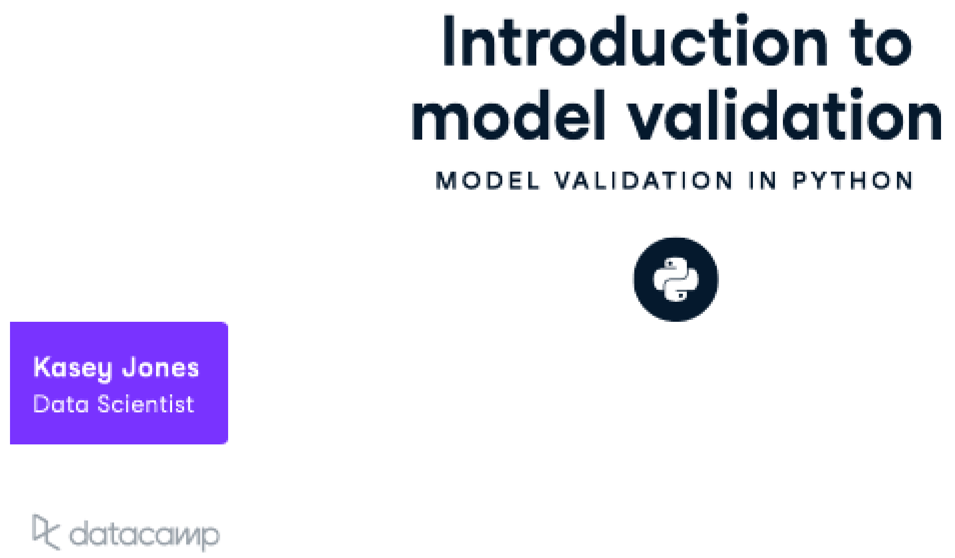
FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

datacamp

You can learn all this in an interactive online course!

https://app.datacamp.com/learn/courses/model-validation-in-python



Introduction to model validation

MODEL VALIDATION IN PYTHON

Kasey Jones
Data Scientist

datacamp