CAS ML
Supervised Learning 1

Dr. Mark Rowan – Rowan Cognitive Data Science Solutions

Learning goals

- Understand ML context in data science
- Understand and practice gradient descent
- Understand linear regression
- Use loss functions to describe model performance
- Learn how to deal with missing data through imputation
- Learn how to transform data through scaling
- Understand need for train / test / validation set and cross-validation
- Learn how to perform feature selection
- Understand the bias/variance trade-off, diagnosing under-fitting and over-fitting
- Learn basics of regularisation in context of linear regression
- Understand model selection principles including hyperparameter search
- Hands-on: train and evaluate basic regression models in Python / sklearn

Introduction

- Grew up in UK; Switzerland since 2013
- > 10 years in AI and data science
  - Aerospace, telecoms, insurance, consumer goods
  - Start-ups in speech, medtech, transport
- Ph.D. Computational Neuroscience
- M.Sc. Natural Computation
- B.Sc. Artificial Intelligence & Computer Science

Please ask questions at any time!

ROWAN COGNITIVE
DATA SCIENCE SOLUTIONS

web: www.rowan-cognitive.com

mail: mark@rowan-cognitive.com

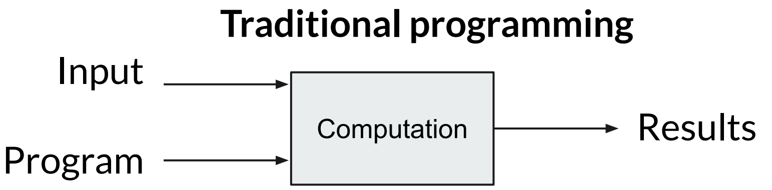
Introduction to Machine Learning

What is learning?

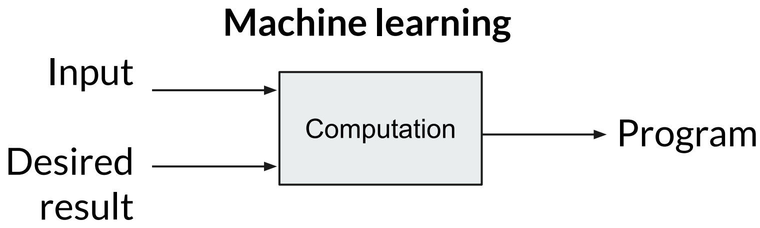
- “A change in the knowledge of a system that allows it to perform better on subsequent tasks”

- How is knowledge represented in various systems?
    - Brain
    - Computer
    - Stone tablets
    - …

Context: what is ML good for?



- Define rules and adjust to get desired result



- Discover patterns and derive rules automatically from data to get desired result
- Like human brain

Machine learning models

A model is simply a representation of knowledge

- Coin-toss game:

    What is the chance of next toss being H?

    What is the model?

- Assuming a model vs deriving a model
- Fitting data to a model vs fitting a model to data

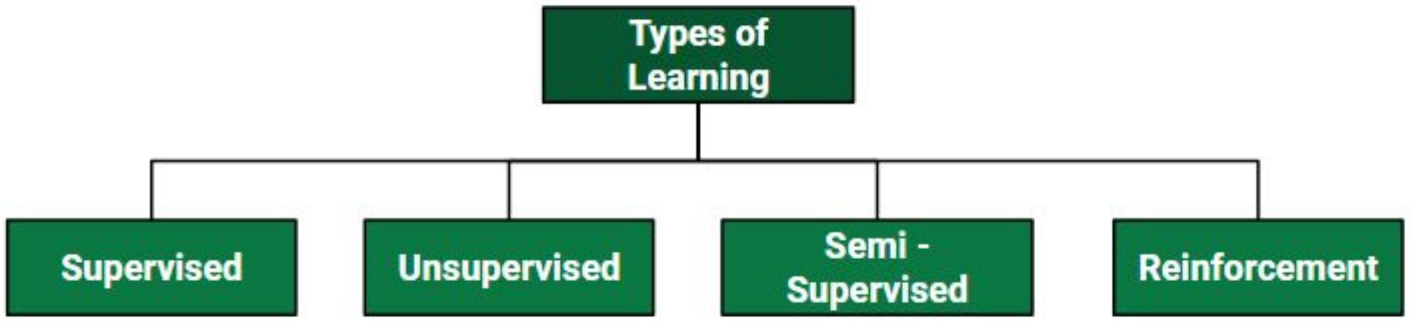
What can ML do?

- Recognize emotion in human speech
- Predict probability of live birth following IVF
- Detect candidates for pulsars in radio telescope data
- Detect “toxic” clauses in insurance contracts
- Predict likelihood of disease from diagnostic images
- Generate a sequence of actions to maximize return in trading
- Group customers into self-similar segments
- Detect and extract information from tables in documents
- Recognise and diagnose faults in a robot
- Optimize routes around a city
- Play games
- Drive vehicles
- Detect fraud

… anything a human can do? Better?

ML in action

Types of ML



Types of Learning

Supervised

Unsupervised

Semi - Supervised

Reinforcement

Types of ML: supervised

- Learning from a series of examples to minimize error on a task
- Requires a training set containing examples and labels (desired output)
- Labels are often the hardest thing to obtain!
- e.g. regression, classification

Types of ML: unsupervised

- Learning without feedback - finding structure directly in the data
- e.g. clustering, grouping

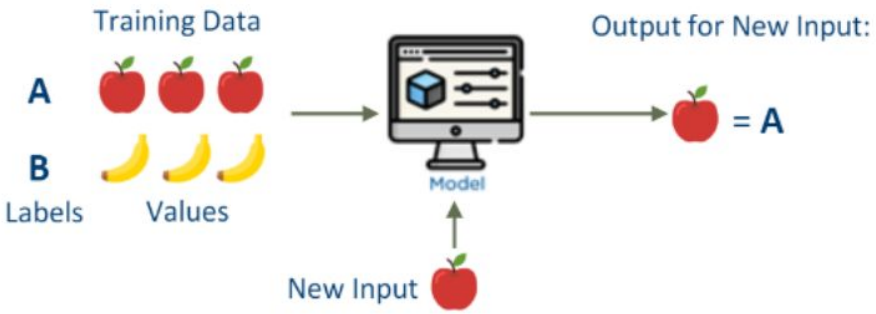
Types of ML: semi-supervised

- Using unsupervised learning to group data points
- Labelling the groups so they can be used as training examples in supervised learning

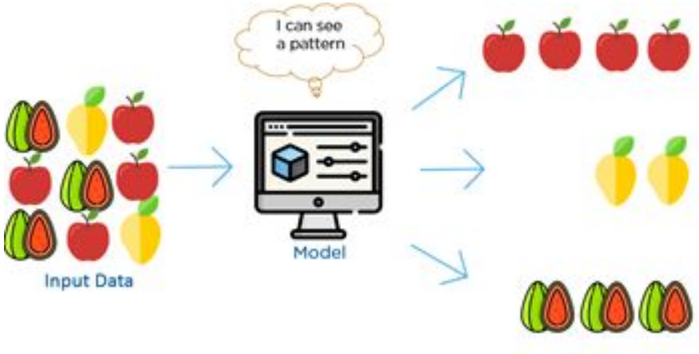
Types of ML: reinforcement learning

- Learning via indirect feedback after exploring many examples
  - i.e. learning with reward from mistakes
- Exploring a *state*, *action* space to generate a *policy*

Supervised vs unsupervised learning

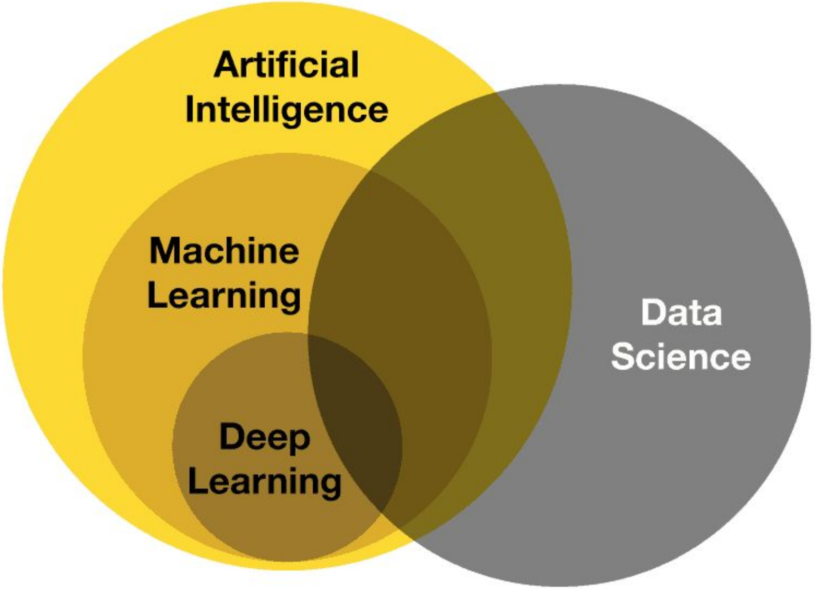


Supervised



Unsupervised

ML, data science, AI – where does it all fit?

