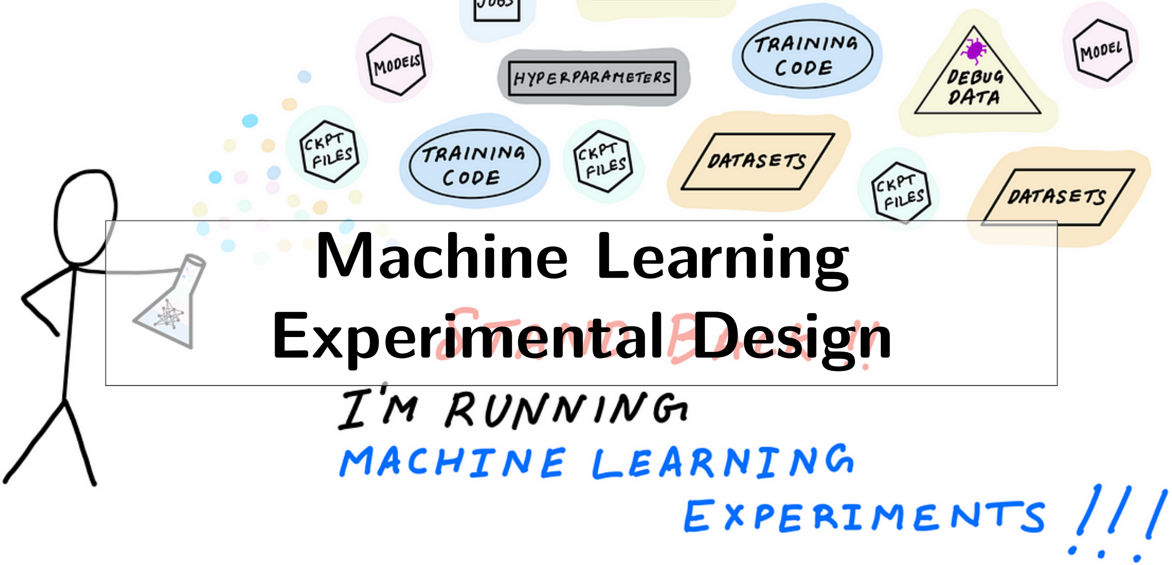
Machine Learning Experimental Design



Learning outcomes:

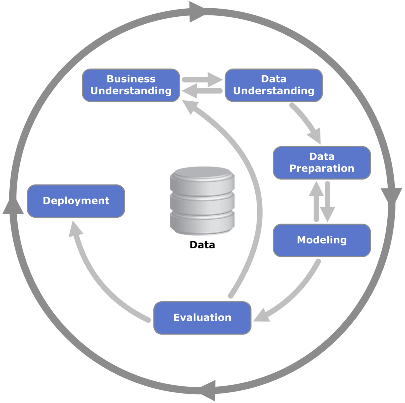


Figure: The CRISP-DM process.

- Explain the goals of ML experimental design;
- Define bias/variance tradeoff;
- Compare and contrast five ML experimental approaches;
- Compare and contrast two approaches to scaling;
- Compare and contrast four approaches to sample balance.

Where are we going?

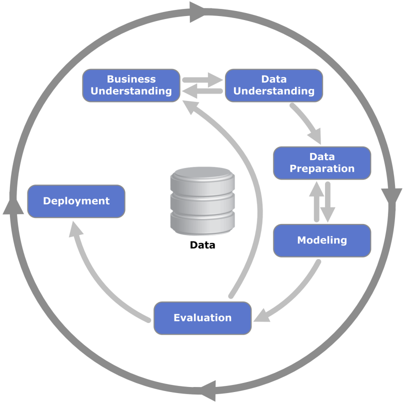


Figure: The CRISP-DM process.

We've arrived (finally!) at **machine learning**!

Machine Learning in the data science process

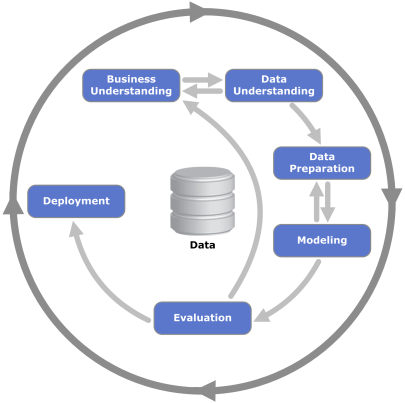


Figure: The CRISP-DM process.

We've arrived (finally!) at **machine learning**!

