Practical Machine Learning

An introductory course
in Python

Course Agenda (I)

- Starting with data:
  - Inspecting data; Types of Data
  - A crash course in Python
  - Using Pandas and Numpy
  - Visualize Data in Python using Matplotlib Pyplot
  - Imputation: filling missing data; Normalization
  - Introduction to feature engineering
  - Examples with the Titanic and MPG models

Course Agenda (II)

- Modeling and Prediction
  - Basics of Modeling; General Workflow; Supervised and Unsupervised Learning
  - Short presentation of Machine Learning techniques
  - Classification problems
    - Example of classification in python and scikit
  - Regression problems
    - Examples of regression in python
    - A framework for regression analysis in python

Course Agenda (III)

- Model evaluation and optimization
  - Overfitting and model optimism
  - Cross-validation techniques
  - Evaluation of classification models
  - Evaluation of regression models
- Feature Engineering
  - Basic feature engineering on time and text
  - Feature selection
  - Feature engineering on time-series data
- Reinforcement Learning
  - Neural Networks, Keras, TensorFlow

Labs of this course on GitHub

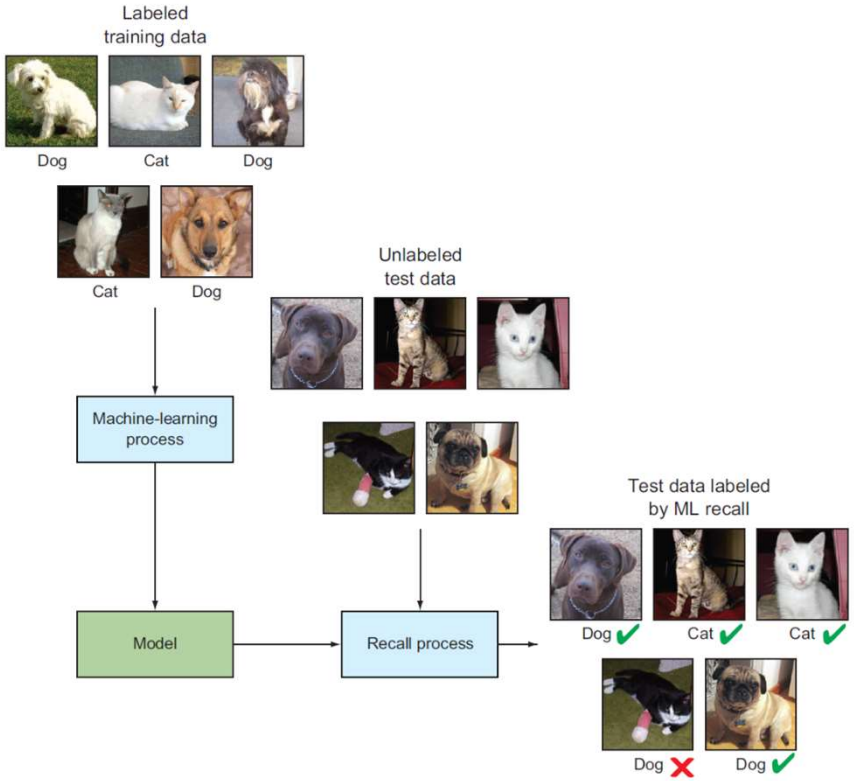
- Main Course Repo: https://github.com/thimotyb/real-world-machine-learning
- Pandas Cookbook: https://github.com/thimotyb/pandas-cookbook
- Pandas Exercises: https://github.com/thimotyb/pandas_exercises
- Python Lectures: https://github.com/thimotyb/Python-Lectures
- NumPy and MatPlotLib: https://github.com/thimotyb/AnatomyOfMatplotlib.git
- Linear Regression and Poly: https://github.com/thimotyb/python-machine-learning-book-2nd-edition.git
- NN, Keras, TensorFlow: https://github.com/thimotyb/handson-ml2.git

Go on your /home/user/git folder
git clone <repo_url>

What is Machine Learning?

- Machine learning refers to a specific body of knowledge and an associated set of techniques which employ a variety of strategies, in various combinations, depending on the problem at hand.
  - These strategies are embodied in collections of algorithms developed over the course of recent decades by academics and practitioners in various disciplines with various strengths and weaknesses.
  - Some of them are classifiers. Others predict a numeric measurement. Some measure the similarity or difference of comparable entities (for example, people, machines, processes, cats, dogs).
- What the algorithms have in common is learning from examples (experience) and the capacity to apply what they've learned to new, unseen cases—the ability to generalize.

The cats&dogs kaggle competition

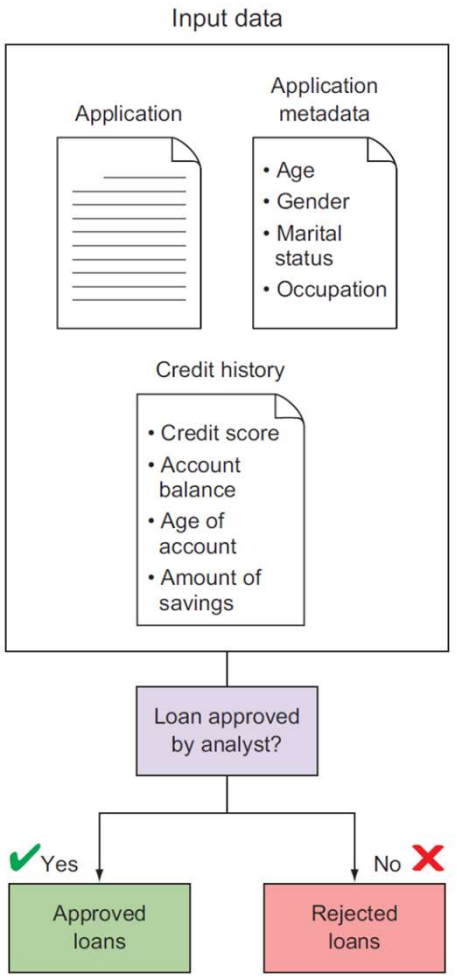


Some ML use cases for supervised learning

Problem	Description	Example use cases
Classification	Determine the discrete class to which each individual belongs, based on input data	Spam filtering, sentiment analysis, fraud detection, customer ad targeting, churn prediction, support case flagging, content personalization, detection of manufacturing defects, customer segmentation, event discovery, genomics, drug efficacy
Regression	Predict the real-valued output for each individual, based on input data	Stock-market prediction, demand forecasting, price estimation, ad bid optimization, risk management, asset management, weather forecasting, sports prediction
Recommendation	Predict which alternatives a user would prefer	Product recommendation, job recruiting, Netflix Prize, online dating, content recommendation
Imputation	Infer the values of missing input data	Incomplete patient medical records, missing customer data, census data

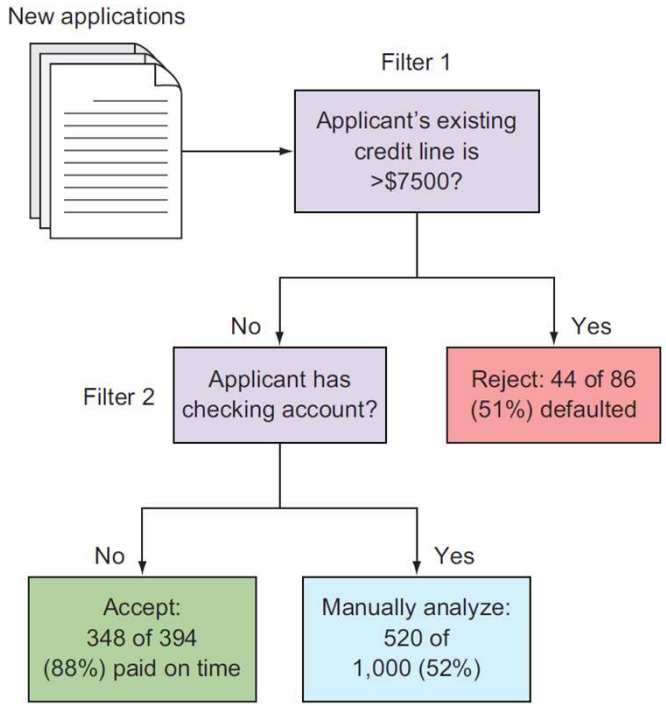
A first example: microlending process

- Need: automatize the criteria with which financial analyst approve loans in microlending
- High volumes
- Hiring more analysts create discrepancies and does not scale well



Using Declarative Business Rules

- Define some business rules: filters to limit manual processing and default decision on the rest
- Manually finding effective filters becomes harder and harder as the filtering system grows in complexity.
- The business rules become complicated and opaque that debugging them and ripping out old, irrelevant rules becomes virtually impossible.
- The construction of your rules has no statistical rigor.
- As the patterns of loan repayment change over time the system doesn't adapt to those changes.



The Machine Learning approach

Machine learning is an attractive option because is completely automated.

Unlike business rules, ML learns the optimal decisions **directly from the data** without having to arbitrarily hardcode decision rules.

In machine learning, the data provides the foundation for deriving insights about the problem at hand.

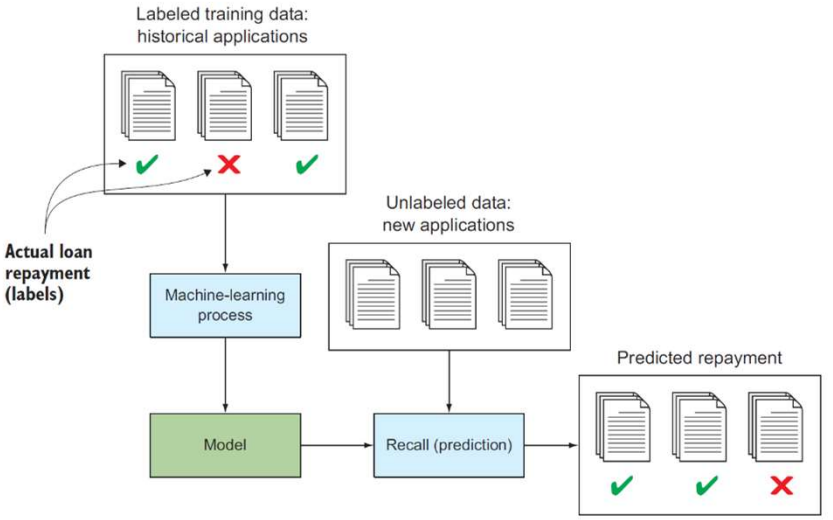
To determine whether to accept each new loan application, ML uses historical training data to predict the best course of action for each new application.

Training data for the Microlending case

To get started with ML for loan approval, you begin by assembling the training data for the 1,000 loans that have been granted.

This training data consists of the input data for each loan application, along with the known outcome of whether each loan was repaid on time.

The input data, in turn, consists of a set of features—numerical or categorical metrics that capture the relevant aspects of each application—such as the applicant's credit score, gender, and occupation.



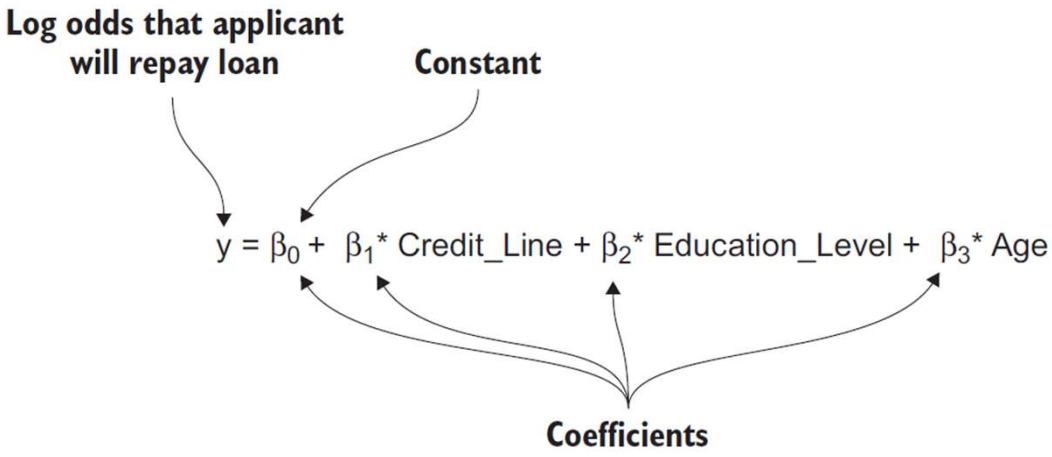
Selecting a ML algorithm

Machine learning comes in many flavors, ranging from simple statistical models to more-sophisticated approaches. There are two viable approaches for the microlending case:

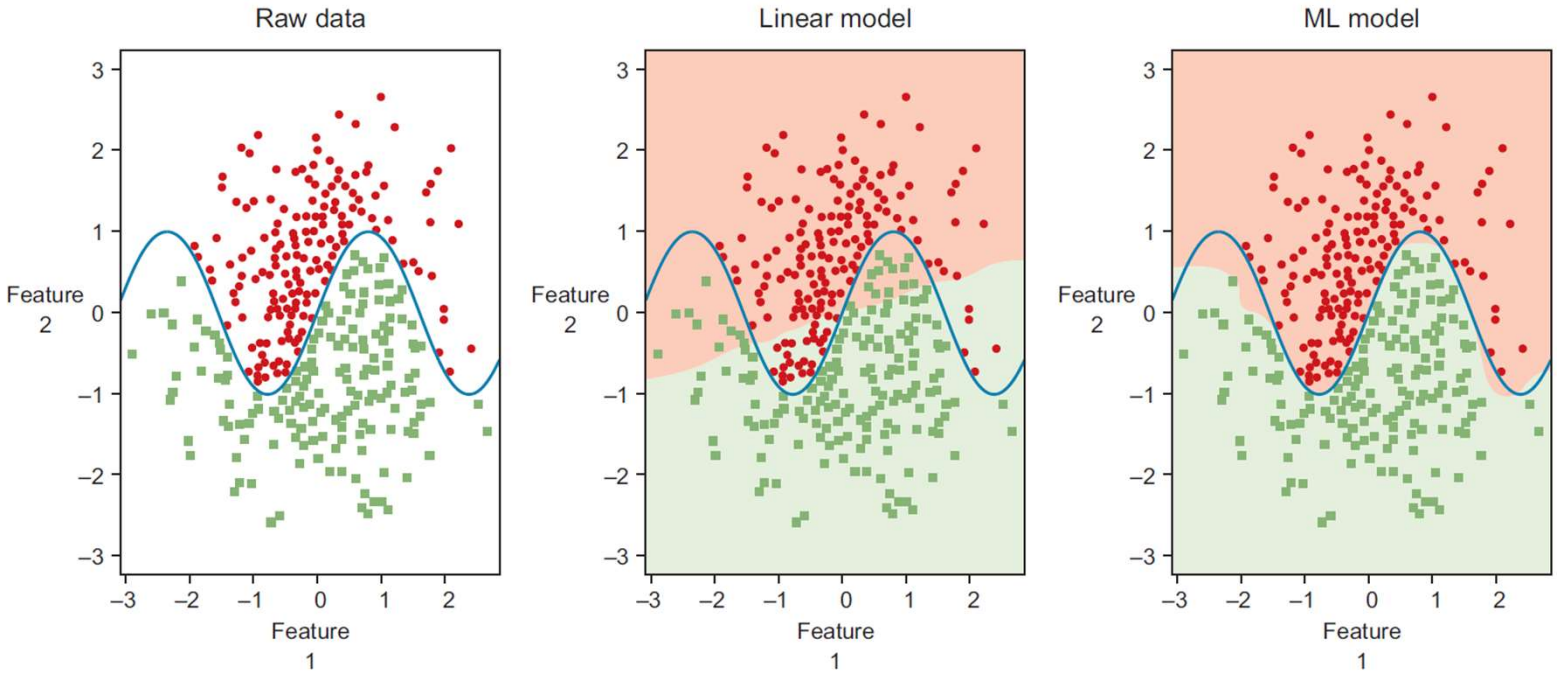
- using a simple parametric model: parametric models use simple, fixed equations to express the relationship between the outcome and the inputs. Data is then used to learn the best values of the unknown terms in the equation: linear regression, logistic regression, ...
- using a nonparametric set of classification trees: this is necessary when the optimal decision boundary is not linear. k-nearest neighbors (KNN), kernel smoothing, support vector machines, decision trees (random forests)...

A simple classifier: Logistic Regression

If each new application contains three relevant **features**—the applicant's credit line, education level, and age—then logistic regression attempts to predict the log odds that the applicant will default on the loan:



A more sophisticated classifier: Random Forest



Five advantages of Machine Learning

Accurate—ML uses data to discover the optimal decision-making engine for your problem. As you collect more data, the accuracy can increase automatically.

Automated—As answers are validated or discarded, the ML model can learn new patterns automatically. This allows users to embed ML directly into an automated workflow.