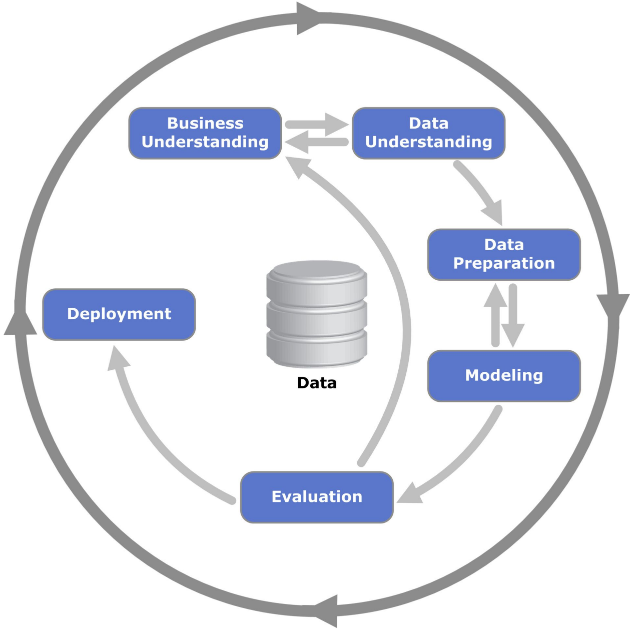


Data Science / ML project lifecycle

CRISP-DM

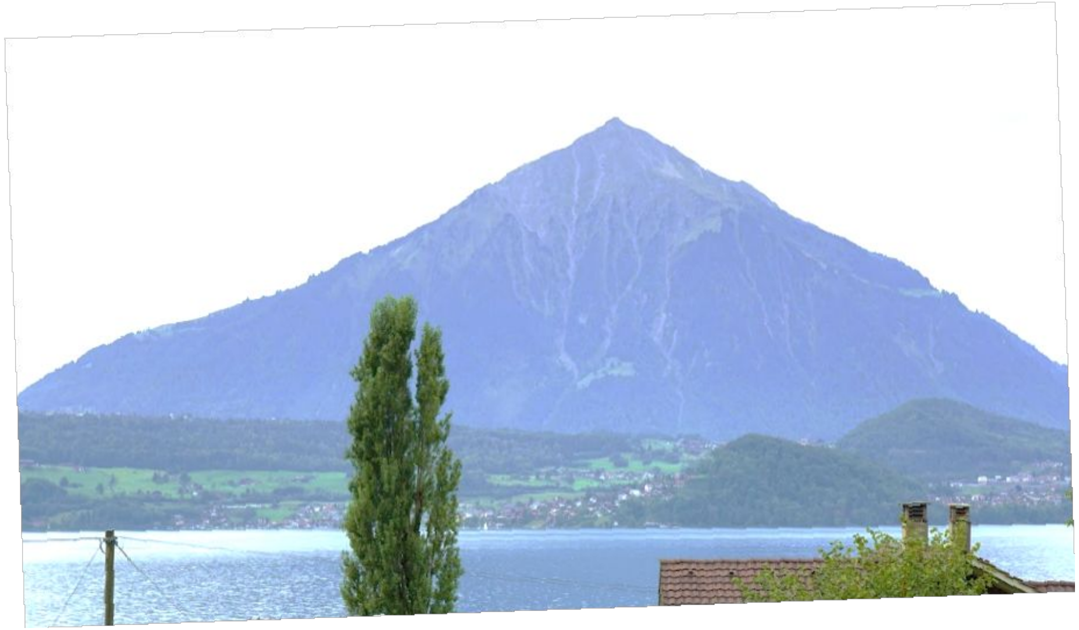
CRoss-Industry Standard Process for Data Mining

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment



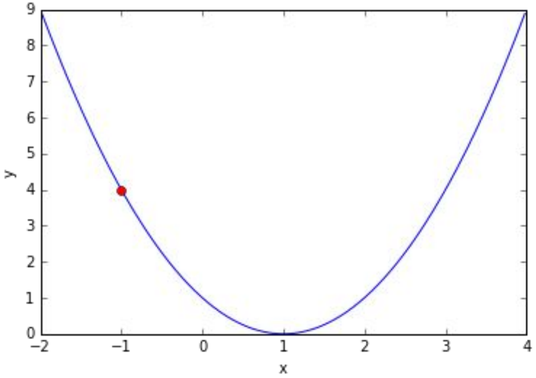
Gradient Descent

What is gradient descent (ascent)?



What is gradient descent (ascent)?

Iterative gradient descent algorithm



Find the *x* which gives lowest *y*

$$f(x) = (x - 1)^2$$

Gradient descent. Iterate until convergence (no more change):

$$x_1 = x_0 - \boxed{\eta} f'(x_0)$$

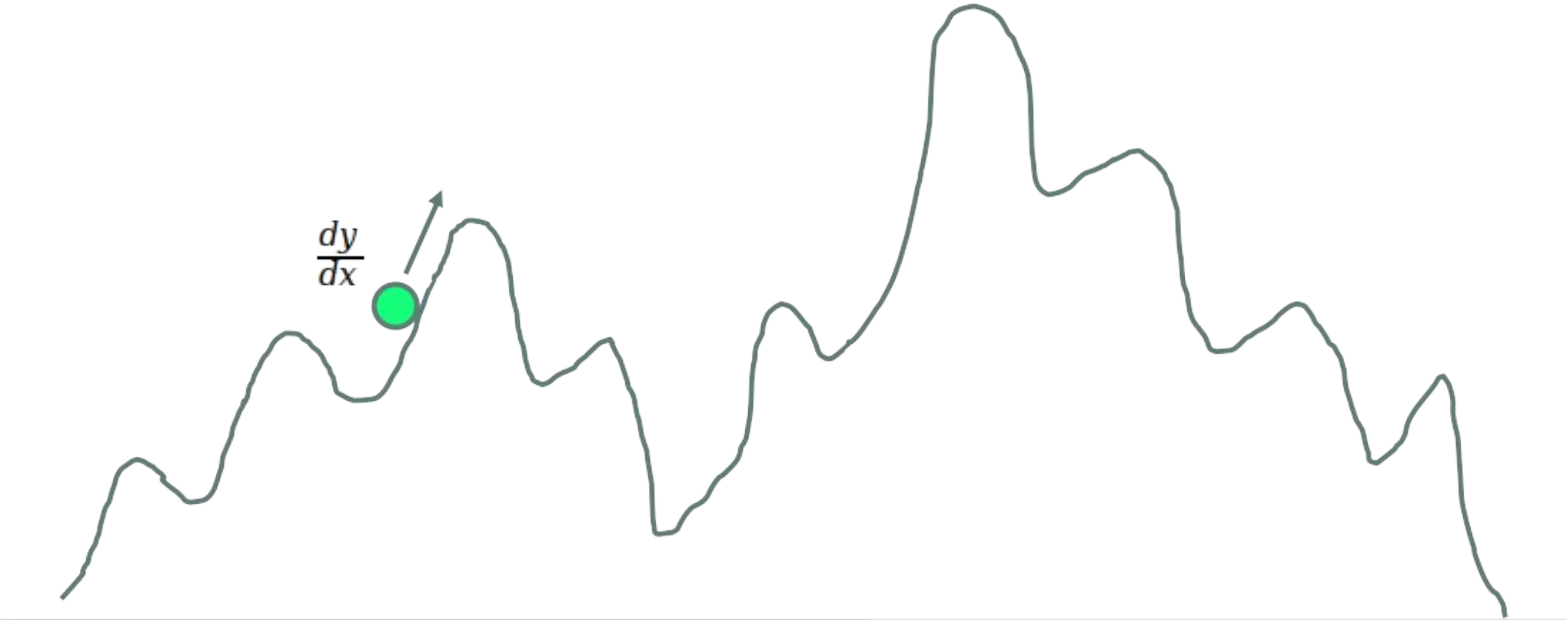
Learning rate

Derivative at point x gives the gradient:

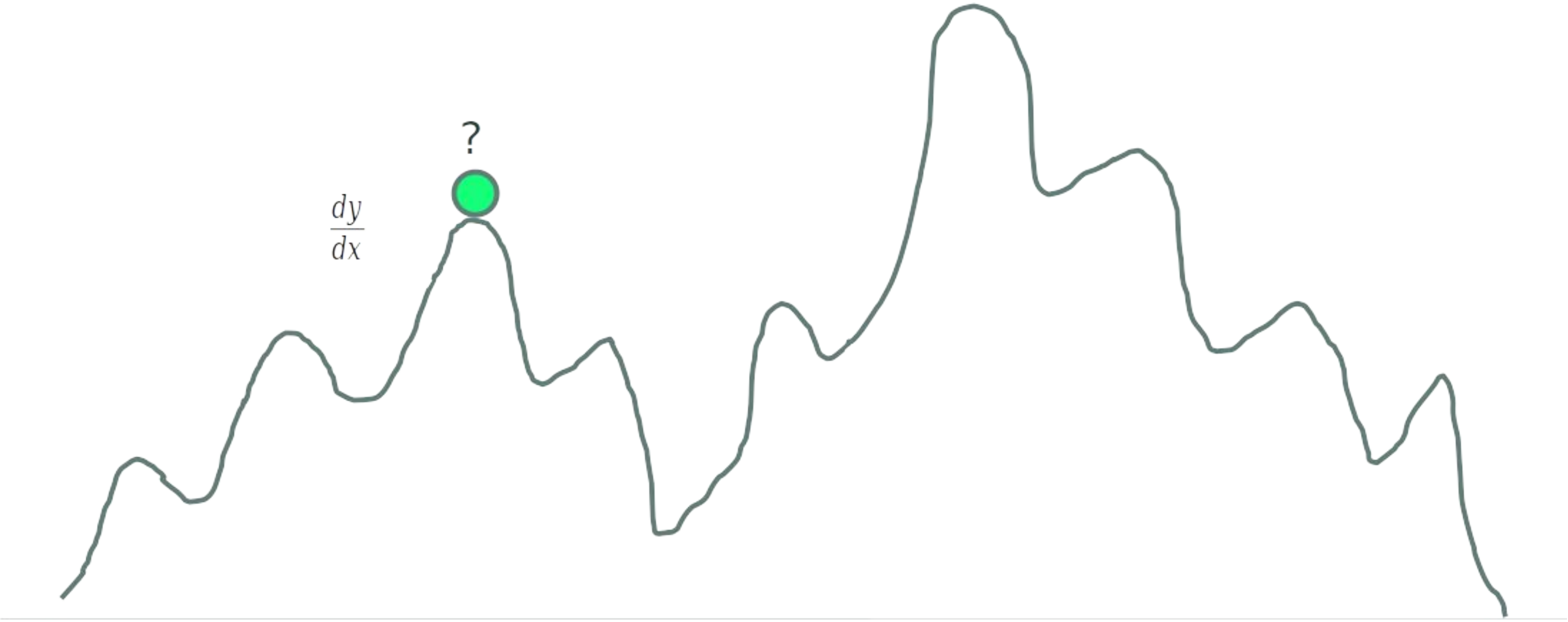
$$f'(x) = \frac{\delta y}{\delta x}$$

Route to the top is not always straightforward

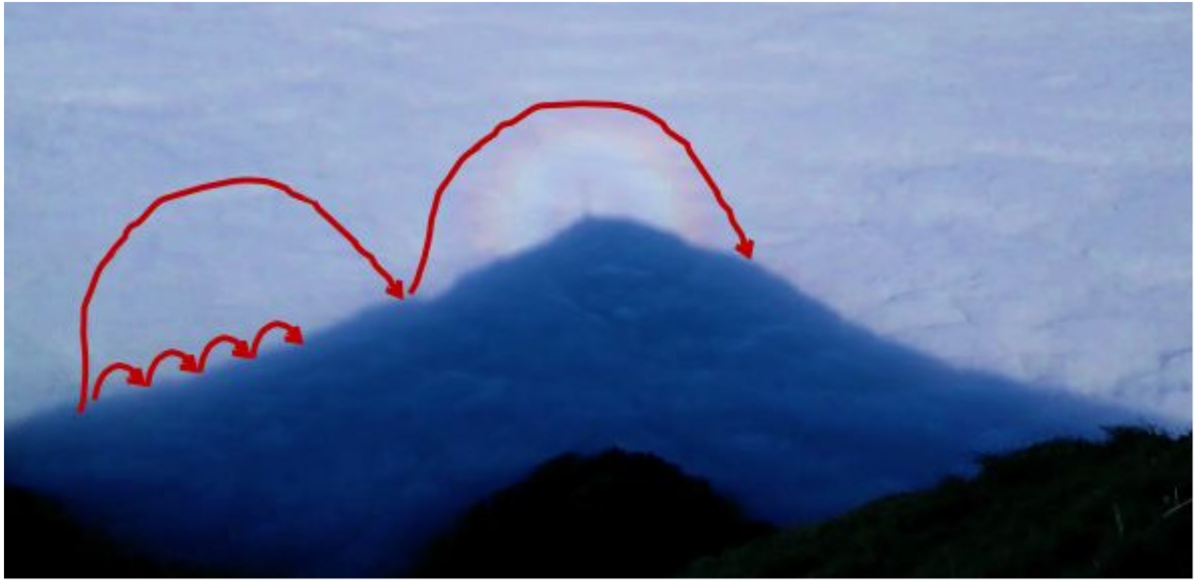
Gradient descent and local optima



Gradient descent and local optima



Gradient descent: learning rate



Also called:

- Step size
- Update rate

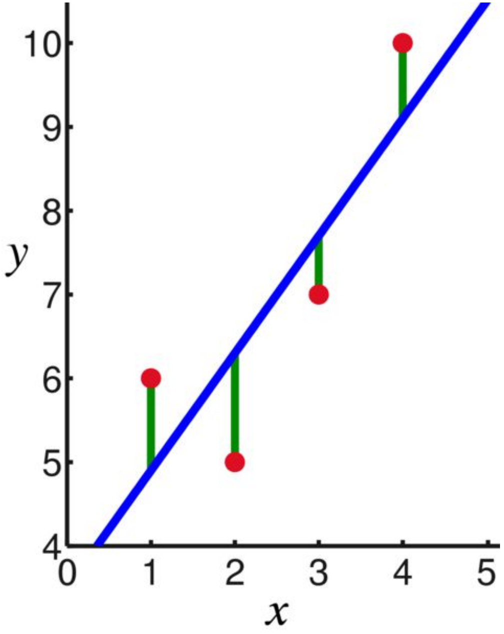
Take care if too large or too small!

Strategy: reduce learning rate over time (*annealing*)

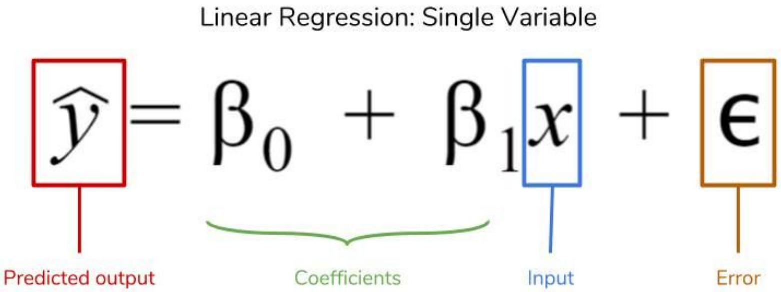
Linear regression

Fitting a line to points

- Find the best description of a relationship between *explanatory variable* (x) and *response variable* (y)
- e.g. number of rooms vs house price
- Assumes a *linear relationship* between variables
  - → Not always the reality!

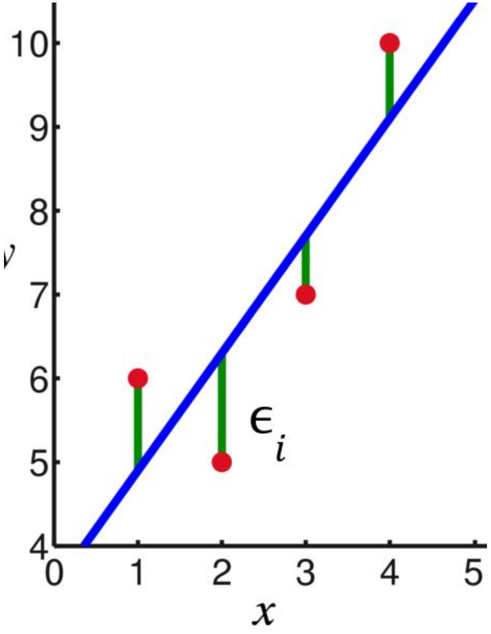


Fitting a line to points

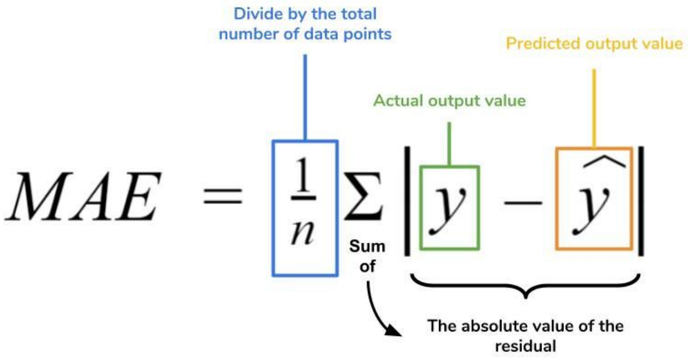


Linear Regression: Single Variable

$$\boxed{\hat{y}} = \underbrace{\beta_0 + \beta_1}_{\text{Coefficients}} \boxed{x} + \boxed{\epsilon}$$