Here are all the steps we do first:

1. Gather data;
2. Clean/Preprocess data;
3. Exploratory data analysis;
4. Generate hypotheses;
5. Devise experimental structure, e.g. split test/train;
6. Engineer features;
7. Apply data transformations;
8. Sample balance;

Machine Learning in the data science process

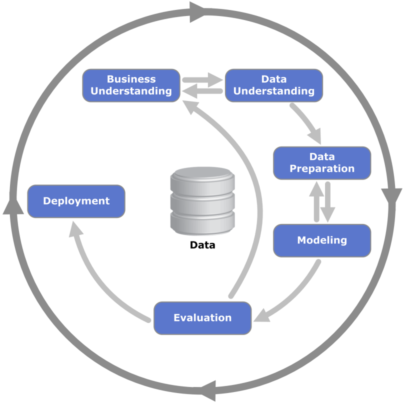


Figure: The CRISP-DM process.

We've arrived (finally!) at **machine learning**!

Here are all the steps we do first:

1. Gather data;
2. Clean/Preprocess data;
3. Exploratory data analysis;
4. Generate hypotheses;
5. Devise experimental structure, e.g. split test/train;
6. Engineer features;
7. Apply data transformations;
8. Sample balance;
9. Train/Evaluate model.

Machine Learning in the data science process

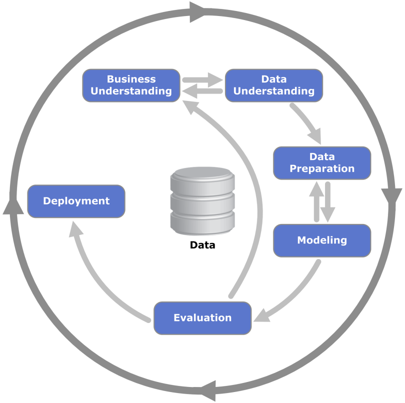


Figure: The CRISP-DM process.

We've arrived (finally!) at **machine learning**!

Here are all the steps we do first:

1. Gather data;
2. Clean/Preprocess data;
3. Exploratory data analysis;
4. Generate hypotheses;
5. **Devise experimental structure, e.g. split test/train;**
6. Engineer features;
7. **Apply data transformations;**
8. **Sample balance;**
9. Train/Evaluate model.

Three types of Machine Learning

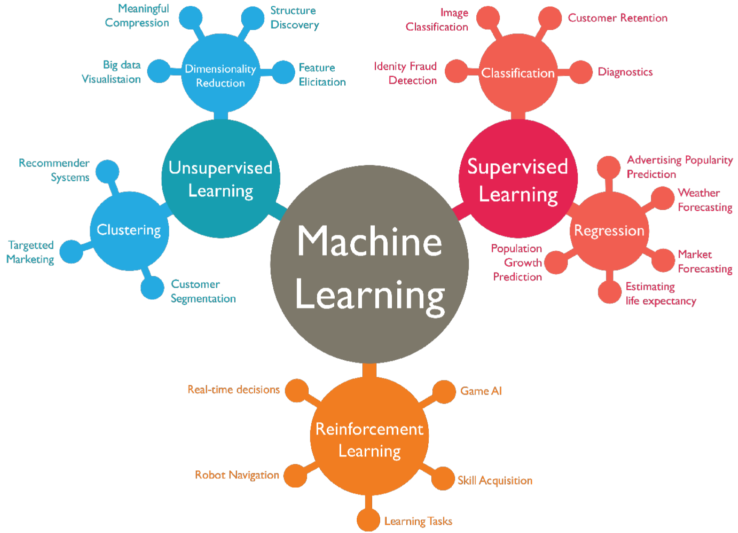


Figure: The source for this graphic.

Experimental Setup & Model Evaluation

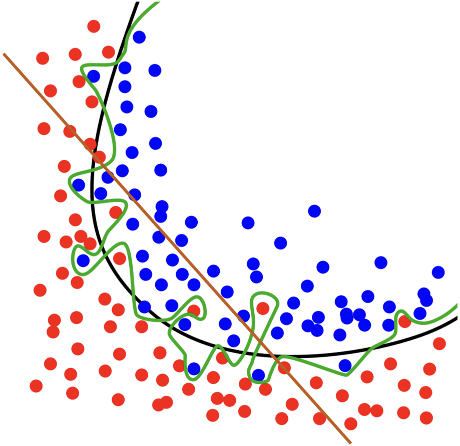
- In most real-world use cases, we want to:
  1. Train a model on data we have...
  2. ...to make predictions on data we haven't seen yet;

- Examples:
  - Predict what ads a user will click on given their past history;
  - Determine if a scan indicates cancer or not;
  - Determine if an email is spam or not.