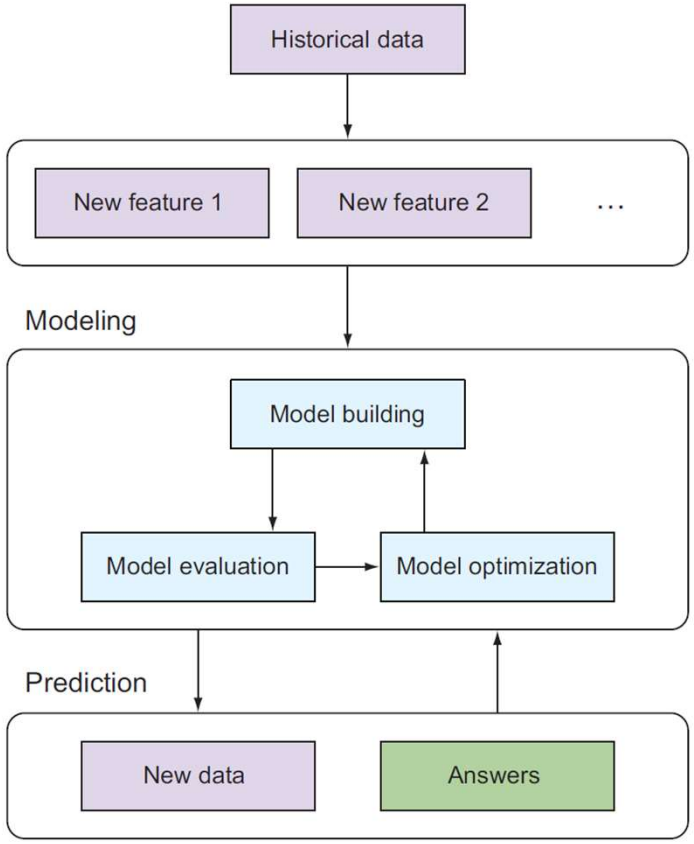
Fast—ML can generate answers in a matter of milliseconds as new data streams in, allowing systems to react in real time.

Customizable—Many data-driven problems can be addressed with machine learning. ML models are custom built from your own data, and can be configured to optimize whatever metric drives your business.

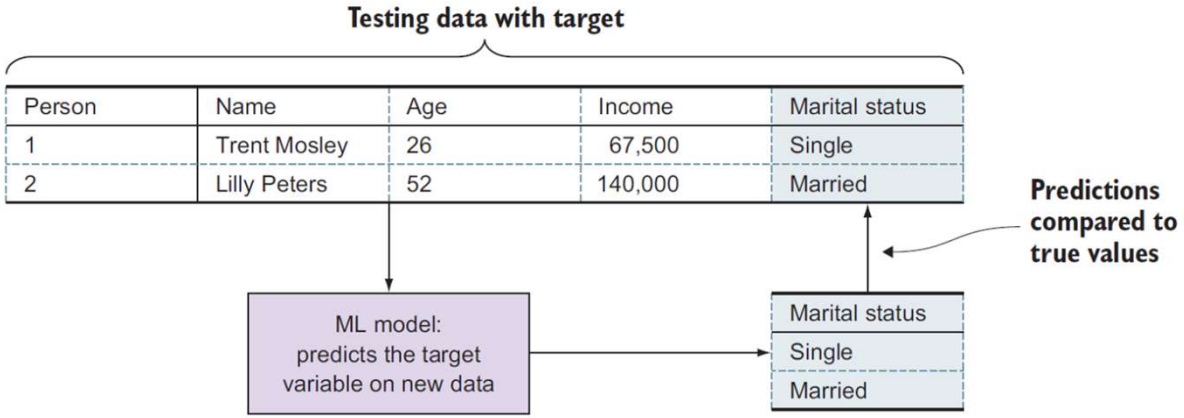
Scalable—As your business grows, ML easily scales to handle increased data rates. Some ML algorithms can scale to handle large amounts of data on many machines in the cloud.

Basic ML workflow

1) Data collection and preparation, with *feature engineering* techniques
2) Learning a model from *training* data (features + target). If necessary, *feature selection and prioritization*
   a) Evaluate the model using test data (the target is known)
   b) Run optimization steps
3) Using the model to perform a prediction



A simple dataset example




```
data = load_data(...)
training_data, testing_data = split_data(data)
model = build_model(training_data, target="Marital status")
true_values = testing_data.extract_column("Marital status")
predictions = model.predict(testing_data)
accuracy = compare_predictions(predictions, true_values)
```

Tuning the model performance

Tuning the model parameters—ML algorithms are configured with parameters specific to the underlying algorithm, and the optimal value of these parameters often depends on the type and structure of the data. The value of each parameter, or any of them combined, can have an impact on the performance of the model.

Selecting a subset of features—Many ML problems include a large number of features, and the noise from those features can sometimes make it hard for the algorithm to find the real signal in the data, even though they might still be informative on their own.

Data pre-processing - Most datasets need careful inspection and data wrangling to be reduced to a state suitable to be fed in any ML algorithm. The quality of data can impact the performance of the prediction. This includes scaling and normalization for algorithms which are sensitive to the quantitative scale of a feature. Other example include feature engineering on data types such as date, time, multimedia information, location, time series-related data, etc.

Gathering and preparing data

Main aspects of a ML problem

- Predicting a target variable (or variables) of interest
  - Is this purchase fraudolent or not?
  - Which of my customers will churn this month?
  - The target result may be binary, or classified on multiple classes (negative versus positive versus neutral), or even hundreds or thousands, and in others takes on numerical values.
- In statistics, the target is referred as the **response** or **dependent** variable.
- The data instances on which the model is built are called input features (**explanatory** or **independent** variables)
- The role of the ML algorithm is to use the training set extracted from the data to determine how the set of input features can most accurately predict the target variable.

A dataset example for the “churn” problem

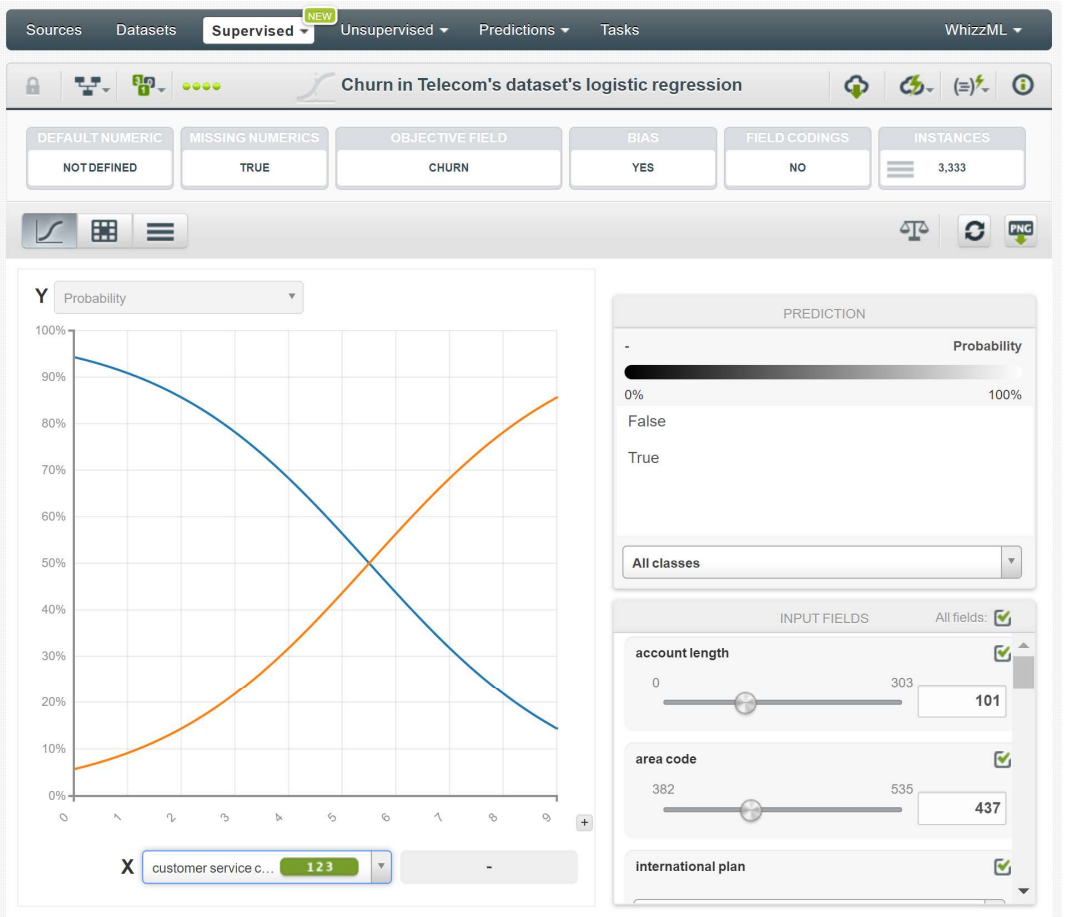
https://bigml.com/user/francisco/gallery/dataset/5163ad540c0b5e5b22000383

Features									Target
Cust. ID	State	Acct length	Area code	Int'l plan	Voicemail plan	Total messages	Total mins.	Total calls	Churned?
502	FL	124	561	No	Yes	28	251.4	104	False
1007	OR	48	503	No	No	0	190.4	92	False
1789	WI	63	608	No	Yes	34	152.2	119	False
2568	KY	58	606	No	No	0	247.2	116	True

Which input features should I include?
How do I obtain known values of my target variable?
How much training data do I need?
How do I know if my training data is good enough / representative enough (bias)?

Feature evaluation

- BigML platform
- Interactive Data Exploration
- Interactive Model Selection



A general approach to feature selection

1. Include all the features that you suspect to be predictive of the target variable.
2. Fit an ML model. If the accuracy of the model is sufficient, stop.
3. Otherwise, expand the feature set by including other features that are less obviously related to the target.
4. Fit another model and assess the accuracy. If performance is sufficient, stop.
5. Otherwise, starting from the expanded feature set, run an ML feature selection algorithm to choose the best, most predictive subset of your expanded feature set.

Obtaining ground truth on the target data

- One of the most difficult hurdles in getting started with supervised machine learning is the aggregation of training instances with a known target variable.
  - This process often requires running an existing, suboptimal system for a period of time, until enough training data is collected, sustaining the potential loss of getting to target
- Other ways of obtaining ground-truth values of the target variable include the following:
  - Dedicating analysts to manually look through past or current data to determine or estimate the ground-truth values of the target
  - Using crowdsourcing to use the “wisdom of crowds” in order to attain estimates of the target
  - Conducting follow-up interviews or other hands-on experiments with customers

How much training data do you need?

Very application specific

1. Using the current training set, choose a grid of subsample sizes to try. For example with this telecom training set of 3,333 instances of training data, your grid could be 500; 1,000; 1,500; 2,000; 2,500; 3,000.
2. For each sample size, randomly draw that many instances (without replacement) from the training set.
3. With each subsample of training data, build an ML model and assess the accuracy of that model
4. Assess how the accuracy changes as a function of sample size.

Data preprocessing for modeling

- The first issue to deal with is handling categorical features
- A feature is categorical if values can be placed in buckets and the order of values isn't important

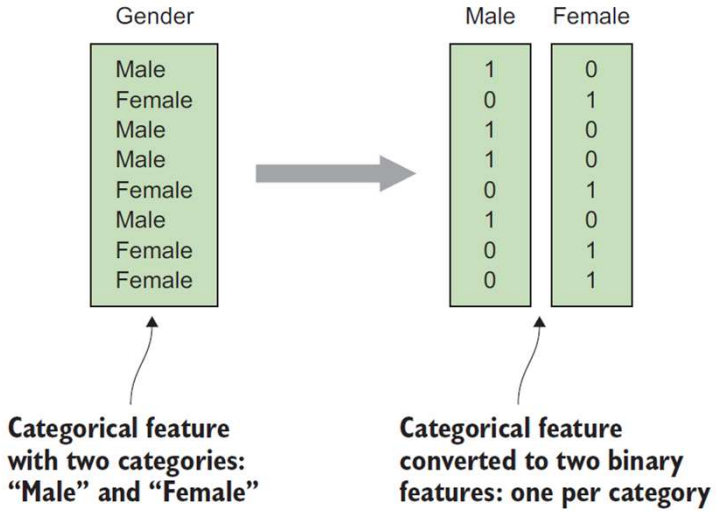
Person	Name	Age	Income	Marital status
1	Jane Doe	24	81,200	Single
2	John Smith	41	121,000	Married

Categorical features

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

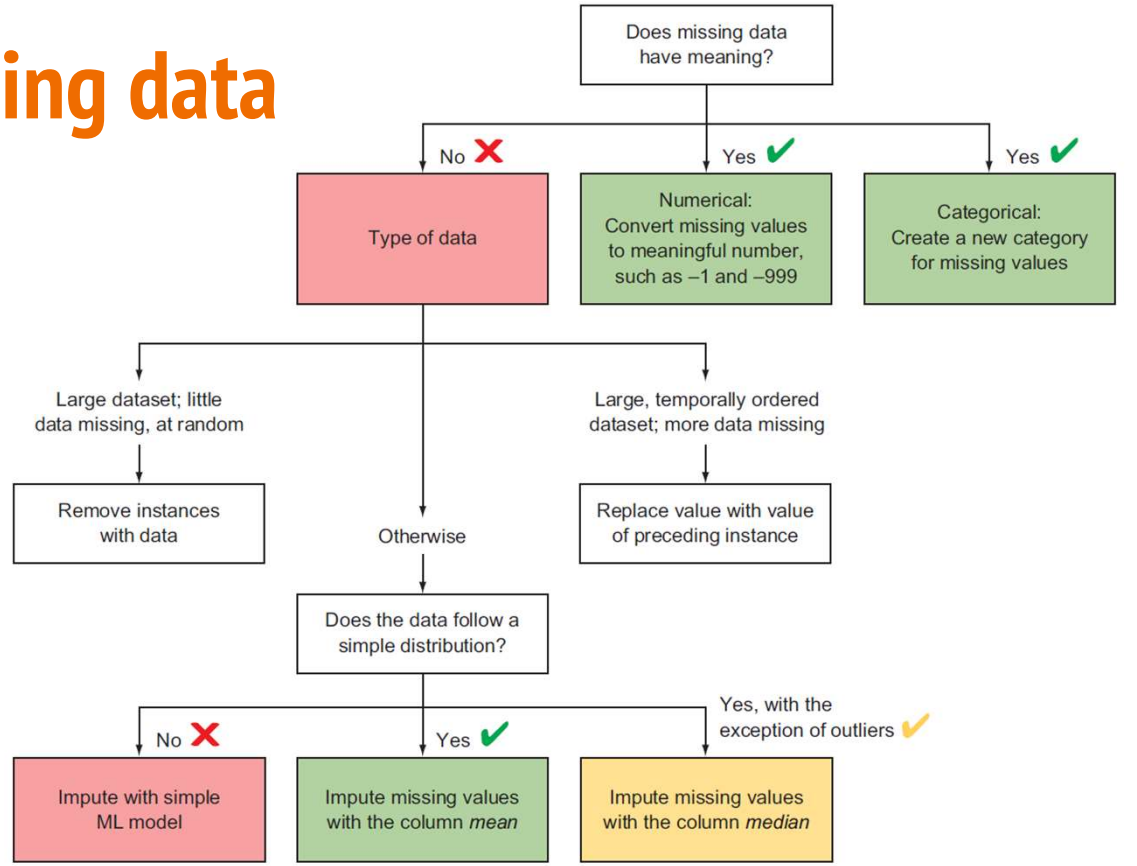
Handling categorical features

- It is not possible to encode categorical data as numbers (e.g. 1,2,3...) as a true categorical feature does not have an order, and such encoding creates an (arbitrary) order of categories
- Instead, convert each of the categories into a separate binary feature that has value 1 for instances for which the category appeared, and value 0 when it didn't