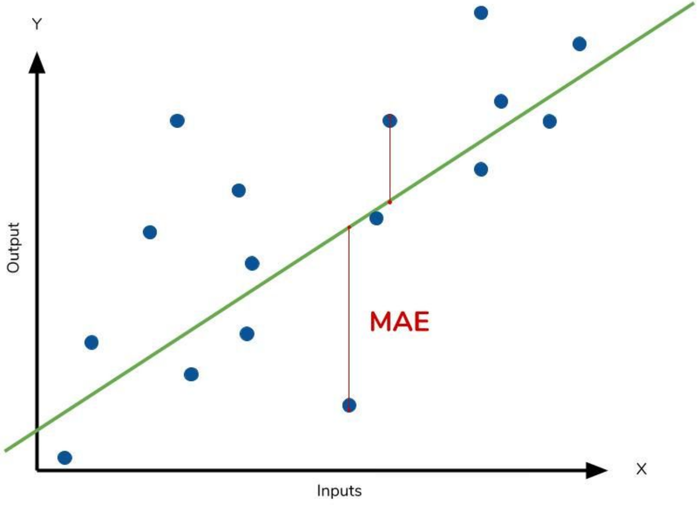


Measuring performance: mean absolute error



$$MAE = \boxed{\frac{1}{n}} \sum \underbrace{\left| \boxed{y} - \boxed{\widehat{y}} \right|}$$

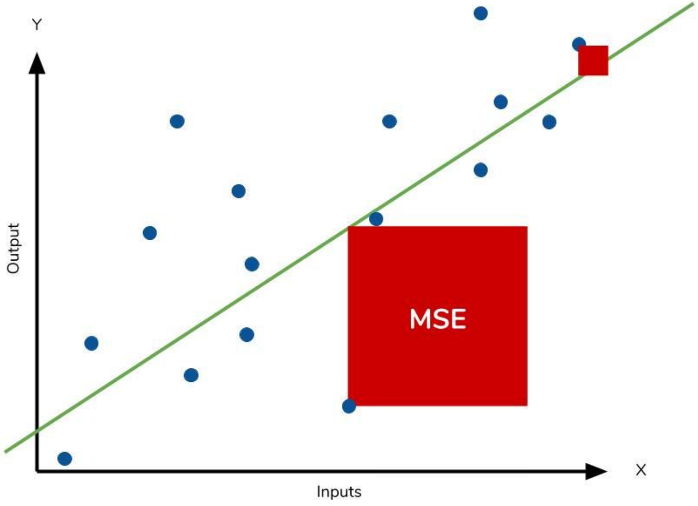
Divide by the total number of data points
Predicted output value
Actual output value
Sum of
The absolute value of the residual



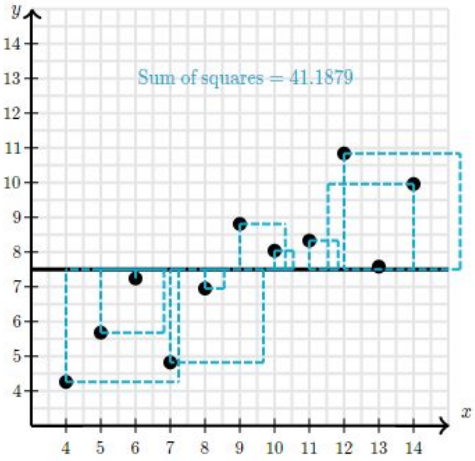
Measuring performance: mean squared error

$$MSE = \frac{1}{n}\Sigma\left(\underbrace{y - \widehat{y}}_{\text{The square of the difference between actual and predicted}}\right)^2$$

- Squaring the error penalises outliers / bad models greater than MAE
- You will also see RMSE (root MSE) and SSE (sum squared error)

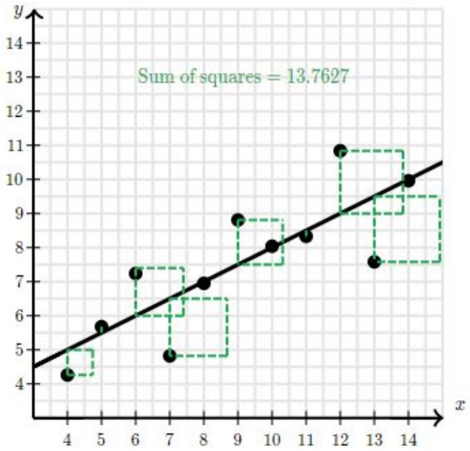


Baseline: dummy regressor



- Regression line is simply the mean of the response variable *y*
- Without regression, SSE = 41

Linear regressor



- Regression line is the best-fitting (least-squares) line
- SSE reduces to 13.7627
- But there is still error!

→ Impossible with this data to fit perfectly.

R-squared: coefficient of determination

What percentage of prediction error in response variable *y* is eliminated by the regression? R^2 takes values from 0→1

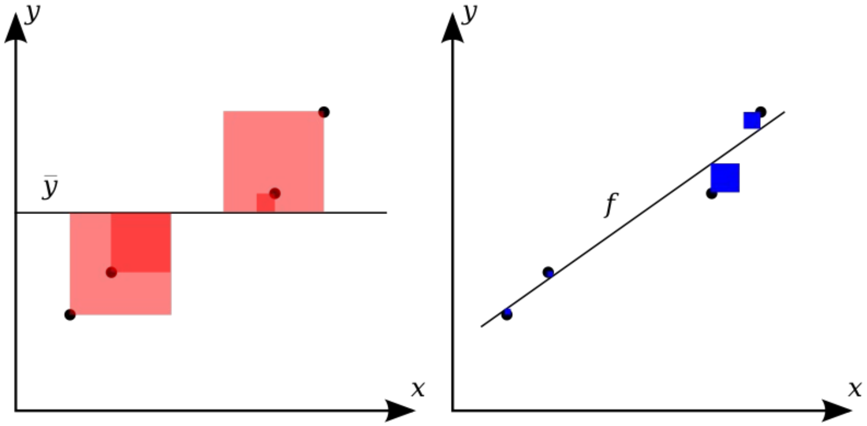
Sum of squares of residuals SS_res

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

Total sum of squares SS_tot

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

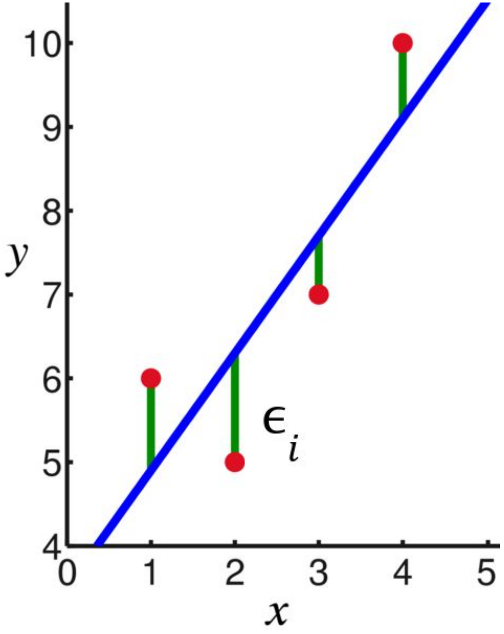


Using gradient descent to iterate

Linear Regression: Single Variable

$$\underset{\text{Predicted output}}{\boxed{\hat{y}}} = \underbrace{\beta_0 + \beta_1}_{\text{Coefficients}} \underset{\text{Input}}{\boxed{x}} + \underset{\text{Error}}{\boxed{\epsilon}}$$

- Initialise random values for the coefficients
- Calculate error metric (MAE, MSE, RMSE) over all *x*
- Adjust the coefficients and calculate change in error metric → *this gives us the gradient*
- Keep moving along the gradient until no further adjustment improves the error metric

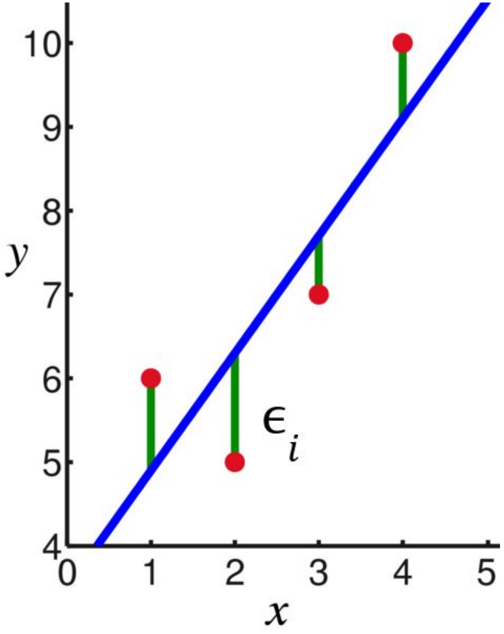


Multiple linear regression

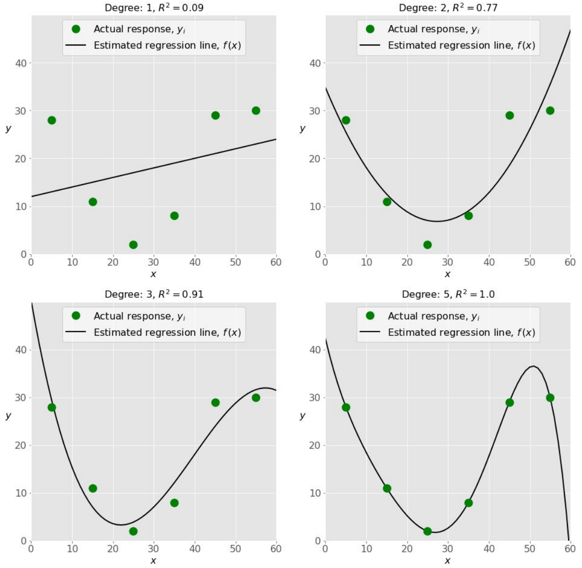
- We already saw that “single” linear regression actually contains two coefficients…
- For additional variables, we can just add more to the mix