- KEY THINGS:
  1. Make sure you don't overestimate model performance (no TARGET LEAKAGE!);
  2. Make sure the model doesn't use information to predict it wouldn't actually have in real life!

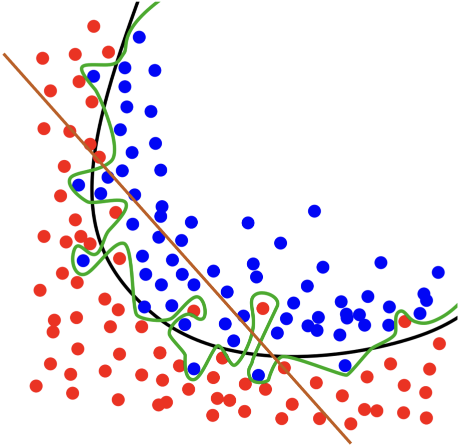
Possible scenarios

- Two reasons a model could be bad;
- The brown model poorly fits the data – why?
- The green model fits the data very well, but is still problematic – why?
- The Goldilocks model!



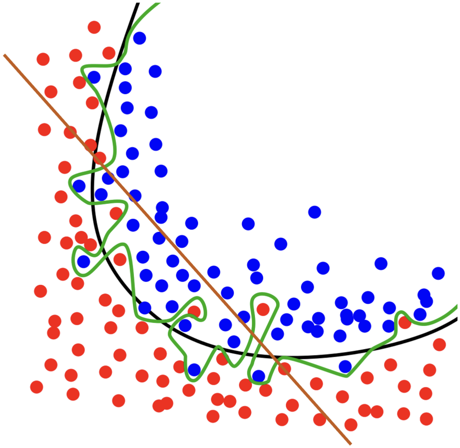
Possible scenarios

- Two reasons a model could be bad;
- The brown model poorly fits the data – why? Too simple – it is underfit
- The green model fits the data very well, but is still problematic – why? Too complex and too specific for the data – is overfit.
- The Goldilocks model!



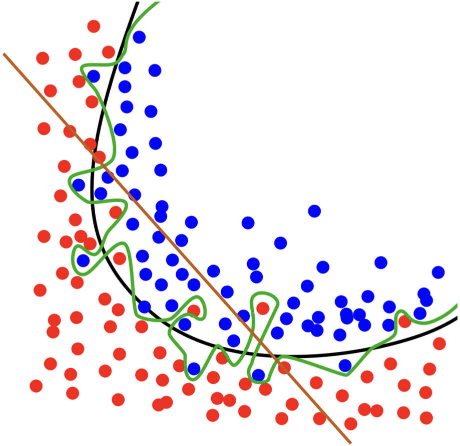
How do we know if a given trained model is underfit or overfit?

- Underfitting: the model does not perform well on the training set;
  - Training error is high;
  - Validation error is high;
- Overfitting: the model performs very well on the training set, but fails on the validation set;
  - Training error is low;
  - Validation error is high;
- **Goal**: find the best model that underfits the data.



How do we solve this?

- Underfitting is easy to solve – just use a more complex model!
- Overfitting is harder:
  - A model can "memorize" its training data...
  - ...and fail to find decision boundaries that work well on any other data;
  - ML experimental design is aimed at evaluating models to detect and avoid overfitting.

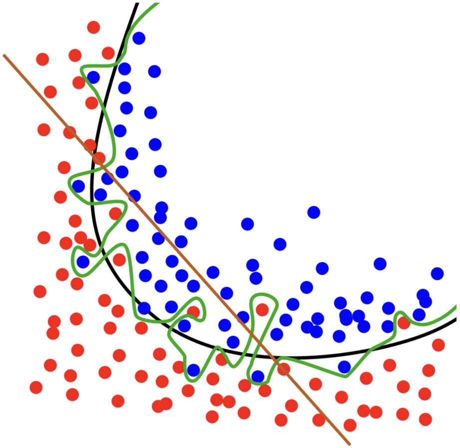


The Bias–Variance Tradeoff

- Given a target variable y and features x the average error of a model m is:

$$\begin{aligned} \text{Error} &= E[(y - m(x))^2] \\ &\vdots \text{ lots of math} \\ &= \text{Bias}^2 + \textit{Variance} + \varepsilon \end{aligned}$$

  - Bias = average difference between model predictions and target;
  - Variance = spread of the possible predictions from the model;
  - ε = irreducible error;



The Bias–Variance Tradeoff

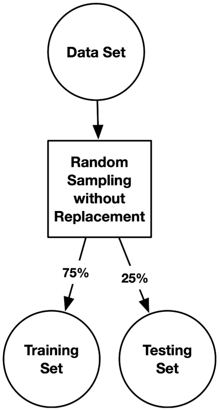
- A high-bias model fails to capture the complex patterns in the training data – it is underfit;
  - Error is introduced due to wrong assumptions such as approximating a complicated pattern in data with a simpler model;
  - An example is when we use a linear classification model on data that is not linearly separable;
- A high variance model will have predictions that change significantly if we retrain the model on a different training set;
  - Over-specialization to a particular training dataset generate large variance;
  - An example is when we use a very capable model on a relatively small data set.