Dealing with missing data

- Usually NaN entries
- See pandas exercise



Lab 1: Introducing Python and Pandas

A Python primer

https://docs.python.org/3.7/tutorial/

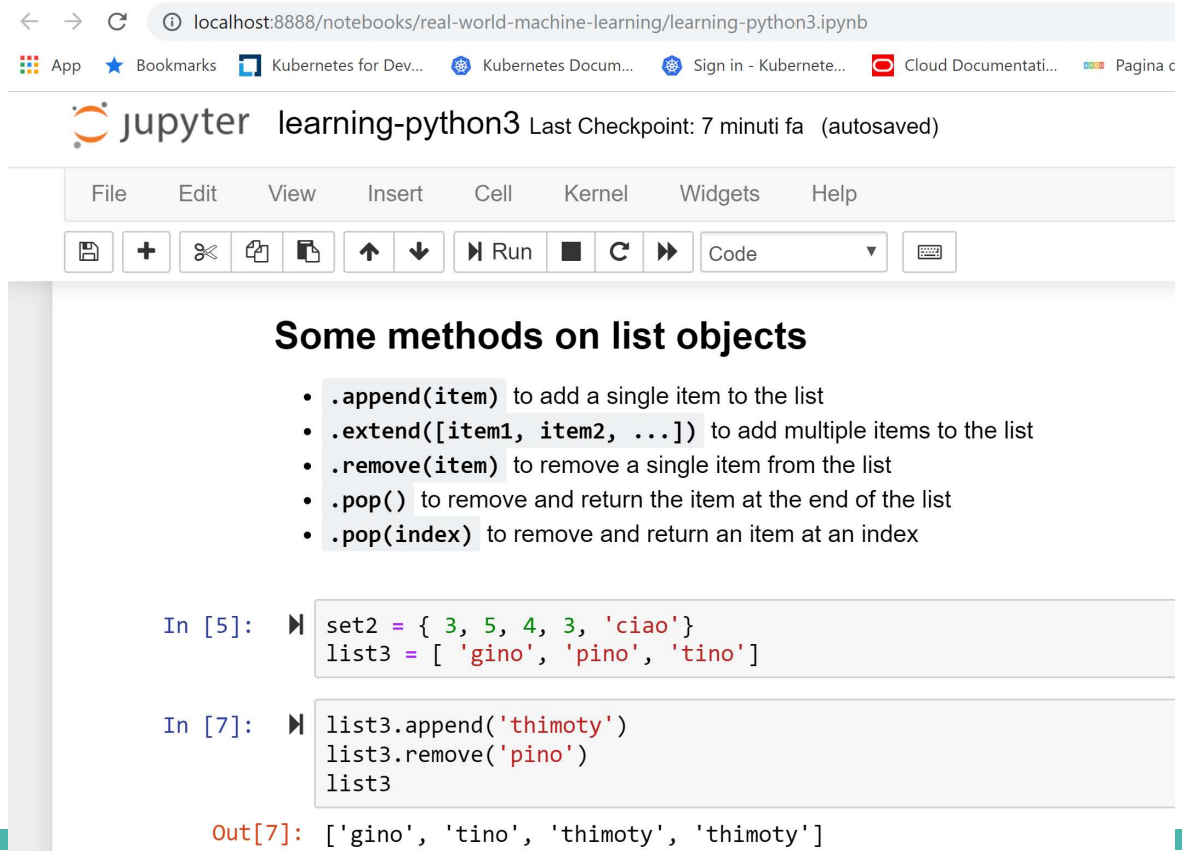
- Control Flow
- Function definition
- Data structures: Tuple, List, Dictionary
- Modules
- Exception handling
- Brief tour of stdlib

jupyter notebook

Python Exercises – Learn to use Jupyter Notebooks

- https://github.com/thimotyb/real-world-machine-learning/blob/python3/learning-python3.ipynb

- http://localhost:8888/notebooks/real-world-machine-learning/learning-python3.ipynb



Introduction to Numpy

- http://localhost:8888/notebooks/AnatomyOfMatplotlib-Part0-Intro2NumPy.ipynb
    - **Array Creation, Manipulation and Indexing**
    - **Broadcasting**

Introduction to Pandas

Data Structures

https://pandas.pydata.org/pandas-docs/stable/getting_started/dsintro.html

- Pandas User Guide: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- http://localhost:8888/notebooks/real-world-machine-learning/pandas-dataframe.ipynb

The basics of indexing are as follows:

Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

```
df.loc[df['column_name'] == some_value]
```

A Pandas primer – Exploring Data

- Some pandas exercises: https://github.com/thimotyb/pandas_exercises
  - http://localhost:8888/notebooks/pandas_exercises/01_Getting_%26_Knowing_Your_Data/Chipotle/Exercise_with_Solutions.ipynb
  - Do this exercise: http://localhost:8888/notebooks/pandas_exercises/01_Getting_%26_Knowing_Your_Data/Occupation/Exercises.ipynb

-

Pandas Primer – Grouping and Filtering, Apply Lambda

- http://localhost:8888/notebooks/pandas_exercises/02_Filtering_%26_Sorting/Chipotle/Exercises_with_solutions.ipynb
- http://localhost:8888/notebooks/pandas_exercises/03_Grouping/Alcohol_Consumption/Exercise_with_solutions.ipynb
- http://localhost:8888/notebooks/pandas_exercises/04_Apply/US_Crime_Rates/Exercises_with_solutions.ipynb

- Do these Exercises
  - http://localhost:8888/notebooks/pandas_exercises/03_Grouping/Occupation/Exercise.ipynb
  - http://localhost:8888/notebooks/pandas_exercises/02_Filtering_%26_Sorting/Fictional%20Army/Exercise.ipynb

Pandas – Stats and Visualisation

- http://localhost:8888/notebooks/pandas_exercises/06_Stats/US_Baby_Names/Exercises_with_solutions.ipynb#Step-4.-See-the-first-10-entries
- http://localhost:8888/notebooks/pandas_exercises/07_Visualization/Chipotle/Exercise_with_Solutions.ipynb
    - Do the exercise in the last row

Pandas – Series, Data Frames, and Time Series

- http://localhost:8888/notebooks/pandas_exercises/08_Creating_Series_and_DataFrames/Pokemon/Exercises-with-solutions-and-code.ipynb
- http://localhost:8888/notebooks/pandas_exercises/09_Time_Series/Apple_Stock/Exercises-with-solutions-code.ipynb
- http://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html
- Do this exercise:
    - http://localhost:8888/notebooks/pandas_exercises/09_Time_Series/Investor_Flow_of_Funds_US/Exercises.ipynb

Pandas Cookbook

- Pandas tutorial (Python 2.7): https://github.com/thimotyb/pandas-cookbook

    - Reading data from a csv file
    - Handling dataframes and plotting
    - Data and String manipulation
    - Data cleaning

Feature Engineering - basics

http://localhost:8888/notebooks/real-world-machine-learning/Chapter%202%20-%20Data%20Processing.ipynb

Let's analyze the feature engineering example in IPython:

Converting categorical data to numerical features

In [2]:
```
cat_data = array(['male', 'female', 'male', 'male', 'female', 'male', 'female', 'female'])
```

In [3]:
```
def cat_to_num(data):
    categories = unique(data)
    features = []
    for cat in categories:
        binary = (data == cat)
        features.append(binary.astype("int"))
    return features
```

In [4]:
```
cat_to_num(cat_data)
```

Out[4]:
```
[array([0, 1, 0, 0, 1, 0, 1, 1]), array([1, 0, 1, 1, 0, 1, 0, 0])]
```

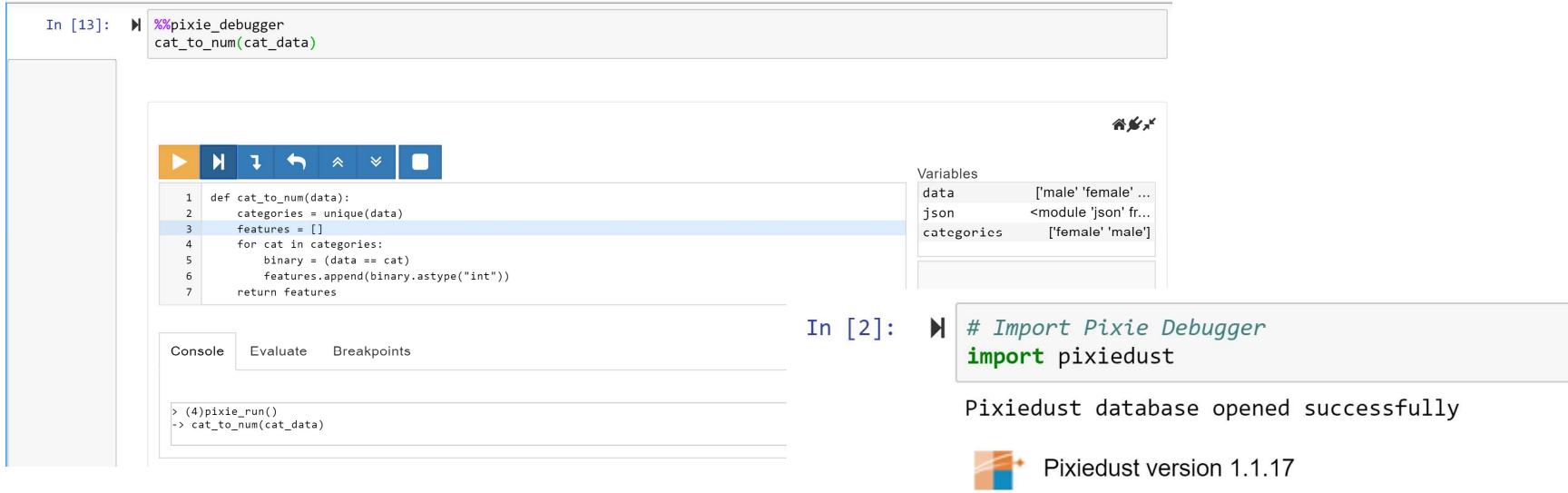
Conversion of categorical features in binary form

- Example of conversion algorithm in Python

```
def cat_to_num(data):
    categories = unique(data)
    features = []
    for cat in categories:
        binary = (data == cat)
        features.append(binary.astype("int"))
    return features
```

Inspect the flow with Pixie Debugger

- https://medium.com/codait/the-visual-python-debugger-for-jupyter-notebooks-youve-always-wanted-761713babc62



Feature engineering on the Titanic Data

Feature engineering: using the existing features to create new features that increase the value of the original data by applying our knowledge of the data or domain in question: e.g. interpret cabin numbering codes to extract topology

```
def cabin_features(data):
    features = []
    for cabin in data:
        cabins = cabin.split(" ")
        n_cabins = len(cabins)
        # First char is the cabin_char
        try:
            cabin_char = cabins[0][0]
        except IndexError:
            cabin_char = "X"
            n_cabins = 0
        # The rest is the cabin number
        try:
            cabin_num = int(cabins[0][1:])
        except:
            cabin_num = -1
        # Add 3 features for each passanger
        features.append( [cabin_char, cabin_num, n_cabins] )
    return features
```

Data Normalization

```
def normalize_feature(data, f_min=-1.0, f_max=1.0):
    d_min, d_max = min(data), max(data)
    factor = (f_max - f_min) / (d_max - d_min)
    normalized = f_min + (data - d_min)*factor
    return normalized, factor
```

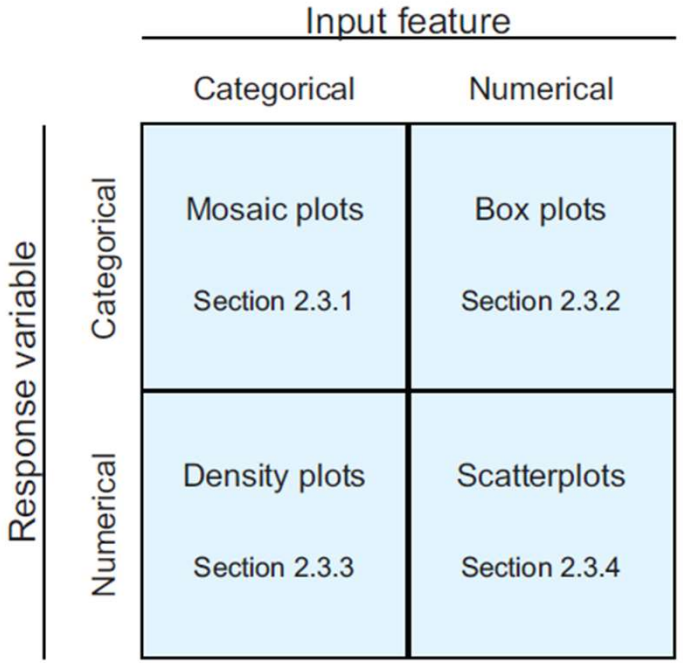
- Some ML algorithms require data to be normalized, meaning that each individual feature has been manipulated to reside on the same numeric scale.
- The value range of a feature can influence the importance of the feature compared to other features.
- To make sure all features are considered equally, you need to normalize the data. Often data is normalized to be in the range from 0 to 1, or from –1 to 1.

Lab 2: Visualisation on some datasets using Python and matplotlib

Purpose of data visualisation

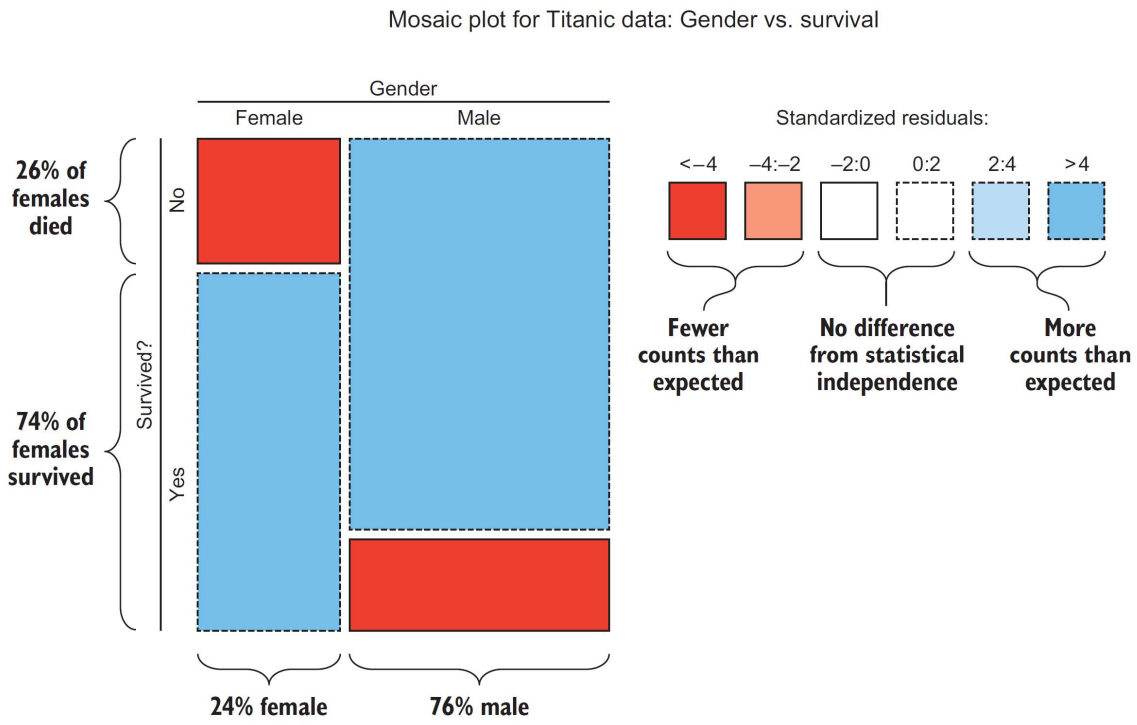
Data visualization serves as a sanity check of the training features and target variable before diving into the mechanics of machine learning and prediction.

The type of plot depends from the types of exploratory and response variable.