Linear Regression: Multiple Variables

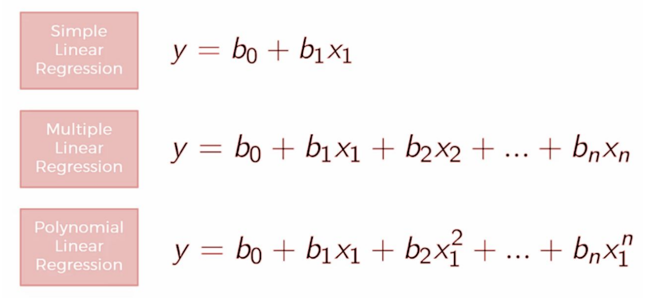
$$\boxed{\hat{y}} = \underbrace{\beta_0 + \beta_1}_{}\boxed{x_1} + \dots + \underbrace{\beta_p}_{}\boxed{x_p} + \boxed{\epsilon}$$



Polynomial linear regression



- Not all relationships are a straight line
- e.g. number of people infected vs time



Simple Linear Regression

$$y = b_0 + b_1 x_1$$

Multiple Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

Polynomial Linear Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

Data preparation

Data cleaning

Already covered!

But let’s do a quick recap…

Dealing with missing data

- Real-world data can have missing values
  - Human error
  - Sensor failure
  - Data corruption
  - Systematic reasons
- Many ML algorithms cannot cope with missing values

Strategy 1: Remove observations with missing values

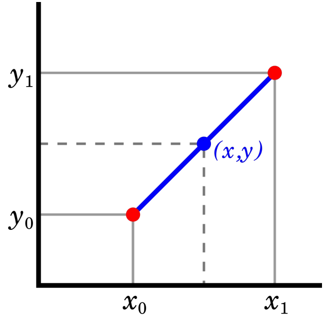
Strategy 2: Impute missing values

Simple imputation techniques

- Mean/median imputation: Replace missing values with the mean or median of the corresponding feature
    - Easy and fast
    - Doesn’t consider correlations between features
    - Poor results on encoded categorical features
- Modal/constant imputation: Replace missing values with the mode or a constant value of the corresponding feature
    - Or even just a placeholder constant e.g. -1
    - Works for categorical data
    - Ignores correlations
    - Can introduce bias

Advanced imputation techniques

- K-nearest neighbor (k-NN) imputation: Find the k-nearest neighbors to the data point with missing values and fill in the missing value with the average of the neighbors' values
  - Can be much more accurate than mean, median, mode
  - Computationally expensive
  - Sensitive to outliers
- Regression imputation: Use a regression model to predict missing values based on other features in the dataset
  - Take correlations into account
  - Computationally expensive
- Interpolation: estimate values from other observations within a set range



Feature engineering

The basics

Feature engineering describes the process of:

- selecting
- transforming
- creating features from raw data

to improve the performance of machine learning algorithms.

→ Can include scaling, feature extraction, feature selection

Data scaling

- Data contains features with varying magnitude

Example: relationship between fuel economy (l/100 km), engine size (cc) and top speed (km/h)

- How would the residuals look?

→ Bring all features to the same level of magnitude

Economy	Engine	Speed
10	2500	180
12	2700	200
8	1450	160
16	3600	240

Data scaling

$$z = \frac{x - \mu}{\sigma}$$

Standard scaling
Normalizes features with mean = 0 and std dev = 1.
Maintains the original distribution shape!

$$z = \frac{x - avg(x)}{max(x) - min(x)}$$

Mean normalization
Normalizes features with mean = 0 and all values between -1 and 1.

$$z = \frac{x - min(x)}{ma(x) - min(x)}$$

Min-Max scaling
Produces values in [0,1].

Useful when dealing with hard ranges, e.g. colour values from 0-255

$$z = \frac{x}{||x||}$$

Unit vector
Produces values in [0,1].

Boston Housing dataset

Documentation: https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

Data: http://lib.stat.cmu.edu/datasets/boston

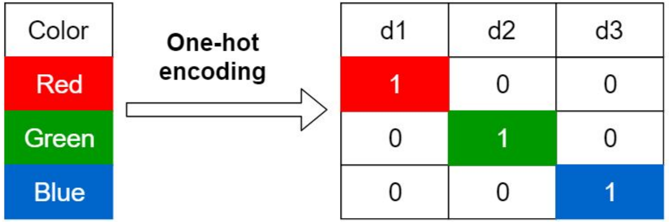
- A well-known dataset used for regression analysis, which includes 13 features related to housing prices in different neighborhoods in Boston
- The target variable for the regression problem is the median value of owner-occupied homes, which is included as a continuous variable in the dataset
- To set up the regression problem, the target variable should be separated from the other features and used to train and test the regression model

Representing categorical data (grouping, binning)

- In some cases, numerical data can be converted to categorical data to create more meaningful features
- For example, the feature "tax" in the Boston Housing dataset can be converted to a categorical variable by creating bins based on the range of tax values
- Caution: category labels should not be treated as ordinals!
    - e.g. Monday → 1, Tuesday → 2, Wednesday → 3, etc.
    - Doing regression on these labels would imply relationships such as “Wednesday > Monday”
    - Is “Monday < Sunday”? Does this even make sense?

Converting categorical to numerical: one-hot coding

- Categorical data can be encoded in different ways, such as label encoding or one-hot encoding
- One-hot encoding is a common method for encoding categorical data, which creates a binary vector for each category of the categorical feature
- For example, the feature "chas" in the Boston Housing dataset is a binary variable indicating whether the neighborhood is bounded by the Charles River or not. This feature can be one-hot encoded to create two new binary features, "chas_0" and "chas_1"



Color
Red
Green
Blue

One-hot encoding

d1	d2	d3
1	0	0
0	1	0
0	0	1

Hands-on: feature engineering
10 mins

Hands-on - Step 1: Load the Data

Task: predict the median housing prices in California, given various input features.

First, let's load the California Housing dataset using scikit-learn. The data is already preprocessed and available in scikit-learn's datasets module.

https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset

```python
import pandas as pd
from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset into a pandas dataframe
california = fetch_california_housing()
data = pd.DataFrame(california.data, columns=california.feature_names)
```

Hands-on - Step 2: Set the Target Variable

In supervised machine learning, we need to set the target variable or output variable that we want to predict.

In this case, our target variable is the median value of owner-occupied homes, which is available in the California Housing dataset as 'target'.

```
# Set the target variable
y = california.target
```

Hands-on - Step 3: Data Scaling

Before we start building our machine learning model, let's scale our data so that all features have a mean of 0 and a variance of 1.

This will help our model converge faster and improve its performance.

```
from sklearn.preprocessing import StandardScaler


# Scale the data

scaler = StandardScaler()

X = pd.DataFrame(scaler.fit_transform(data),

columns=california.feature_names)
```

Hands-on - Step 4: Feature Engineering

Feature engineering is the process of transforming raw data into features that better represent the underlying problem. This can help improve the performance of our machine learning models.

Here we are using the pandas cut function to bin the ‘median_income’ feature into categories. The cut function is used to segment and sort data values into bins. The result is a new column 'income_bin' that contains the bin numbers for each data value.

```
# Feature engineering on the 'median_income' feature
bins = [-1, 3, 6, 16]
labels = ['low', 'medium', 'high']
X['income_bin'] = pd.cut(data['MedInc'], bins=bins, labels=labels)
```

Hands-on - Step 5: One-Hot Encoding

Let's bin the ‘income_bin’ feature into categories using one-hot encoding via pandas’ **get_dummies**. One-hot encoding is a process of converting categorical variables into a set of binary variables.

This allows us to include categorical data in our machine learning models.

```
X = pd.get_dummies(X,
    columns=['income_bin'],
    prefix='median_income_binned')
```

Model assessment

Evaluating models

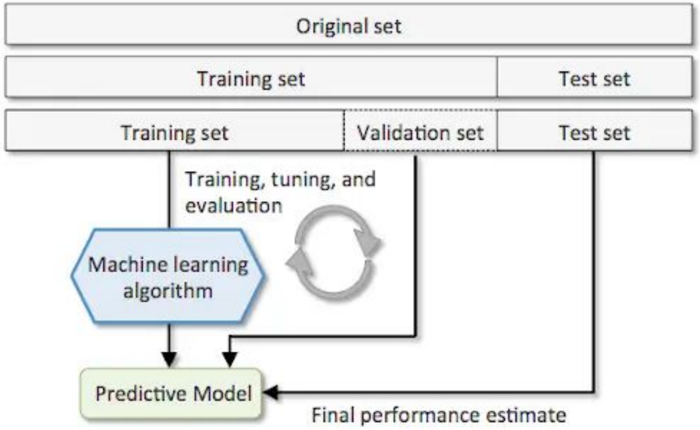
- Model assessment and selection is the process of evaluating and choosing the best machine learning model for a given problem
- Models can differ in the scope of:
  - chosen algorithm (e.g. linear regression, SVM)
  - hyperparameters (e.g. learning rate)
  - features used during training
  - ...many more!
- If we have trained multiple different models on our problem, how do we tell which one is “best”?