Experimental Setup & Model Evaluation

- We want our experiments to tell us how a model will perform when applied to unseen data;
- Our experimental setup needs to simulate how we intend to use the model;
- We can simulate this situation with train-test splitting – we divide data into:
  - Training set – used for training the model;
  - Validation set – used for optimizing model hyper-parameters;
  - Testing set – used for evaluating the model's predictions;
- No observations are present in both sets;
- Since the testing set data is not used at all in training, it becomes an accurate way to evaluate a model's performance on unseen data.

Train-Test Split: Regression

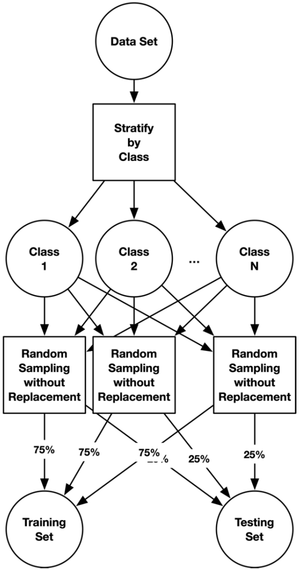
- Random sampling without replacement;
  - Simplest approach;
  - Appropriate for regression problems that are not time dependent;
- Procedure:
  - For each record, flip a coin to decide if the record goes into the training or testing set;
  - Frequently, 75% of records are used for training and 25% for testing;



Train-Test Split: Classification

- Need to ensures class ratios for training and testing sets match original data;
  - Slightly more complicated;
  - Appropriate/necessary for classification problems;

- Procedure:
  - Samples are divided by class labels;
  - Each class is divided into a training and testing set using random sampling without replacement;
  - All training sets are merged to produce a single training set;
  - All testing sets are merged to produce a single testing set.

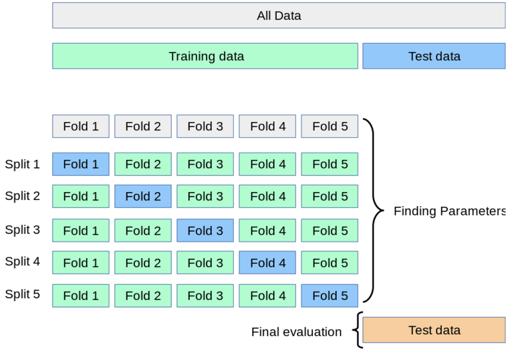


k-fold Cross Validation

- What if your dataset is really small? Might not be able to afford training/test/validation splits;

k-fold Cross Validation

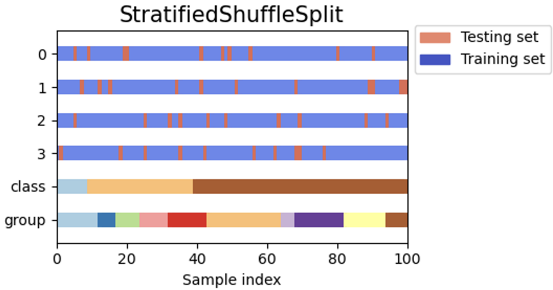
- What if your dataset is really small? Might not be able to afford training/test/validation splits;
- k-fold CV procedure:
  - Divide data into k parts called folds;
  - Assign each data point to one fold;
  - Loop through the folds, using each fold as the hold out set. You train on the remaining folds and evaluate the on the hold out set.



Stratified k-fold Cross Validation

- What if your dataset is really small? Might not be able to afford training/test/validation splits;

- Stratified k-fold CV procedure:
  - Divide data by class;
  - Divide each class into k folds;
  - Assign each data point to one fold;
  - Merge the folds together;
  - Loop through the folds, using each fold as the hold out set. You train on the remaining folds and evaluate the on the hold out set.