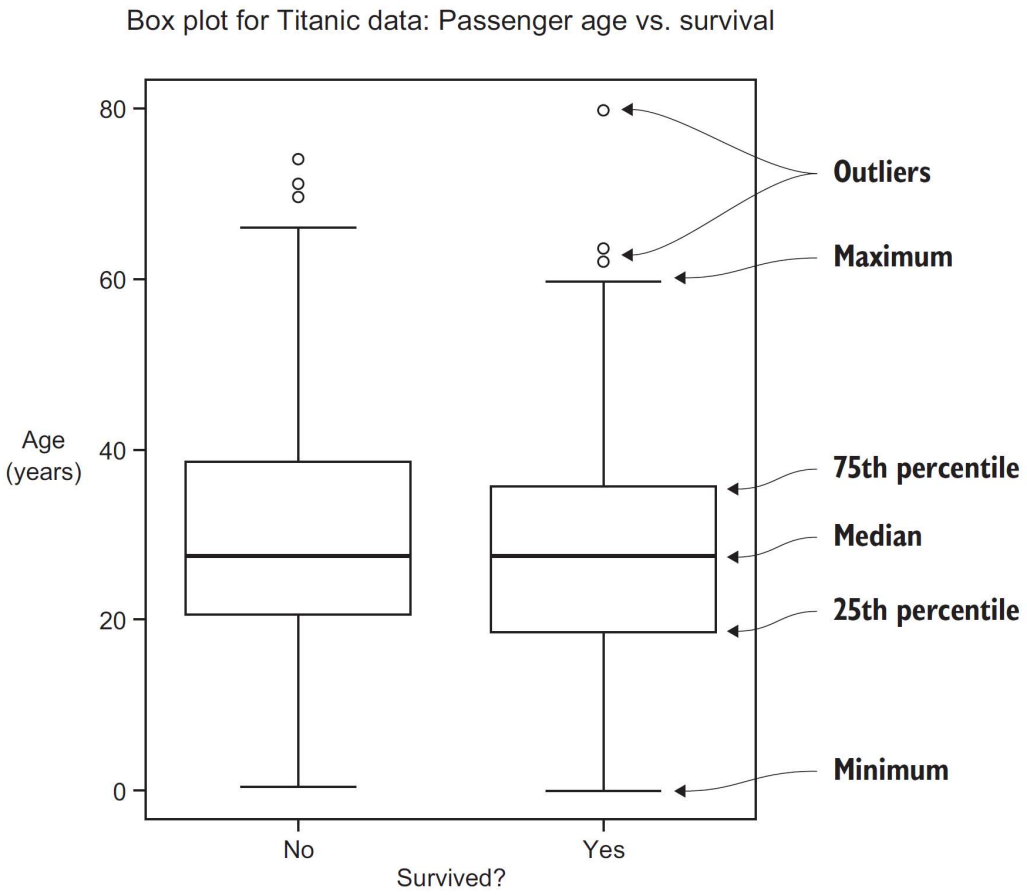
Mosaic Plot

Mosaic plots allow you to visualize the relationship between two or more categorical variables



Box Plot

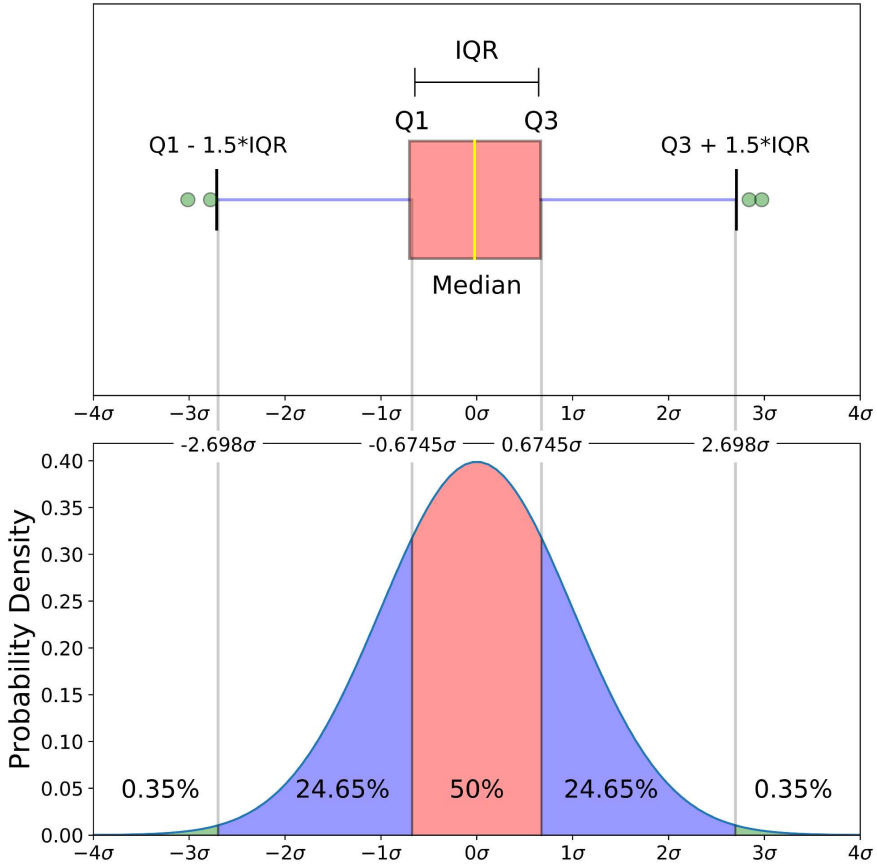
- Box plots are a standard statistical plottin technique for visualizing the distribution of a numerical variable. For a single variable, a box plot depicts the quartiles o its distribution: the minimum, 25th percentile, median, 75th percentile, and maximum of the values.
- Box-plot visualization of a single variable i useful to get insight into the center, spread, and skew of its distribution of values plus the existence of any outliers
- A percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations falls. For example, the 20th percentile is the value (or score) below which 20% of the observations may be found.



Understanding Box Plots

- **median (Q2/50th Percentile)**: the middle value of the dataset.
- **first quartile (Q1/25th Percentile)**: the middle number between the smallest number (not the "minimum") and the median of the dataset.
- **third quartile (Q3/75th Percentile)**: the middle value between the median and the highest value (not the "maximum") of the dataset.
- **interquartile range (IQR)**: 25th to the 75th percentile.
- **whiskers (shown in blue)**
- **outliers (shown as green circles)**
- **"maximum"**: Q3 + 1.5*IQR
- **"minimum"**: Q1 -1.5*IQR

https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51



Mean

The *sample mean*, denoted as $\bar{x}$, can be calculated as

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n},$$

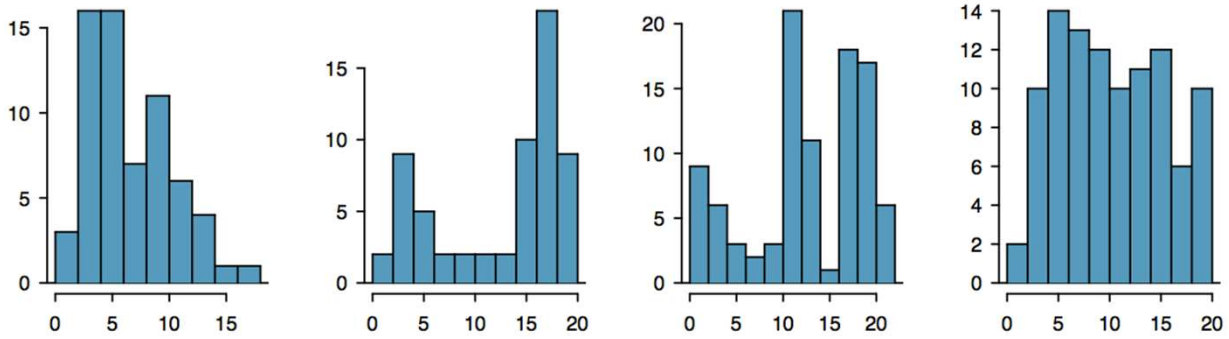
where x_1, x_2, ..., x_n represent the *n* observed values.

The *population mean* is also computed the same way but is denoted as *μ*. It is often not possible to calculate μ since population data are rarely available.

The sample mean is a *sample statistic*, and serves as a *point estimate* of the population mean. This estimate may not be perfect, but if the sample is good (representative of the population), it is usually a pretty good estimate.

Shape of a Distribution: Modality

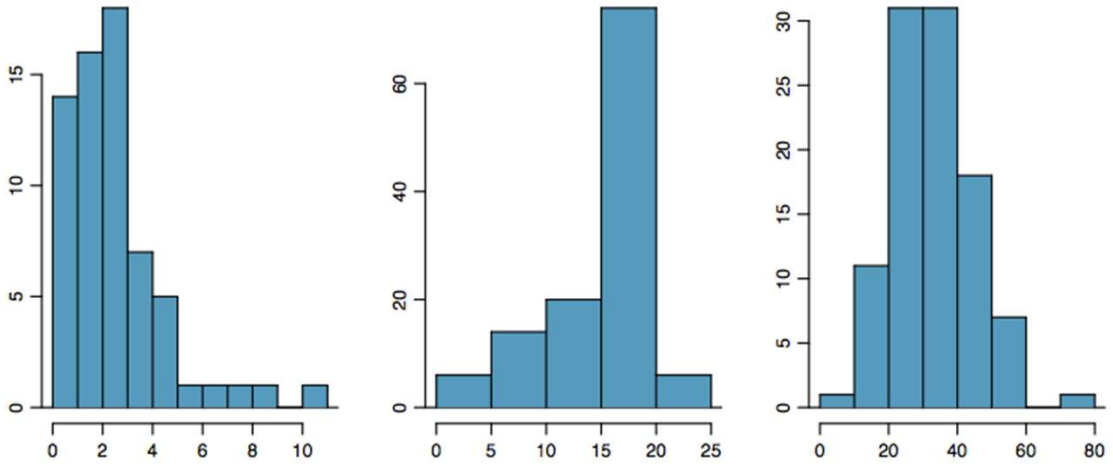
Does the histogram have a single prominent peak (*unimodal*), several prominent peaks (*bimodal/multimodal*), or no apparent peaks (*uniform*)?



Note: In order to determine modality, step back and imagine a smooth curve over the histogram -- imagine that the bars are wooden blocks and you drop a limp spaghetti over them, the shape the spaghetti would take could be viewed as a smooth curve.

Shape of a Distribution: Skewness

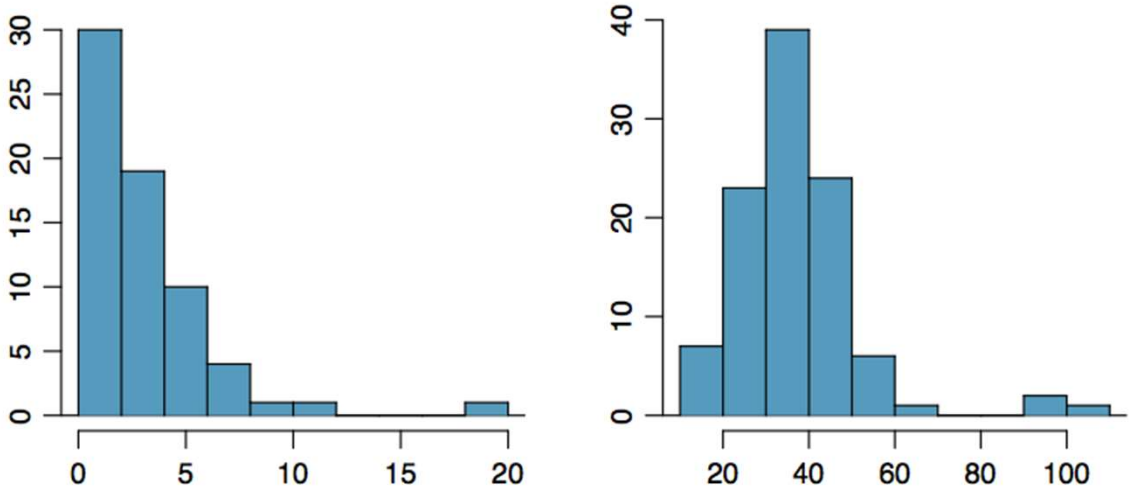
Is the histogram *right skewed*, *left skewed*, or *symmetric*?



Note: Histograms are said to be skewed to the side of the long tail.

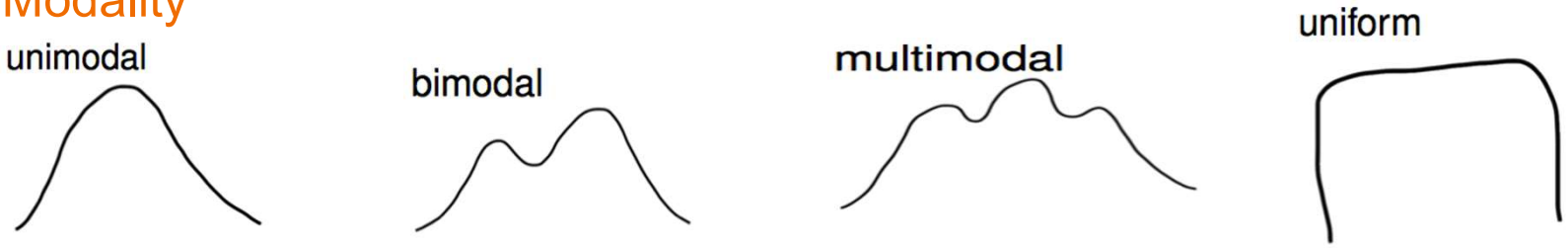
Shape of a Distribution: Unusual Observations

Are there any unusual observations or potential *outliers*?

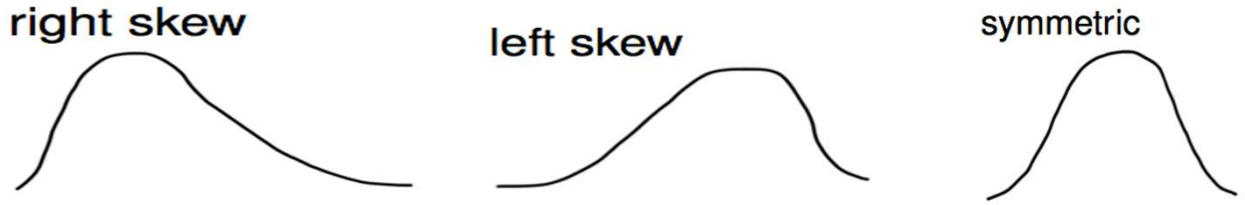


Commonly observed shapes of distributions

Modality



Skewness

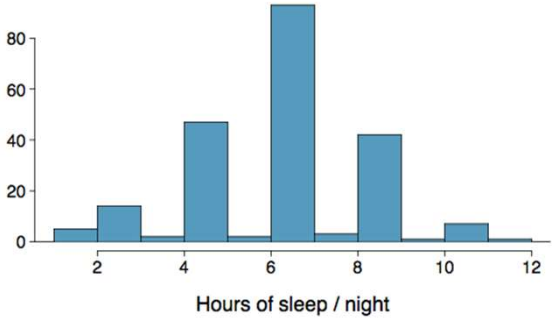


Variance

Variance is roughly the average squared deviation from the mean.

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

- The sample mean is $\bar{x} = 6.71$, and the sample size is n = 217.
- The variance of amount of sleep students get per night can be calculated as:



$$s^2 = \frac{(5-6.71)^2 + (9-6.71)^2 + \cdots + (7-6.71)^2}{217-1} = 4.11\ hours^2$$

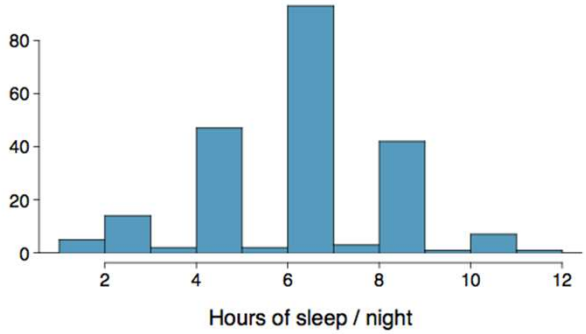
Standard Deviation

The *standard deviation* is the square root of the variance, and has the same units as the data.

$$s = \sqrt{s^2}$$

- The standard deviation of amount of sleep students get per night can be calculated as:

  $s = \sqrt{4.11} = 2.03\ hours$



- We can see that all of the data are within 3 standard deviations of the mean.

Median

The *median* is the value that splits the data in half when ordered in ascending order.

$$0, 1, 2, 3, 4$$

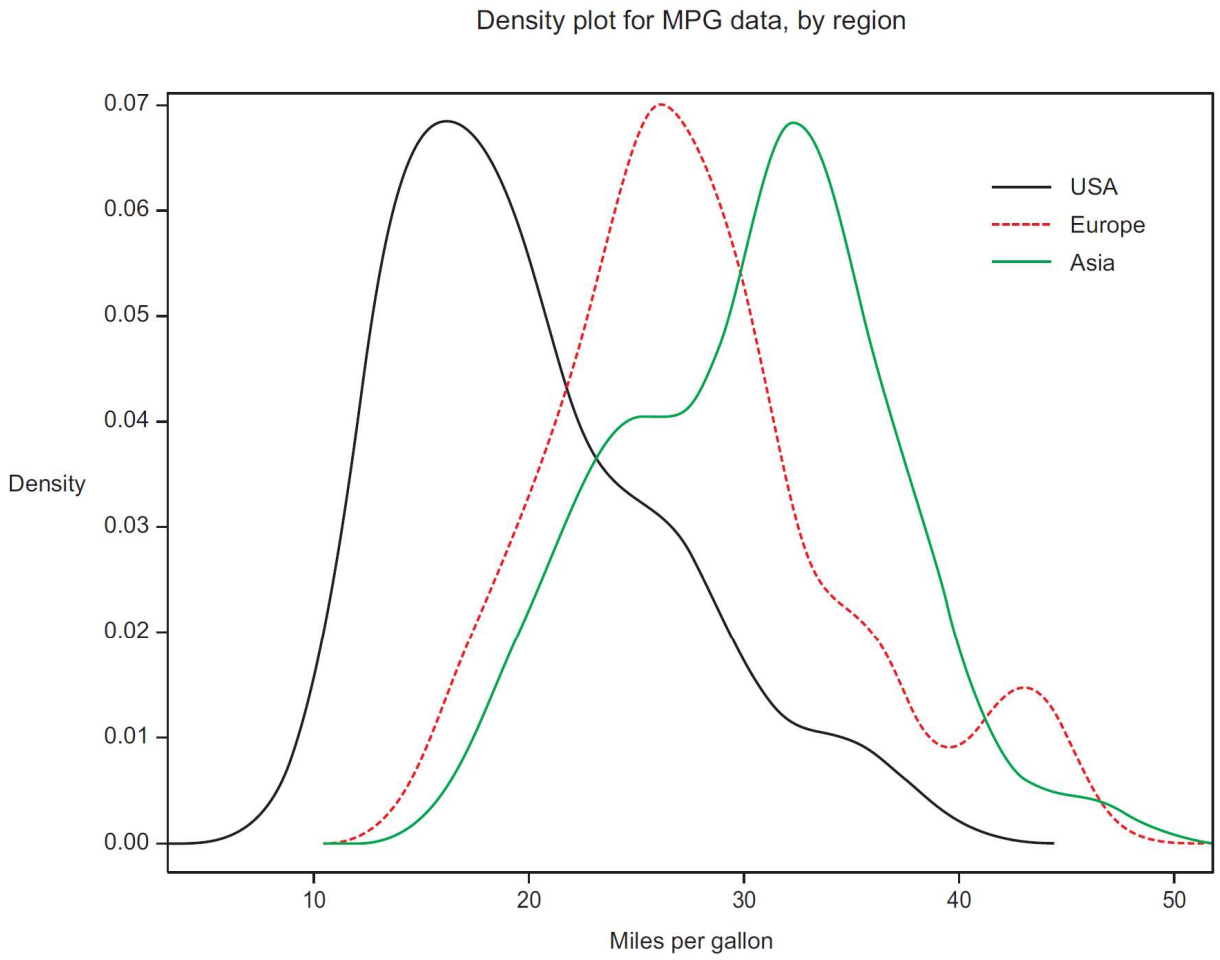
If there are an even number of observations, then the median is the average of the two values in the middle.

$$0, 1, \underline{2, 3}, 4, 5 \rightarrow \frac{2+3}{2} = 2.5$$

Since the median is the midpoint of the data, 50% of the values are below it. Hence, it is also the 50th percentile.