Training, test, and validation sets

- ML is all about learning from patterns in data. So far we worked with a single fixed model.
- Problem: how do we select the best model from multiple approaches?

Solution: Split data into 3 subsets **- training**, **validation** and **test**:

1. train models on **training** dataset
2. assess generalization performance on **validation** dataset
3. vary hyperperameters (e.g. learning rate)
4. repeat 1. and 2. until hyperparameters optimized for all models
5. select best model and train one last time on **training**+**validation** set
6. finally assess best model's performance on **test** dataset. Only once!



Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

Training, test, and validation sets

- Why not compare generalization performance on test data? Why use a validation dataset?
  - validation dataset is used multiple times to fine-tune the models (steps 1. and 2.).
  - this indirectly affects training, as we fine-tune models depending on performance on validation dataset
  - therefore the validation set is not really unseen data → performance more optimistic than on unseen data

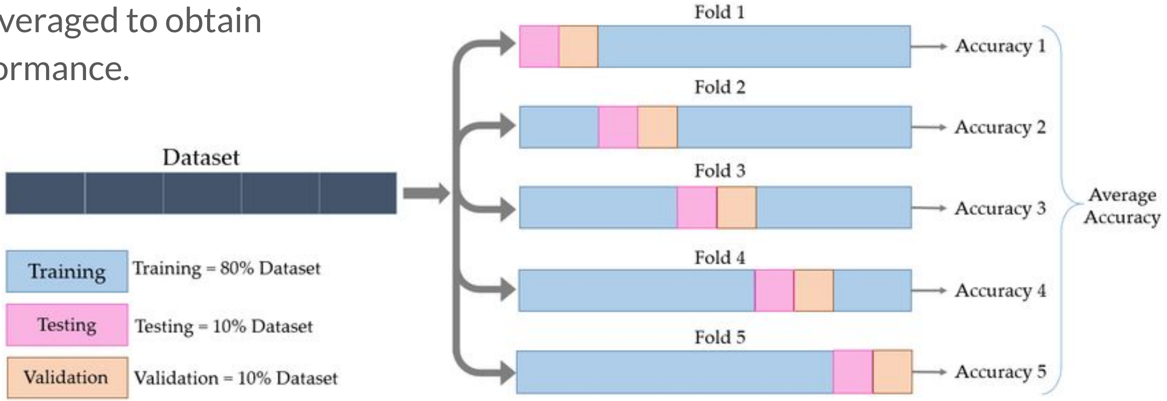
- Why retrain best model on training + validation dataset (step 4) ?
  - every model generally benefits from being trained on more data (“more data is better!”)
  - not a problem, since we do it only once before testing on the test data

Training, test, and validation sets

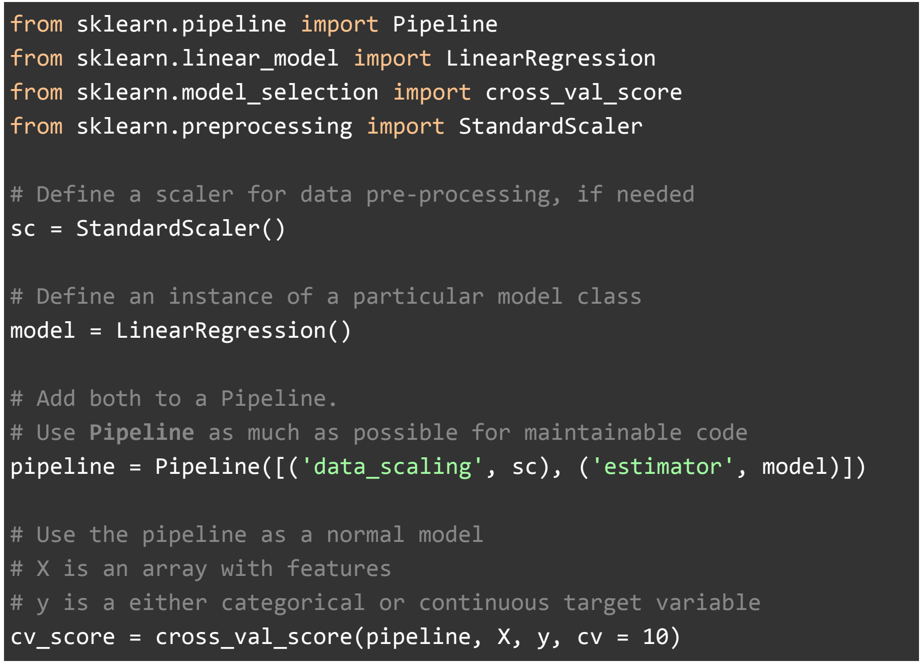
- Data should be *randomly split* into training, validation, and test sets.
- The ratio of data split depends on the size of the dataset and the problem.
- A common ratio is 60/20/20 for training, validation, and test sets, respectively.
  - Other common ratios are 80/10/10 and 70/20/10
  - Choice depends on data set size

Cross-validation

- Cross-validation is a method for estimating the performance of a model on unseen data.
- Cross-validation involves dividing the data into k folds, training the model on k-1 folds, and evaluating the model's performance on the remaining fold.
- This process is repeated k times, with each fold used once as the test set.
- The results from each fold are averaged to obtain an estimate of the model's performance.



Cross-validation code example



```python
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

# Define a scaler for data pre-processing, if needed
sc = StandardScaler()

# Define an instance of a particular model class
model = LinearRegression()

# Add both to a Pipeline.
# Use Pipeline as much as possible for maintainable code
pipeline = Pipeline([('data_scaling', sc), ('estimator', model)])

# Use the pipeline as a normal model
# X is an array with features
# y is a either categorical or continuous target variable
cv_score = cross_val_score(pipeline, X, y, cv = 10)
```

Types of cross-validation

- K-Fold Cross-Validation: The data is divided into k equal-sized folds.
- Stratified K-Fold Cross-Validation: The folds are created to ensure that each fold has approximately the same proportion of samples from each class.
- Leave-One-Out Cross-Validation: Each sample is used as the test set once, and the remaining samples are used to train the model.

Advantages of cross-validation

- Cross-validation provides a more accurate estimate of the model's performance on new data compared to using a single validation set.
- Cross-validation allows for better use of limited data by using all available samples for training and testing.
- But it is more computationally expensive! *(k-times more expensive, to be exact)*

Hands-on: model assessment
20 mins

Group work

1. Load the California Housing dataset into a pandas dataframe and Separate the feature data and target variable data into X and y.
2. Perform feature engineering as we explored before.
3. Split the data into training and testing sets using scikit-learn's `train_test_split` function.
4. Train a linear regression model on the training data.
5. Evaluate the model using mean squared error (MSE), mean absolute error (MAE), and R-squared (R2) scores on the testing data.
6. Use scikit-learn's `cross_val_score` function to perform 5-fold cross-validation on the model.
7. Calculate the mean and standard deviation of the MSE, MAE, and R2 scores.

Hints to explore further

1. Experiment with different feature engineering techniques, such as adding polynomial features or interaction terms, and see if it improves the model performance. You can use scikit-learn's PolynomialFeatures for this.
2. Try different scaling techniques, such as standardization or min-max scaling, and see how they affect the model performance.
3. Try different evaluation metrics, such as root mean squared error (RMSE) or mean absolute percentage error (MAPE), and see which one is the most useful for this dataset.
4. Try different regression models, such as Ridge regression or Lasso regression, and compare performance with the linear regression model.

Feature selection

Curse of dimensionality

- The curse of dimensionality is a term used to describe the problems that arise when working with high-dimensional data.
- As the number of features increases, the data becomes more sparse and the distance between data points becomes larger, making it more difficult to build accurate models.
- Feature selection is the process of identifying and selecting the most important features in a dataset, which can help to reduce the impact of the curse of dimensionality and improve model performance.

Statistical tests for feature selection

Several statistical tests can be used for feature selection. Some common examples:

1. **Pearson's correlation coefficient**: A measure of the linear relationship between two variables. Features with a high correlation to the target variable may be selected.
2. **Chi-squared test**: A test of independence between two categorical variables. Features that are independent of the target variable may be removed.
3. **ANOVA F-test**: A test of the difference in means between two or more groups. Features with a significant difference in means between groups may be selected.
4. **Mutual information**: A measure of the dependence between two variables. Features with a high mutual information with the target variable may be selected.
5. **t-test**: A test of the difference in means between two groups. Features with a significant difference in means between groups may be selected.