Bootstrapping

- What if you want to get a sense of the possible variation in model performance – a single test set won't let you do this;

- Bootstrap procedure option 1 – estimate the variation in models you could learn:
  1. Divide the data into train/test;
  2. Sample data points from your training data with replacement – this is the bootstrap training data;
  3. Train the model using the bootstrap training data and evaluate using the (fixed) test data;
  4. Save the metrics you've computed for model performance for the model learned from this bootstrap training data;
  5. Repeat 2–4 times.

Bootstrapping

- What if you want to get a sense of the possible variation in model performance – a single test set won't let you do this;

- Bootstrap procedure option 2 – estimate the variation in performance you could get for a fixed model:
  1. Divide the data into train/test;
  2. Train the model using the training data;
  3. Sample data points from your test data with replacement – this is the bootstrap test data;
  4. Evaluate the (fixed) model using the bootstrap test data;
  5. Save the metrics you've computed for model performance for this bootstrap test data;
  6. Repeat 3–5 times.

Bootstrapping

- What if you want to get a sense of the possible variation in model performance – a single test set won't let you do this;

- Bootstrap procedure option 3 – maximum variation:
  1. Sample n examples from your data with replacement – this is the bootstrap data;
  2. Divide the bootstrap data into a bootstrap train and a bootstrap test;
  3. Train the model using the bootstrap training data and evaluate using the bootstrap test data;
  4. Save the metrics you've computed for model performance in this bootstrap data
  5. Repeat 1–4 times.

Complicated situations...

- **Records may not be independent** – for example:
  - Time series data has before-after structure – random sampling to make train/test data will lead to the model learning from info it wouldn't actually have;
  - Individual rows in the data might be grouped, e.g. time series of records could belong to a single patient – random sampling to make train/test data will lead to target leakage;

- Example: Predict whether a patient will be diagnosed with Alzheimer's in the future based on (other) past diagnoses
  - Time dependent: need to use records BEFORE Alzheimer's diagnoses for training;
  - Class dependent: stratify by Alzheimer's vs not;
  - Patient dependent: need to keep records grouped by patient.

Complicated situations...

- **Records may not be independent** – for example:
  - Time series data has before-after structure – random sampling to make train/test data will lead to the model learning from info it wouldn't actually have;
  - Individual rows in the data might be grouped, e.g. time series of records could belong to a single patient – random sampling to make train/test data will lead to target leakage;

- Example: Predict whether a patient will be diagnosed with Alzheimer's in the future based on (other) past diagnoses
  - Time dependent: need to use records BEFORE Alzheimer's diagnoses for training;
  - Class dependent: stratify by Alzheimer's vs not;
  - Patient dependent: need to keep records grouped by patient.

- Solution:
  - For Alzheimer's positive patients, divide each patients' records into before and after diagnosis; discard records from AFTER diagnosis;
  - Within each class, assign patients using random sampling with replacement;
  - Merge training and testing sets.

Data scaling

- **Problem**: different features often have radically different scales (e.g. income in dollars vs time in years) – why is this a problem?

Data scaling

- **Problem**: different features often have radically different scales (e.g. income in dollars vs time in years) – why is this a problem?
  - Some ML models (KNN, SVM) use distance. If one feature has larger scale than other features, it will swamp the impact of other features;
  - Some ML models (logistic regression) are trained using an iterative algorithm. Features with radically different scale can cause convergence problems;

Data scaling

- **Problem**: different features often have radically different scales (e.g. income in dollars vs time in years) – why is this a problem?
  - Some ML models (KNN, SVM) use distance. If one feature has larger scale than other features, it will swamp the impact of other features;
  - Some ML models (logistic regression) are trained using an iterative algorithm. Features with radically different scale can cause convergence problems;
- **Goal**: transform the values of each feature so that they end up having similar range of possible values.
- Options:

Standardization	Normalization
$\frac{x-\bar{x}}{\sigma_x}$	$\frac{x-\min\{x\}}{\max\{x\}-\min\{x\}}$

- Normalization leads to a more predictable range but is more sensitive to outliers.

Sample balancing

- **Problem**: you could have n examples of class 1 and $m >> n$ of class 2 – why is this a problem?
  - It will skew some model performance metrics (upwards);
  - It will make learning to predict the smaller class much harder;
- Option 1 – downsample:
  - Sample n examples of the more numerous class without replacement;
  - Downside – you are throwing away information;
- Option 2 – upsample:
  - Sample $m - n$ examples of the less numerous class with replacement;
  - Downside – you are not actually adding any information;
- Option 3 – add synthetic data:
  - Sample points in the feature space and use KNN to label them;
  - Downside – these synthetic labels are noisy;
- Option 4 – use a model optimized for imbalanced data.

Should scaling and sample balancing (or feature engineering) happen before or after your training/test split?