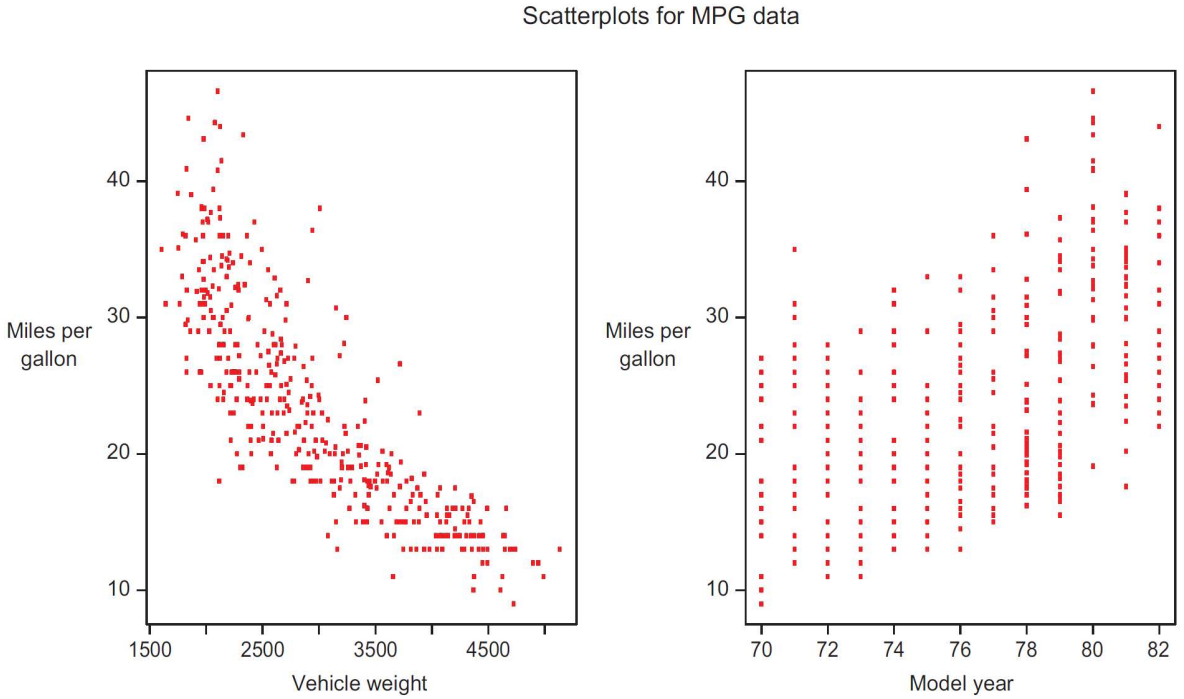
Density Plot

- *Density plots* display the distribution of a single variable in more detail than a box plot.



Scatter Plot

A *scatter plot* is a simple visualization of the relationship between two numerical variables and is one of the most popular plotting tools in existence. In a scatter plot, the value of the feature is plotted versus the value of the response variable, with each instance represented as a dot.



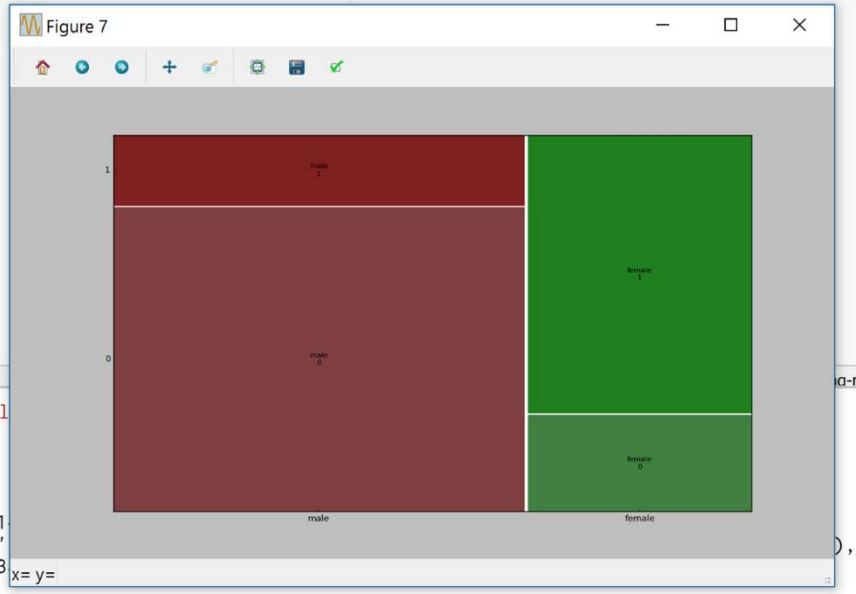
Primer - Matplotlib

- https://matplotlib.org/gallery/index.html
- http://localhost:8888/notebooks/AnatomyOfMatplotlib/AnatomyOfMatplotlib-Part1-Figures_Subplots_and_layouts.ipynb
- http://localhost:8888/notebooks/AnatomyOfMatplotlib/AnatomyOfMatplotlib-Part2-Plotting_Methods_Overview.ipynb
- http://localhost:8888/notebooks/AnatomyOfMatplotlib/AnatomyOfMatplotlib-Part3-HowToSpeakMPL.ipynb
- http://localhost:8888/notebooks/AnatomyOfMatplotlib/Examples_from_Gallery.ipynb
- Exercise: Practice with other examples taken from the Gallery

A mosaic plot on Titanic data in Python

titanic_example.py svm_gui.py

```
1 import pandas as pd
2 import numpy as np
3 import os
4 from statsmodels.graphics.mosaicplot import mosaic
5
6
7 os.chdir('C:/Users/thimo/Dropbox/corsi/machine_learning/real-world-machine-learning-master/')
8
9 #Create a mosaic plot for titanic
10 df = pd.read_csv('./data/titanic.csv')
11 mosaic(df, ['Sex', 'Survived'])
12
13 # boxplot
14 df.boxplot(column='Age', by='Survived')
15
16
```



Python

```
In [21]: %run "C:\Users\thimo\Dropbox\corsi\machine_l

In [22]: mosaic(df, ['Sex', 'Survived'])
Out[22]:
(<matplotlib.figure.Figure at 0x18592e80>,
 OrderedDict([(('male', '0'), (0.0, 0.0, 0.6443651551
0.6443651551445914, 0.18828054376802911)), (('female'
(('female', '1'), (0.6493402795227009, 0.260427026683
```

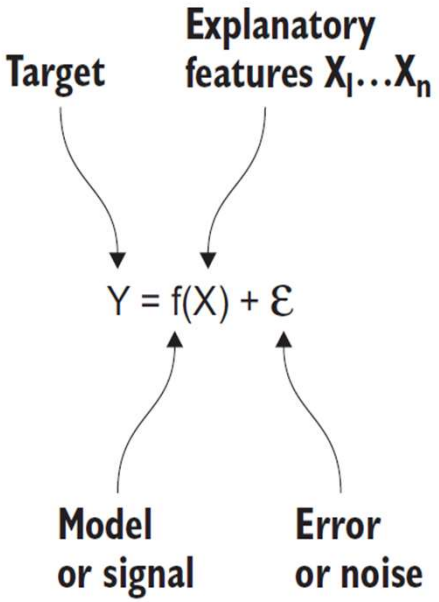
Modeling and Prediction

Basics of Modeling: Signal, Noise

Let's consider the MPG auto dataset as an example: it contains metrics about automobiles, and the target is determine the MPG (Miles Per Gallon) based on features (weight, cc...);
auto = pandas.read_csv("data/auto-mpg.csv")

Input features are typically referred to using the symbol X, with subscripts differentiating inputs when multiple input features exist. X1 = manufacturer region, X2 = model year, ecc. The target variable is typically referred to as Y.

f: **signal**, estimated via ML using a model
e: **noise**, determined by measurement error, manufacturing fluctuations, ecc.



Purpose of the model: Prediction and Inference

Prediction: The model is used to generate predictions of the target, Y, for new data, Xnew, by plugging those new features into the model. fest denotes the machine-learning estimate of f (the true relationship between the features and the target): Ypred = fest(Xnew)

"What would the MPG of a certain automobile with known input metrics be?"

Inference: machine-learning models allow to better understand the relationships between the input features and the output target.

1. Which input features are most strongly related to the target variable?
2. Are those relationships positive or negative?
3. Is f a simple relationship, or is it a function that's more nuanced and nonlinear?

Types of modeling methods

Statistical modeling has a general trade-off between predictive accuracy and model interpretability.

- Simple models are easy to interpret, yet won't produce accurate predictions (particularly for complicated relationships).
- Complex models may produce accurate predictions, but may be black-box and hard to interpret.

Two main types: parametric and nonparametric.

- parametric models assume that f takes a specific functional form
- nonparametric models don't make such strict assumptions.

Parametric approaches tend to be simple and interpretable, but less accurate. Likewise, nonparametric approaches are usually less interpretable but more accurate across a broad range of problems.

Parametric methods

The simplest example of a parametric approach is linear regression. In linear regression,*f* is assumed to be a linear combination of the numerical values of the inputs.

$$f(\mathbf{X}) = \beta_0 + X_1 \times \beta_1 + X_2 \times \beta_2 + \ldots$$

In this equation, the unknown parameters, 0, 1,... can be interpreted as the intercept and slope parameters (with respect to each of the inputs). When you fit a parametric model to some data, you estimate the best values of each of the unknown parameters. Then you can turn around and plug those estimates into the formula for f(X) along with new data to generate predictions.

Parametric methods

Another common parametric method is Logistic Regression (used for Classification problems).

The drawback of parametric approaches is that they make strong assumptions about the true form of the function f.

In most real-world problems, f doesn't assume such a simple form, especially when there are many input variables (X).

In these situations, parametric approaches will fit the data poorly, leading to inaccurate predictions.

Non-parametric methods

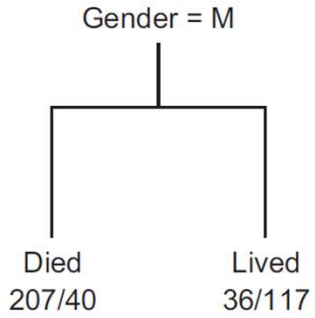
In nonparametric models, *f* doesn't take a simple, fixed function. Instead, the form and complexity of *f* adapts to the complexity of the data. A simple example of a nonparametric model is a classification tree. A classification tree is a series of recursive binary decisions on the input features.

The classification tree learning algorithm uses the target variable to learn the optimal series of splits such that the terminal leaf nodes of the tree contain instances with similar values of the target.

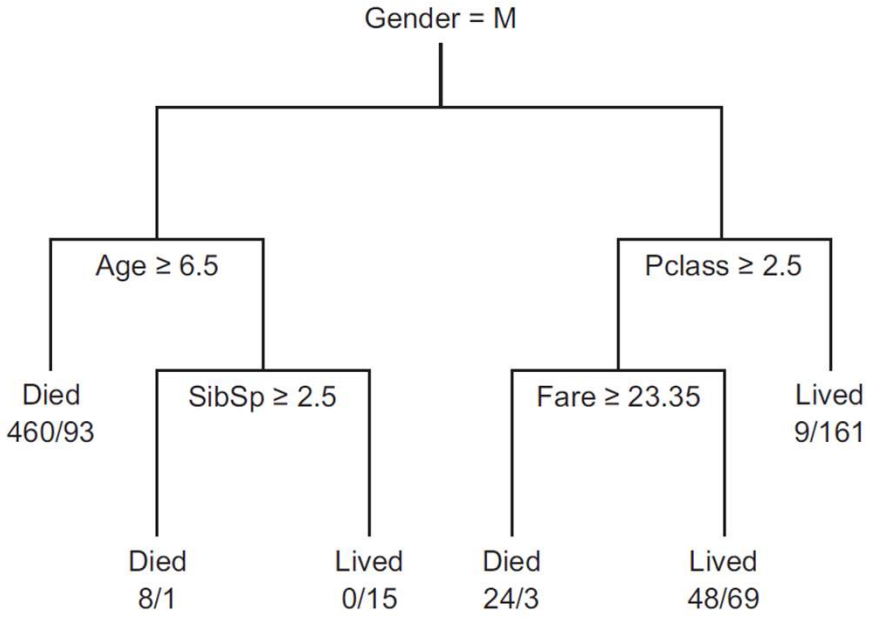
Classification trees are nonparametric because the depth and complexity of the tree isn't fixed in advance, but rather is learned from the data. If the relationship between the target variable and the input features is complex and there's a sufficient amount of data, then the tree will grow deeper, uncovering more-nuanced patterns.

Classification Tree Example

Classification tree: Small amount of data



Classification tree: Large amount of data



Supervised versus unsupervised learning

Supervised problems are ones in which you have access to the target variable for a set of training data

Unsupervised problems are ones in which there's no identified target variable.

Most ML techniques are designed to solve supervised problems, and this course will consider only supervised techniques (Regression and Classification)

Types of unsupervised problems

Clustering—Use the input features to discover natural groupings in the data and to divide the data into those groups. Methods: k-means, Gaussian mixture models, and hierarchical clustering.

Dimensionality reduction—Transform the input features into a small number of coordinates that capture most of the variability of the data. Methods: principal component analysis (PCA), multidimensional scaling, manifold learning.