Other methods

There are three main types of feature selection: filter methods, wrapper methods, embedded methods

1. **Filter methods** use statistical tests or correlation matrix to rank the features based on their relevance to the target variable.
2. **Wrapper methods** use a model to evaluate the performance of different subsets of features. For example, post-hoc analysis of feature importance.
3. **Embedded methods** incorporate feature selection into the model building process, such as in Lasso or Ridge regression → See later!

Effect on model performance

- Feature selection can help to improve model performance by reducing the impact of irrelevant or redundant features.
- Removing irrelevant features can help to reduce the complexity of the model, which can improve its generalization performance.
- Removing redundant features can help to reduce over-fitting, which can also improve the generalization performance of the model.

Effect on model interpretability

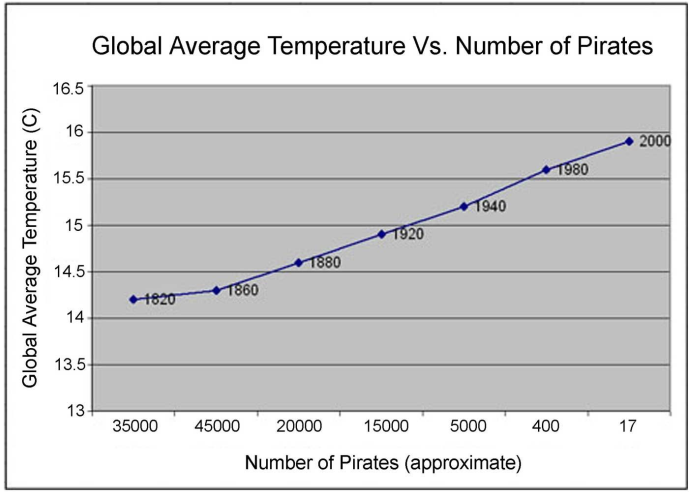
- Feature selection can also improve the interpretability of the model by focusing on the most important features.
- By selecting a smaller subset of features, the model becomes easier to understand and explain.
- This can be particularly important in applications where the model needs to be transparent and explainable, such as in healthcare or finance.

Example

What does the churn rate of customers of a telecommunications company have to do with the percentage of people in a location who vote for the SVP?

Global Warming and Pirates

STOP GLOBAL WARMING: BECOME A PIRATE



WWW.VENGANZA.ORG

…is this a pie-rate chart?

Hands-on: feature selection
20 mins

Group work

Extend your code to select features based on a correlation matrix

- Tip: pandas provides **`X.corr()`**
- Select features with correlation less than a threshold
- Explore the effect of changing selection thresholds, and other feature selection methods, on your model performance
- Visualise your correlation matrix using *seaborn's* **`heatmap`**:

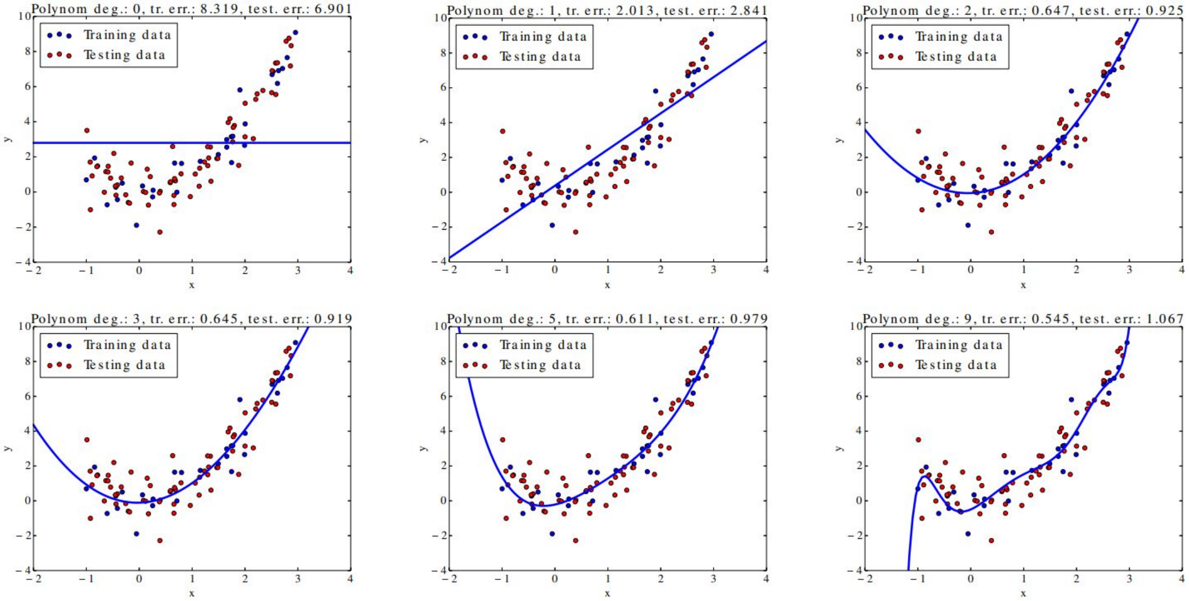
```
import matplotlib.pyplot as plt
import seaborn as sns

corr_matrix = X.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap=plt.cm.Reds)
plt.show()
```

Discussion

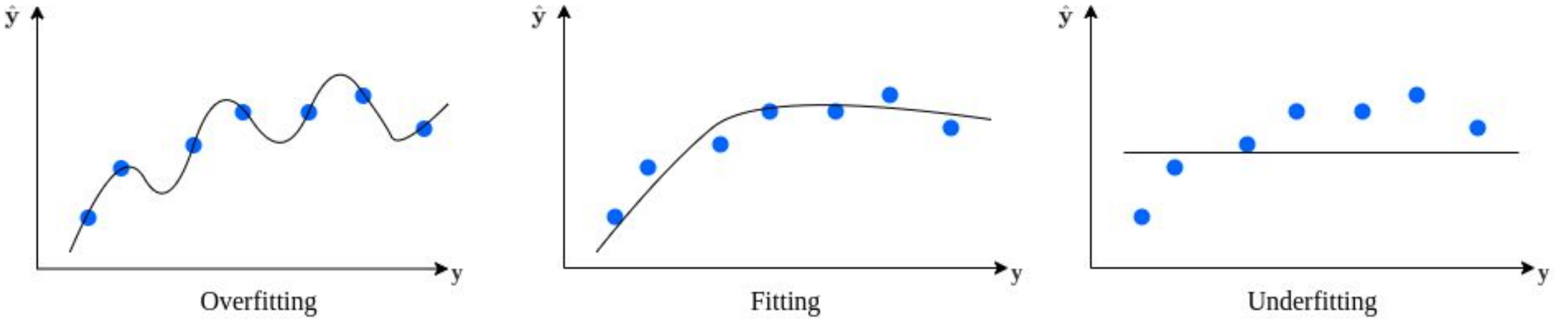
- What threshold did you use for selecting the features?
- How many features were selected and how do they relate to the target variable?
- Did you notice any interesting patterns or correlations among the selected features?
- How do the selected features compare to the original set of features?

Bias/variance trade-off

Best fit on training data doesn't mean best model



Over-fitting and under-fitting



Overfitting

Fitting

Underfitting

Bias and Variance

Bias refers to the difference between the predicted values of the model and the true values.

- A model with high bias is underfitting the data, meaning it is not able to capture the underlying relationships between the features and the target variable.

Variance refers to the variability of the model's predictions for different training sets.

- A model with high variance is overfitting the data, meaning it is too closely fitting the noise in the training data and not generalizing well to new data.

Bias and Variance

The bias-variance trade-off is a key concept in machine learning that describes the relationship between model complexity and prediction error.

In general:

- more complex models have **lower bias** but **higher variance**
- simpler models have **higher bias** but **lower variance**

Which model is better: simple or complex?

Finding the optimal model

To find the optimal model, we need to balance bias and variance to minimize the total error. This can be done by adjusting the complexity of the model.

- Model too simple: add more features or increase the degree of a polynomial to reduce the bias
- Model too complex: reduce the number of features or use regularization to reduce the variance

Cross-validation can be used to estimate the generalization error of the model and help to select the optimal complexity. By finding the sweet spot between bias and variance, we can build a model that generalizes well to new data and makes accurate predictions.

Examples of high bias (under-fitting)

- A simple linear regression model that is unable to capture the non-linear relationships between the features and the target variable in a dataset of house prices. The model has high bias and is not able to make accurate predictions.
- Examples:
  - A spam email filter that is too simple and only looks at a few features, such as the presence of certain words. The filter has high bias and is not able to accurately distinguish between spam and non-spam emails.
  - A linear regression applied to a problem which has a non-linear relationship between the predictors and the target variable (see tomorrow!)