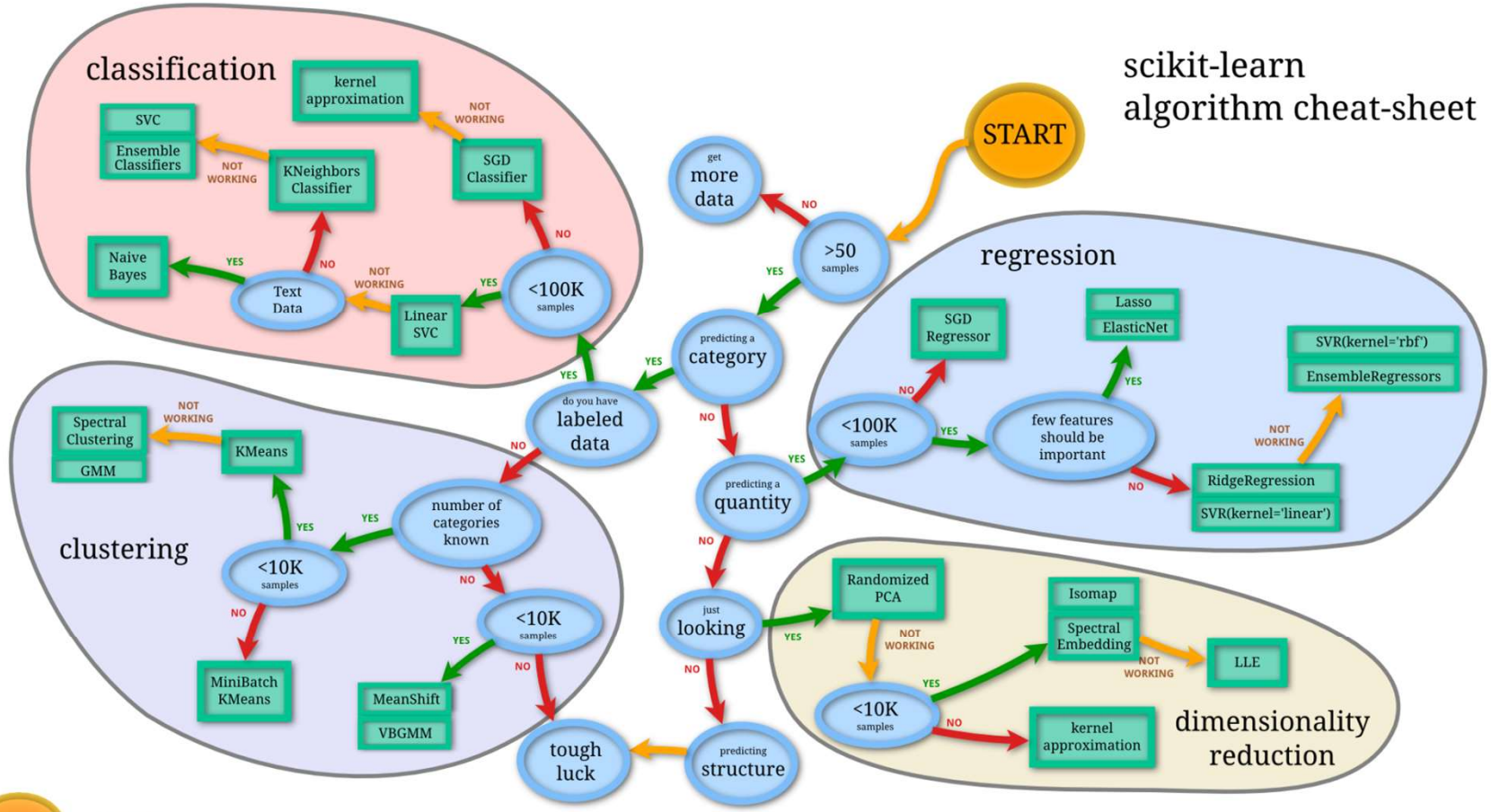
A list of possible supervised models

Name	Type	Use	Linear/ nonlinear	Requires normalization
Linear regression	Regression	Model a scalar target with one or more quantitative features. Although regression computes a linear combination, features can be transformed by nonlinear functions if relationships are known or can be guessed. R: www.inside-r.org/r-doc/stats/lm Python: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression	Linear	Yes
Logistic regression	Classification	Categorize observations based on quantitative features; predict target class or probabilities of target classes. R: www.statmethods.net/advstats/glm.html Python: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html	Linear	Yes

SVM	Classification/regression	Classification based on separation in high-dimensional space. Predicts target classes. Target class probabilities require additional computation. Regression uses a subset of the data, and performance is highly data dependent. R: https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf Python: http://scikit-learn.org/stable/modules/svm.html	Linear	Yes
SVM with kernel	Classification/regression	SVM with support for a variety of nonlinear models. R: https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf Python: http://scikit-learn.org/stable/modules/svm.html	Nonlinear	Yes

K-nearest neighbors	Classification/regression	Targets are computed based on those of the training set that are “nearest” to the test examples via a distance formula (for example, Euclidean distance). For classification, training targets “vote.” For regression, they are averaged. Predictions are based on a “local” subset of the data, but are highly accurate for some datasets. R: https://cran.r-project.org/web/packages/class/class.pdf Python: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html	Nonlinear	Yes
Decision trees	Classification/regression	Training data is recursively split into subsets based on attribute value tests, and decision trees that predict targets are derived. Produces understandable models, but random forest and boosting algorithms nearly always produce lower error rates. R: www.statmethods.net/advstats/cart.html Python: http://scikit-learn.org/stable/modules/tree.html#tree	Nonlinear	No

Random forest	Classification/regression	An "ensemble" of decision trees is used to produce a stronger prediction than a single decision tree. For classification, multiple decision trees "vote." For regression, their results are averaged. R: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf Python: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html	Nonlinear	No
Boosting	Classification/regression	For multitree methods, boosting algorithms reduce generalization error by adjusting weights to give greater weight to examples that are misclassified or (for regression) those with larger residuals. R: https://cran.r-project.org/web/packages/gbm/gbm.pdf https://cran.r-project.org/web/packages/adabag/adabag.pdf Python: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html	Nonlinear	No


scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
<100K samples
Text Data
Naive Bayes
Linear SVC
KNeighbors Classifier
SVC
Ensemble Classifiers
SGD Classifier
kernel approximation
NOT WORKING
YES
NO
classification
<100K samples
SGD Regressor
Lasso
ElasticNet
few features should be important
SVR(kernel='rbf')
EnsembleRegressors
RidgeRegression
SVR(kernel='linear')
regression
number of categories known
<10K samples
KMeans
Spectral Clustering
GMM
MiniBatch KMeans
<10K samples
MeanShift
VBGMM
clustering
Randomized PCA
Isomap
Spectral Embedding
LLE
<10K samples
kernel approximation
dimensionality reduction

Back

Bias and Variance Trade-off in a Model

It can be shown that the prediction error of a model can always be decomposed into the sum of three fundamental quantities:

the variance of ˆ f(x0),
the squared bias of ˆ f(x0) and
the variance of the error

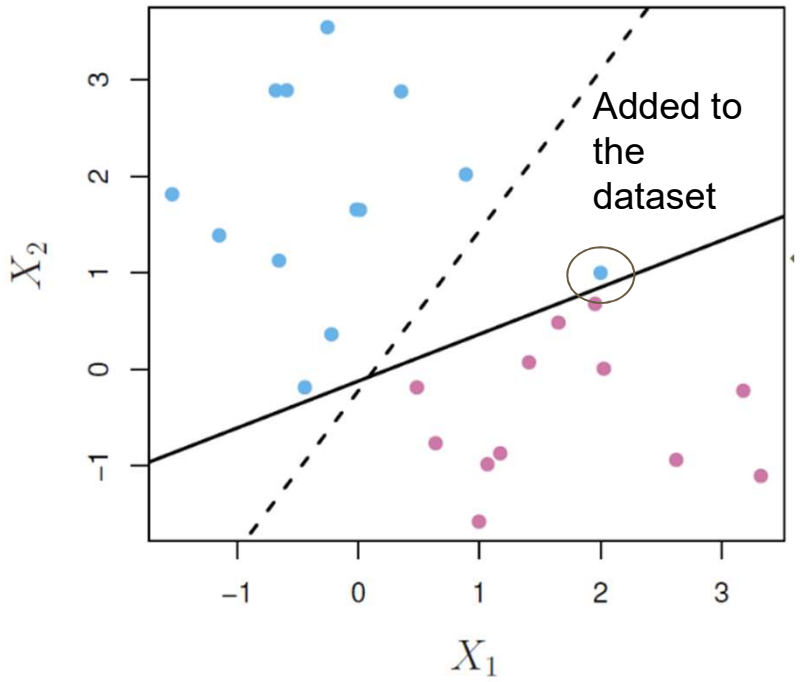
To minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias.

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2 + \mathrm{Var}(\epsilon).$$

Variance

Variance refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. Since the training data are used to fit the statistical learning method, different training data sets will result in a different $\hat{f}$. But ideally the estimate for f should not vary too much between training sets.

However, if a method has high variance then small changes in the training data can result in large changes in $\hat{f}$. In general, more flexible statistical methods have higher variance.

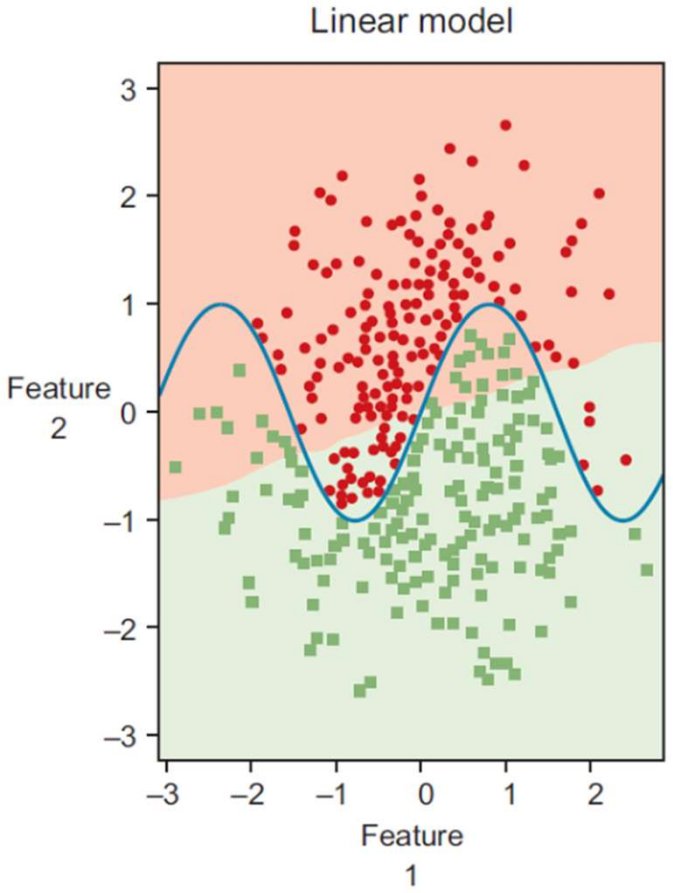


Bias

Bias refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

For example, linear regression assumes that there is a linear relationship between Y and X1,X2, . . . , Xp.

It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f.



Classification with a Logistic Regressor

http://localhost:8888/notebooks/real-world-machine-learning/Chapter%203%20-%20Modeling%20and%20prediction.ipynb

We will try classification using Logistic Regression on the Titanic Data

The data has to be imputed and categories binarized:

```
In [12]: def prepare_data(data):
             """Takes a dataframe of raw data and returns ML model features
             """

             # Initially, we build a model only on the available numerical values
             features = data.drop(["PassengerId", "Survived", "Fare", "Name", "Sex", "Ticket", "Cabin", "Embarked"], axis=1)

             # Setting missing age values to -1
             features["Age"] = data["Age"].fillna(-1)

             # Adding the sqrt of the fare feature
             features["sqrt_Fare"] = sqrt(data["Fare"])

             # Adding gender categorical value
             features = features.join( cat_to_num(data['Sex']) )

             # Adding Embarked categorical value
             features = features.join( cat_to_num(data['Embarked']) )

             return features
```

			Parch	sqrt_Fare	Gender = female	Gender = male	Embarked = C	Embarked = Q	Embarked = S
			0	2.692582	0	1	0	0	1
1	38	1	0	8.442944	1	0	1	0	0
3	26	0	0	2.815138	1	0	0	0	1
1	35	1	0	7.286975	1	0	0	0	1
3	35	0	0	2.837252	0	1	0	0	1

Creating LR model with SciKit

```
from sklearn.linear_model import LogisticRegression as Model

def train(features, target):
  model = Model()
  model.fit(features, target)
  return model

def predict(model, new_features):
  preds = model.predict(new_features)
  return preds

# Assume Titanic data is loaded into titanic_feats,
# titanic_target and titanic_test
model = train(titanic_feats, titanic_target)
predictions = predict(model, titanic_test)
```

Imports the logistic regression algorithm

Fits the logistic regression algorithm using features and target data

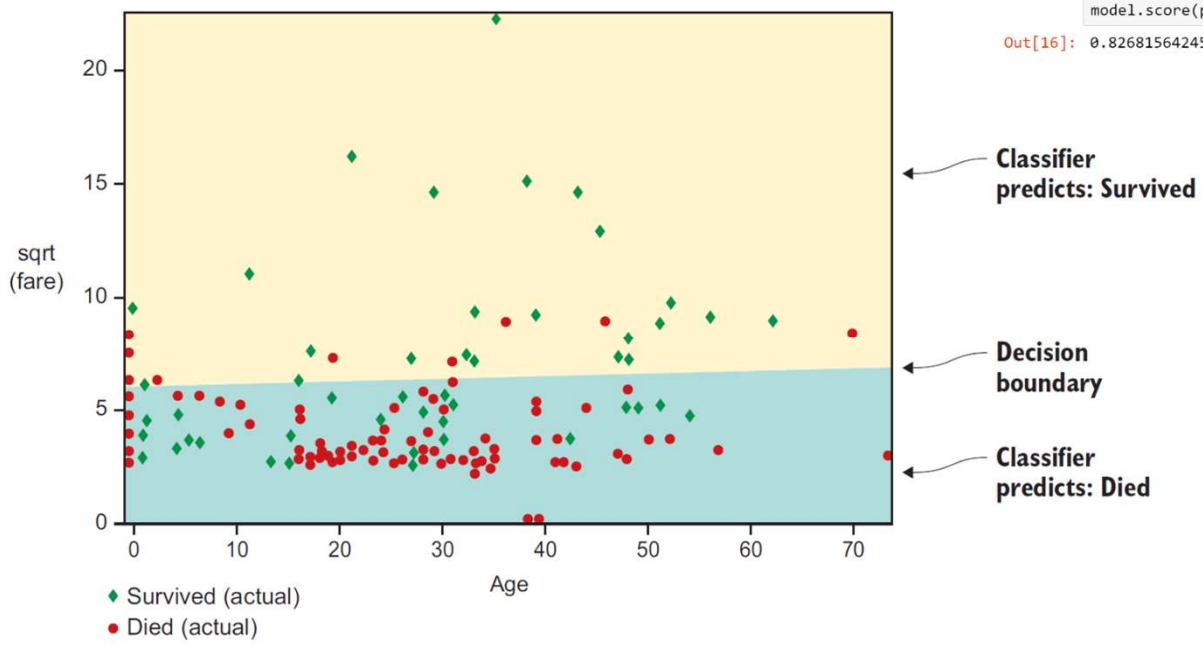
Makes predictions on a new set of features using the model

Returns the model built by the algorithm

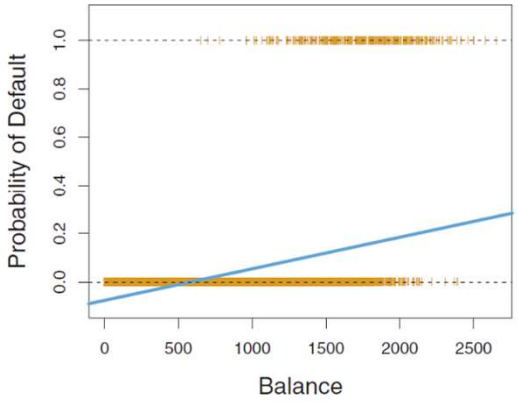
Returns predictions (0 or 1)

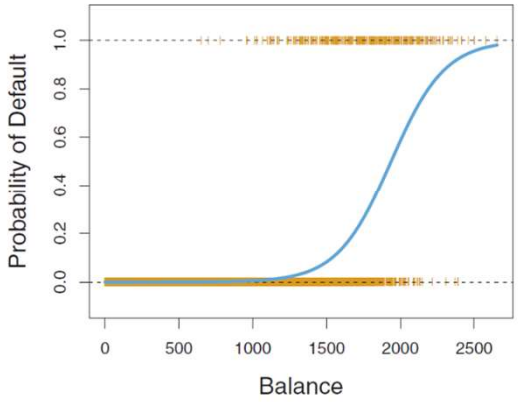
Prediction Results

```
In [15]: # Make predictions
         model.predict(prepare_data(data_test))

Out[15]: array([0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
                0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1,
                1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
                0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
                0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
                0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
                0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0]], dtype=int64)

In [16]: # The accuracy of the model on the test data
         # (this will be introduced in more details in chapter 4)
         model.score(prepare_data(data_test), data_test["Survived"])

Out[16]: 0.82681564245810057
```



Logistic Regression: definition





$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}.$$

LR uses a sigmoid relation to describe the probability of the outcome. Using LR as a classifier, we have as an output a Probability of outcome rather than the outcome itself: we must model p(X) using a function that gives outputs between 0 and 1 for all values of X.

LR: Odds and Logit

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}.$$

The quantity p(X)/[1−p(X)] is called the odds, and can take on any value odds between 0 and ∞. Values of the odds close to 0 and ∞ indicate very low and very high probabilities of survival, respectively. For example, on average 1 in 5 people with an odds of 1/4 will survive, since p(X) = 0.2 implies an odds of 0.2 1−0.2 = 1/4.

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

By taking the log, we define the logit which is linear in X: increasing X by one unit changes the log odds by β1, or equivalently it multiplies the odds by e^β1. Because the relationship between p(X) and X is not linear, β1 does not correspond to the change in p(X) associated with a one-unit increase in X. The amount that p(X) changes due to a one-unit change in X will depend on the current value of X.

Estimating coefficients in LR

The coefficients β0 and β1 in are unknown, and must be estimated based on the available training data, using *maximum likelihood* to fit the model.

We seek estimates for β0 and β1 such that the predicted probability ˆp(xi) of survival for each individual, corresponds as closely as possible to the individual's observed survival status. In other words, we try to find ˆ β0 and ˆ β1 such that plugging these estimates into the model for p(X), yields a number close to one for all individuals who survived, and a number close to zero for all individuals who did not.

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})).$$

Making predictions with LR

Once the coefficients have been estimated, it is a simple matter to compute the probability of survival, for example related to the ticket fare:

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}} = \frac{e^{-10.6513+0.0055\times 1,000}}{1+e^{-10.6513+0.0055\times 1,000}} = 0.00576,$$

With double the fare, the probability of survival dramatically increases (0.586 or 58.6%)

Beyond linear classifiers

Logistic regression is a linear algorithm: the decision boundary is a straight line.

Data might not be well separated by a straight line, so for such datasets you should use a nonlinear algorithm. But nonlinear algorithms are typically more demanding computationally and don't scale well to large datasets.

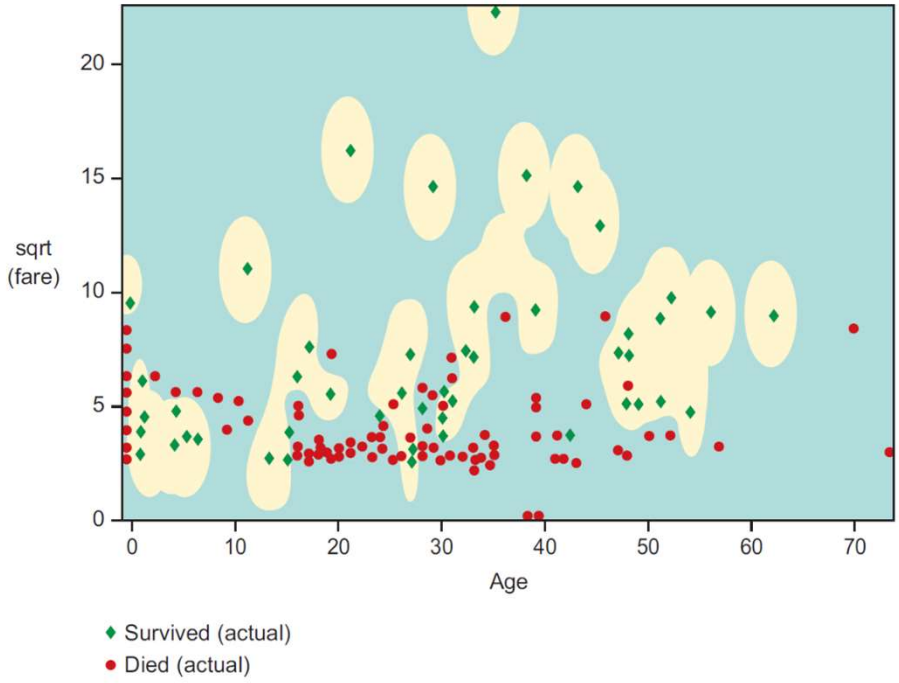
A typical non-linear classifier is SVM (Support Vector Machines)

Changing to Support Vector Machines

By just changing the used model in the previous example, we can apply a non-linear classifier such as SVM.

Non-linear model with Support Vector Machines

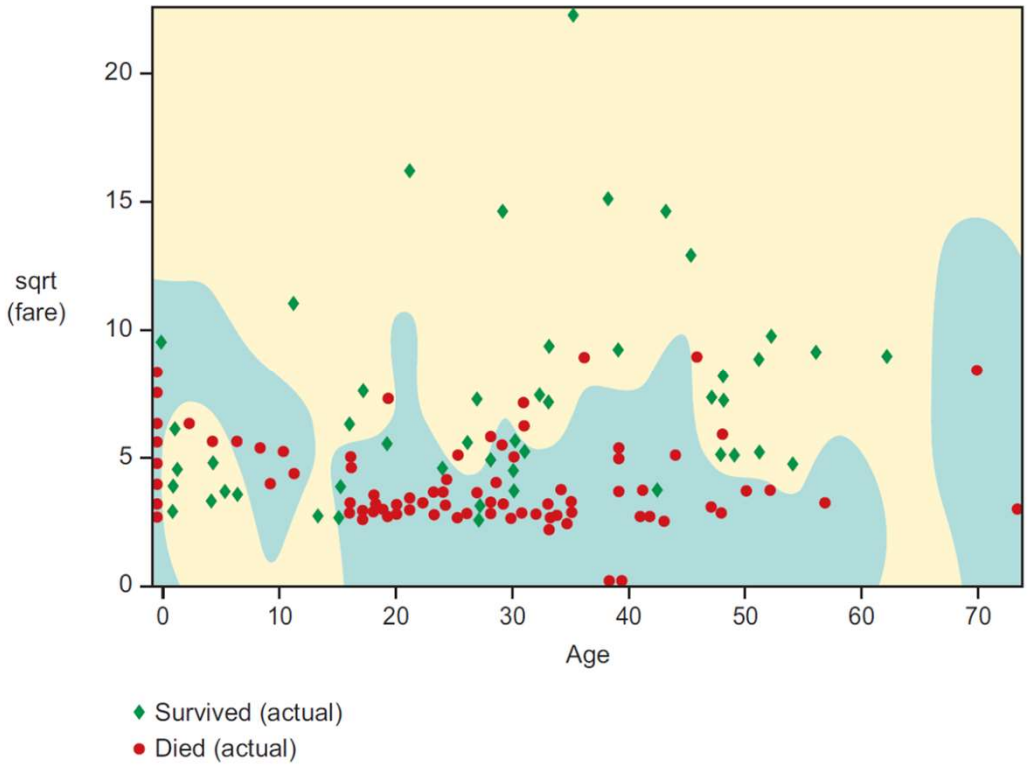
```
In [62]: from sklearn.svm import SVC
         model = SVC()
         model.fit(features, data_train["Survived"])

Out[62]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)

In [63]: model.score(prepare_data(data_test), data_test["Survived"])

Out[63]: 0.86033519553072624
```



Overfitting

The algorithm is capable of fitting well to the data, almost at the single-record level, and you risk losing the ability to make good predictions on new data that wasn't included in the training set; the more complex you allow the model to become, the higher the risk of overfitting.

Lowering gamma on SVM with RBF (Radial Basis Function) kernel improves the situation.



Support Vector Machines: Definition

Here we approach the two-class classification problem in a direct way:

> *We try and find a plane that separates the classes in feature space.*

If we cannot, we get creative in two ways:

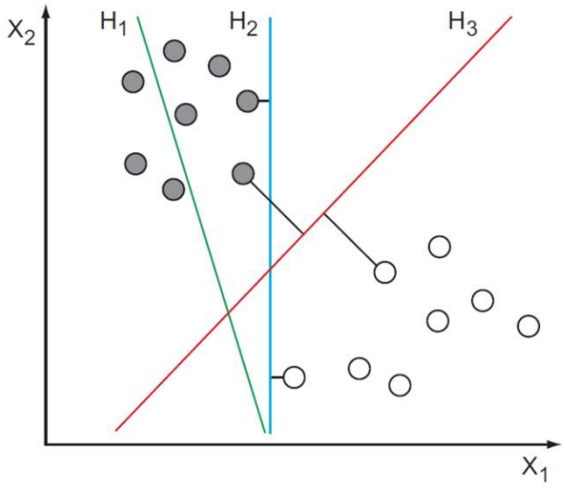
- We soften what we mean by "separates", and
- We enrich and enlarge the feature space so that separation is possible.

Hyperplanes

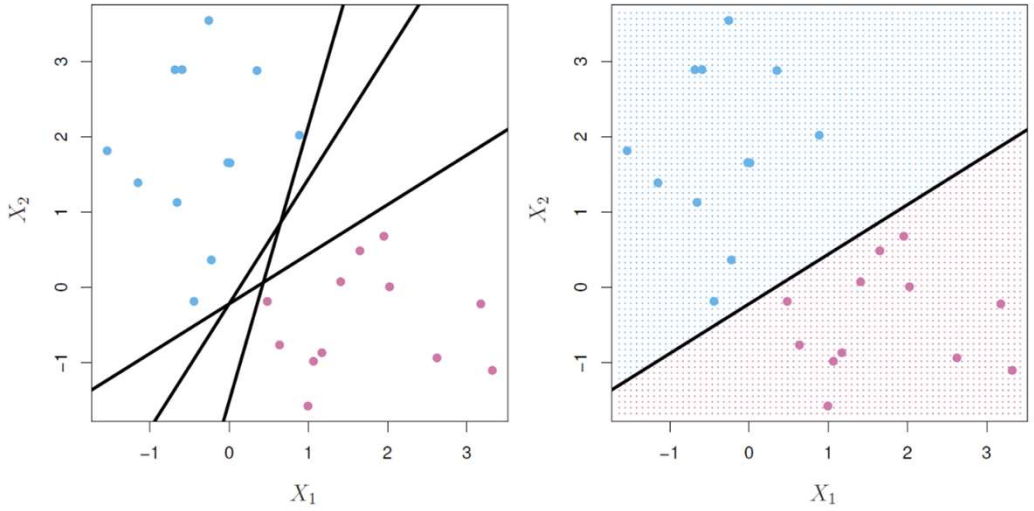
- A hyperplane in p dimensions is a flat affine subspace of dimension $p-1$.
- In general the equation for a hyperplane has the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + | \ldots + \beta_p X_p = 0$$

- In $p = 2$ dimensions a hyperplane is a line.
- If $\beta_0 = 0$, the hyperplane goes through the origin, otherwise not.
- The vector $\beta = (\beta_1, \beta_2, \cdots, \beta_p)$ is called the normal vector — it points in a direction orthogonal to the surface of a hyperplane.



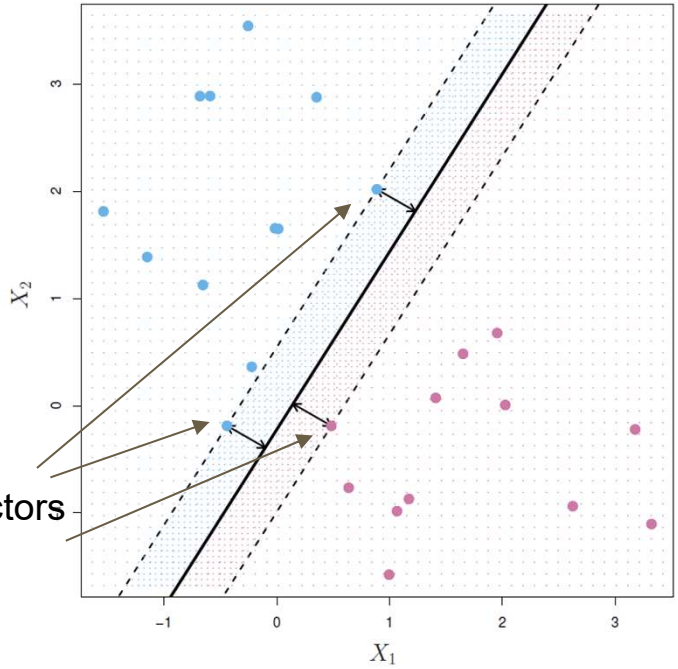
Separating Hyperplanes



- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = +1$ for blue, say, and $Y_i = -1$ for mauve, then if $Y_i \cdot f(X_i) > 0$ for all i, $f(X) = 0$ defines a *separating hyperplane*.

Maximal Margin Classifier

Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.



Constrained optimization problem

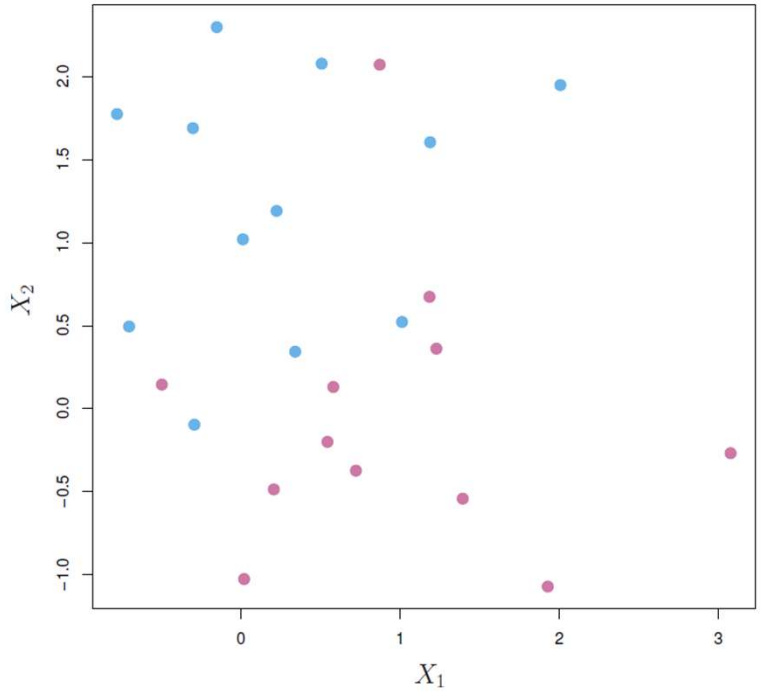
$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M$$

$$\text{for all } i = 1, \ldots, N.$$

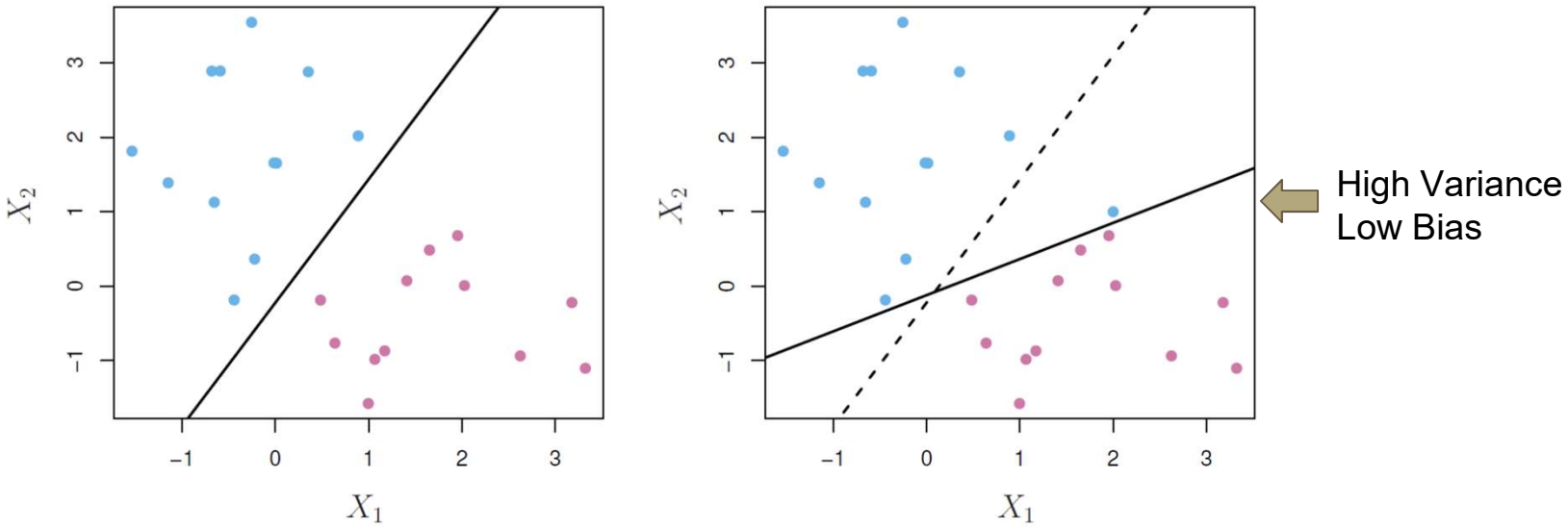
Non-separable Data



The data on the left are not separable by a linear boundary.

This is often the case, unless $N < p$.

Noisy Data



Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

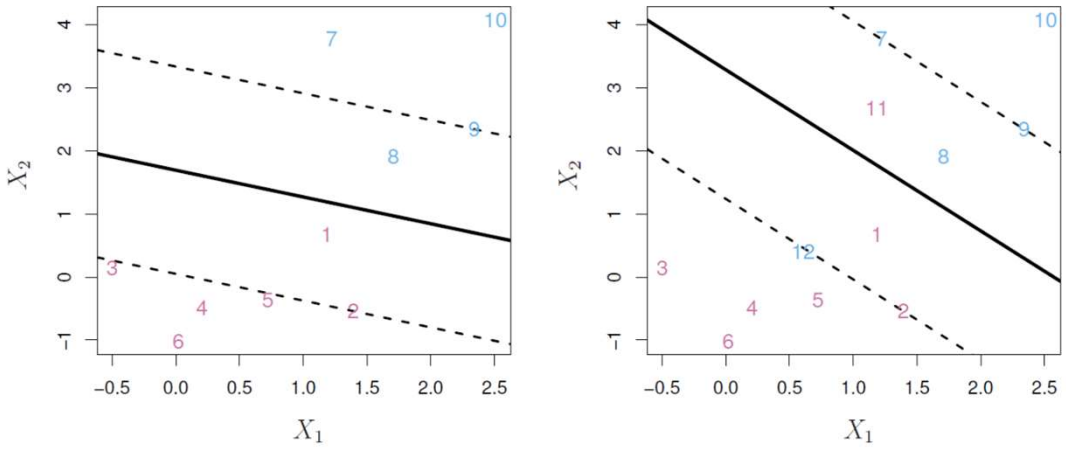
The *support vector classifier* maximizes a *soft* margin.