Examples of high variance (over-fitting)

- Over-fitting in Financial Models:
  - Financial models used for investment and risk management
  - Regression models trained on historical financial data to predict future trends
  - Over-fitting occurs when models are too complex and fit historical data too closely
  - Can lead to inaccurate predictions and financial losses
- Over-fitting in Healthcare Models:
  - Regression models used in healthcare to predict patient outcomes based on medical features
  - Over-fitting occurs when models are too complex and fit the noise in the data
  - Can lead to inaccurate predictions and harmful medical decisions

Diagnosing under-fitting

- Model performs poorly on both training and test data
- Small difference between training and test error
- High bias in model coefficients or predictions
- Model is too simple or has too few features

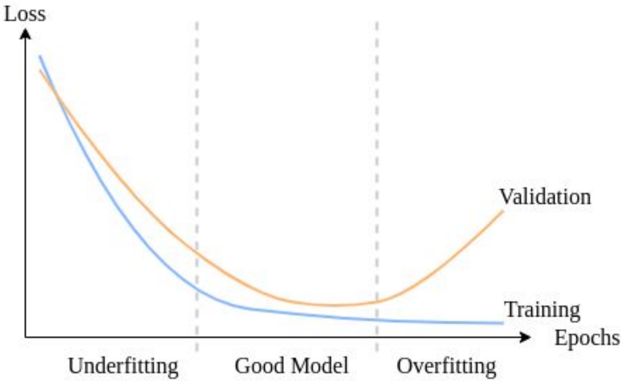
Diagnosing over-fitting

- Model performs well on training data, but poorly on test data
- Large difference between training and test error
- High variance in model coefficients or predictions
- Model is too complex or has too many features

Diagnostic techniques

Techniques for Diagnosing Over-fitting and Under-fitting:

- Visualizing the training and test error over time
- Using cross-validation to estimate model performance on new data
- Examining size and distribution of model coefficients
- Analyzing the residuals of the model predictions



Regularisation

Over-fitting and regularization

- Regularization is a technique to prevent over-fitting in machine learning models.
- It adds a penalty term to the cost function (e.g. MSE) which encourages the model to have smaller weights.
- By controlling the strength of the penalty with a parameter (λ), we can tune the trade-off between model complexity and overfitting.
- The penalty term is quadratic in the weights.
- This has the effect of distributing the weights more evenly and implicitly exploiting underlying dependencies of the set of features.
- Noisy features are penalized more, reducing their impact on the model and taming potential overfitting effects.

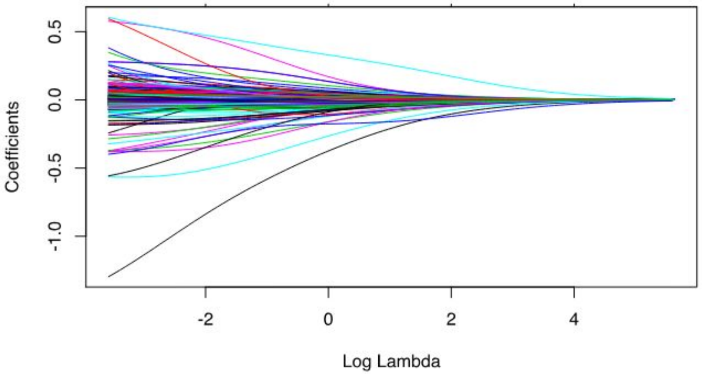
Ridge regression

Also known as L2, coefficient constraint

- Adds a penalty term proportional to the square of the weights.
- Simplifies a model
- Useful when there are many correlated features.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Ridge regression coefficients as strength parameter increases from 0 → infinity.



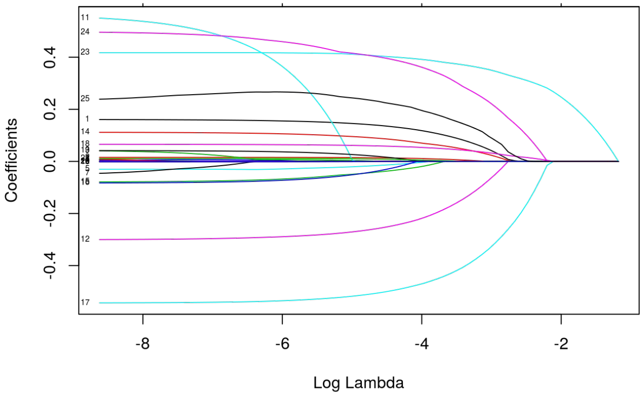
Lasso regression

Also known as L1, feature selection

- Lasso regression adds a penalty term proportional to the absolute value of the weights.
- Useful when there are many features and some of them are irrelevant.
- Has the effect of disabling features (coefficients go to 0)

$$\sum_{i=1}^{n} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Lasso regression coefficients as strength parameter increases from 0 → infinity.



Effect of lasso and ridge regularization

- Over-fitting occurs when a model captures noise instead of underlying patterns due to excessive complexity.
- Lasso and Ridge regression add a penalty term to discourage large weights, reducing model complexity.
- This reduction helps identify the most important features, improving generalization performance.
- They also address multicollinearity by reducing feature weights, stabilizing and making the model less sensitive to data changes.

ElasticNet

L1 + L2 - combination of both approaches

- ElasticNet adds a penalty term that is a combination of the L1 and L2 norms of the weights.
- Useful when there are (too) many features and some of them are correlated.
- Balances the strengths of Lasso and Ridge regression, making it more robust than either method alone.

Model selection

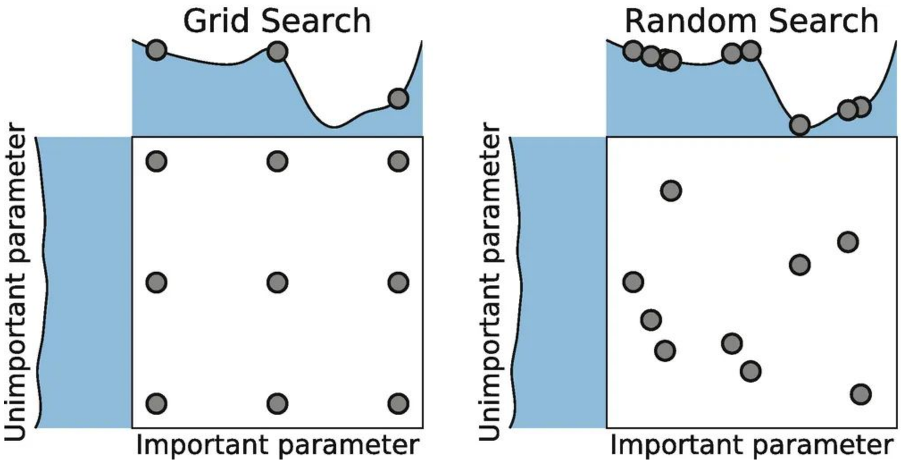
Hyperparameter tuning

- Hyperparameters are parameters that are set before training the model.
- Examples of hyperparameters include
    - learning rate
    - lasso / ridge regression strength (*alpha*)
    - anything else algorithm-specific
- Hyperparameters can significantly affect the model's performance, and tuning them is an essential part of the model assessment and selection process.

Hyperparameter search

How do we know the best hyperparameter values?
→ Search for them!

- Grid search: specify a range of values for each hyperparameter and try all possible combinations.
  - Can be time-consuming for large datasets and many hyperparameters.
- Random search: specify a range of values for each hyperparameter and try a random subset of the possible combinations.
  - Often more efficient than grid search and can produce similar results.



Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305 (2012)

Final model training

- After selecting the best model and tuning hyperparameters using cross-validation, the final model should be trained on all available data.
- This ensures that the model is trained on as much data as possible and can capture all relevant patterns in the data.

Hands on: model selection

Group work

Extend your code to explore different model types and select the best performing on your dataset

- linear regression
- lasso, ridge, ElasticNet regression
- different hyperparameter values
- use Grid and Random search

Remember you will need to create a test (hold-out) set to compare your models on unseen data

Tips

- Use **`Pipeline`** from sklearn.pipeline to chain together feature engineering and models
- Define parameter grids for your Grid and Random searches
- Perform search including cross validation using **`GridSearchCV`** and **`RandomizedSearchCV`**
- Fix the random state of your code using **`random_state`** parameters when appropriate!

```
linear_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    (lasso, Lasso())
])
```

```
grid_lasso_params = {
    'lasso__alpha': [0.1, 1, 10],
    'lasso__max_iter': [100, 1000, 10000]
}
```

```
grid_search_lasso = GridSearchCV(
    lasso_pipeline, grid_lasso_params,
    cv=5, n_jobs=-1)
```

```
random_search_lasso = RandomizedSearchCV(
    lasso_pipeline, rand_lasso_params,
    cv=5, n_jobs=-1, random_state=42)
```

```
best_model_lasso = grid_search_lasso.best_estimator_
best_params_lasso = grid_search_lasso.best_params_
```

- linear, lasso, ridge, ElasticNet regression
- different hyperparameter values
- use Grid and Random search

Example

Wait until the end of the class, we will go through it together!

https://colab.research.google.com/drive/1rnIz4wKcJ_b3QEp1U4Tdg6pZkeeZtEaU?usp=sharing

Another very comprehensive walk-through:
https://gist.github.com/machinelearning-blog/76b50b18c7db3408646cc8d18c50c20b