Soft margin

The support vector classifier, sometimes called a soft margin classifier, does exactly this.

Rather than seeking the largest possible margin so that every observation is not only on the correct side of the hyperplane but also on the correct side of the margin, we instead allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.

Support Vector Classifier



Left: A support vector classifier was fit to a small data set. The hyperplane is shown as a solid line and the margins are shown as dashed lines.Purple observations: Observations 3, 4, 5, and 6 are on the correct side of the margin, observation 2 is on the margin, and observation 1 is on the wrong side of the margin. Blue observations: Observations 7 and 10 are on the correct side of the margin, observation 9 is on the margin, and observation 8 is on the wrong side of the margin. No observations are on the wrong side of the hyperplane.

Right: Same as left panel with two additional points, 11 and 12. These two observation are on the wrong side of the hyperplane and the wrong side of the margin.

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \; M \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \; \sum_{i=1}^{n} \epsilon_i \leq C,$$

Using slack

C is a non negative tuning parameter. As in the Maximal Margin Classifier, M is the width of the margin; we seek to make this quantity as large as possible. In the Support Vector Classifier, the epsilons are slack variables that allow individual observations to be on the wrong side of the margin or the hyperplane.

If $\epsilon i = 0$ then the ith observation is on the correct side of the margin

If $\epsilon i > 0$ then the ith observation is on the wrong side of the margin (margin violation)

If $\epsilon i > 1$ then the ith observation is on the wrong side of the hyperplane

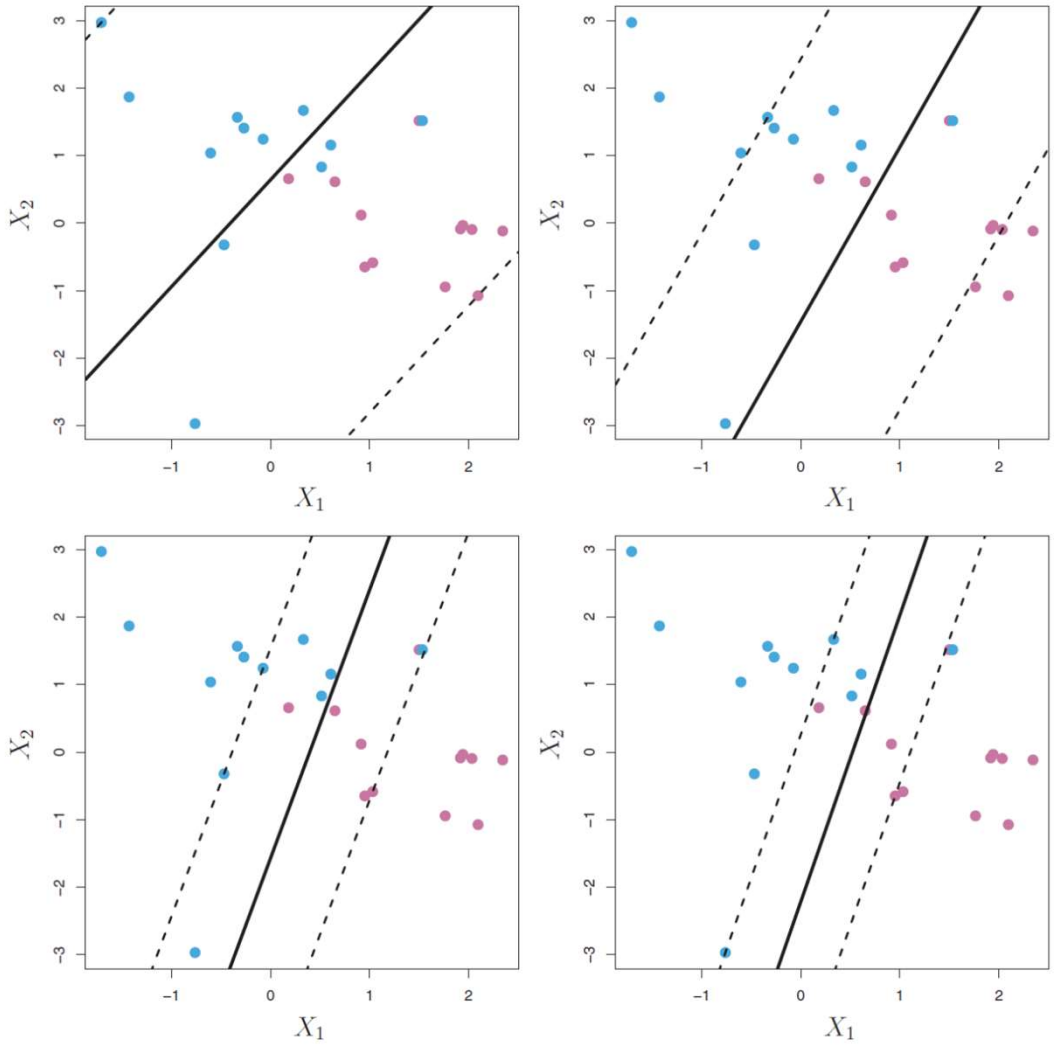
“C” parameter as slack budget

C bounds the sum of the slacks, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate

If C = 0 then there is no budget for violations to the margin and all the slacks have to 0

For C > 0 no more than C observations can be on the wrong side of the hyperplane (each slack = 1)

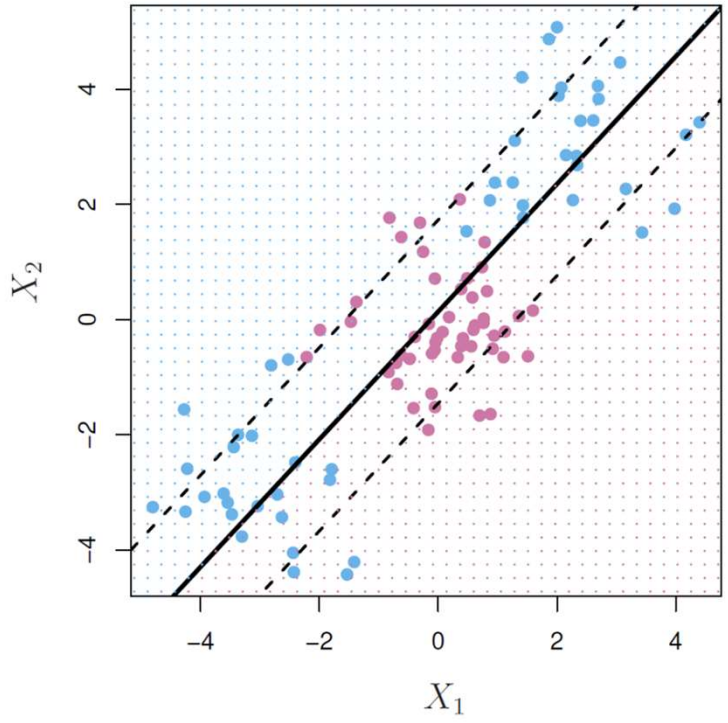
As the budget C increases, we become more tolerant of violations to the margin, and so the margin will widen. Conversely, as C decreases, we become less tolerant of violations to the margin and so the margin narrows.



Top left: high C

Bottom right: low C

Datasets in which SVC imposes linear bias



Sometime a linear boundary simply won't work, no matter what value of C.

The example on the left is such a case.

What to do?

Feature Expansion

- Enlarge the space of features by including transformations; e.g. X_1^2, X_1^3, X_1X_2, $X_1X_2^2$,.... Hence go from a p-dimensional space to a $M > p$ dimensional space.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Example: Suppose we use $(X_1,\ X_2,\ X_1^2,\ X_2^2,\ X_1X_2)$ instead of just (X_1, X_2). Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = 0$$

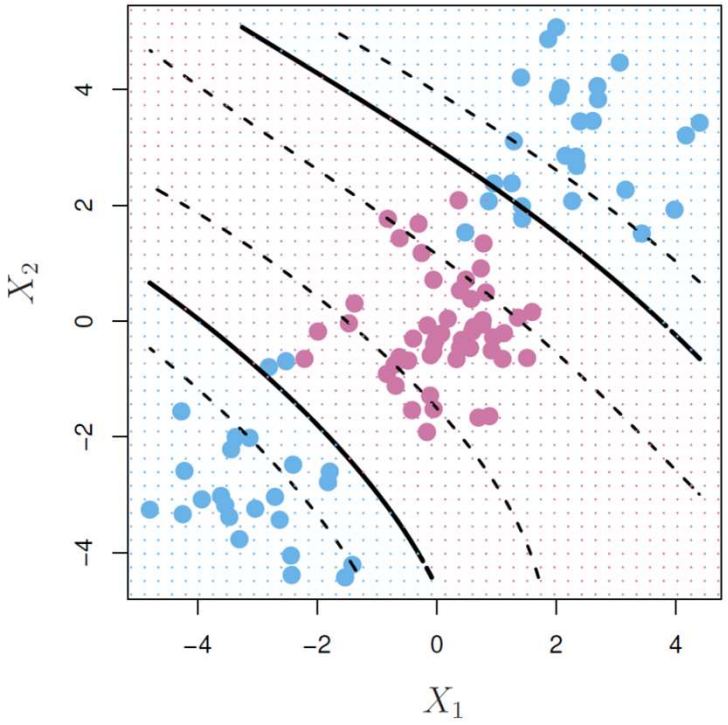
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Polynomial boundary

Here we use a basis expansion of cubic polynomials

From 2 variables to 9

The support-vector classifier in the enlarged space solves the problem in the lower-dimensional space



$$\beta_0+\beta_1X_1+\beta_2X_2+\beta_3X_1^2+\beta_4X_2^2+\beta_5X_1X_2+\beta_6X_1^3+\beta_7X_2^3+\beta_8X_1X_2^2+\beta_9X_1^2X_2=0$$

Inner Product

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \quad \text{— } \textit{inner product between vectors}$$

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \quad \text{— } n \textit{ parameters}$$

- To estimate the parameters $\alpha_1, \ldots, \alpha_n$ and β_0, all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.

It turns out that most of the $\hat{\alpha}_i$ can be zero:

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle$$

$\mathcal{S}$ is the *support set* of indices i such that $\hat{\alpha}_i > 0$.

Using Kernels

- Some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d$$

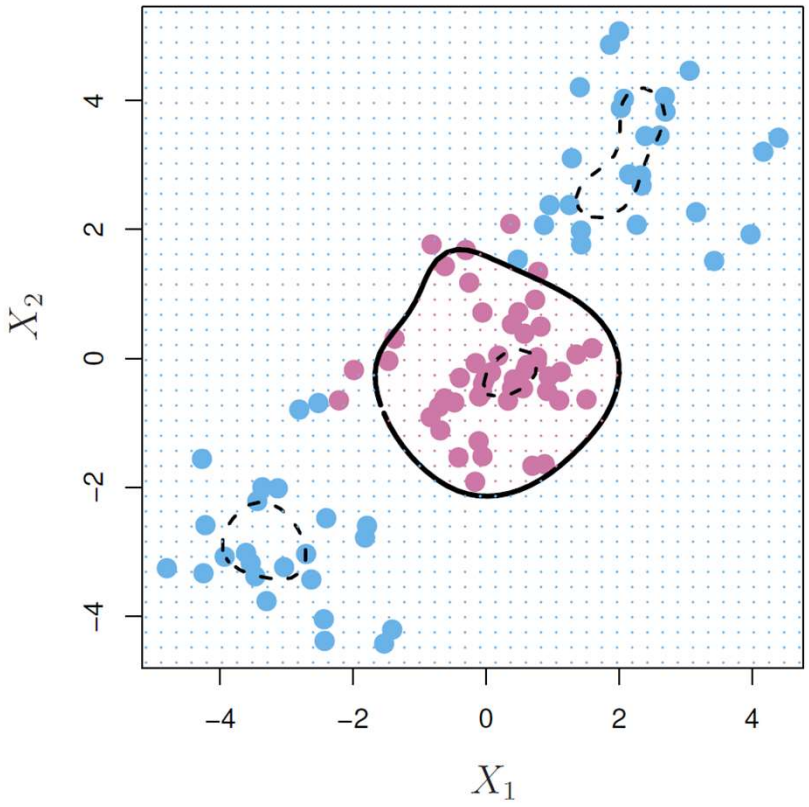
computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

- The solution has the form

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i).$$

Radial Kernel (RBF)

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$$



$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(x, x_i)$$

Implicit feature space; very high dimensional.

Controls variance by squashing down most dimensions severely

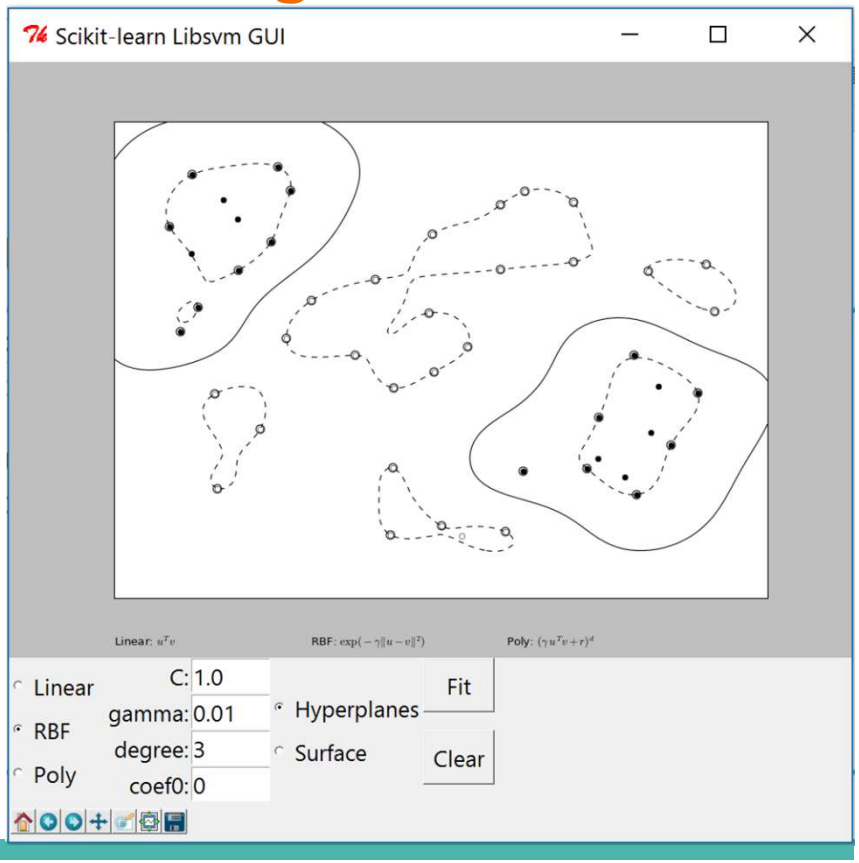
https://scikit-learn.org/stable/auto_examples/applications/svm_gui.html

Playing around with parameter tuning

SVMgui.py in scikit allows to play interactively with three different variations of SVM: Linear, with Kernel and Polynomial.

By arranging parameters and variation, it is possible to study the effect of parameters to avoid overfitting on the training set.

https://cs.stanford.edu/~karpathy/svmjs/demo/ù
https://vimeo.com/18308519



Multiple-class Classifiers

Classifiers with more than two categories are often necessary in real-world problems, such as handwritten digits recognition.

KNN (K-Nearest Neighbours) is a natively multi-class non-parametric classification/regression algorithm which is simple and widely used.

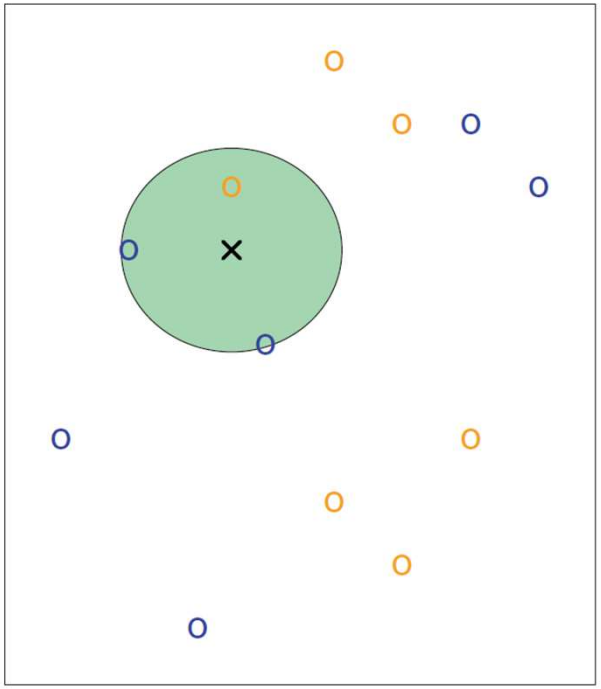
KNN Classification (K-Nearest Neighbors)

Given a positive integer K and a test observation x0, the KNN classifier first identifies the neighboring K points in the training data that are closest to x0, represented by N0.

It then estimates the conditional probability for class j as the fraction of points in N0 whose response values equal j:

$$\Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j).$$

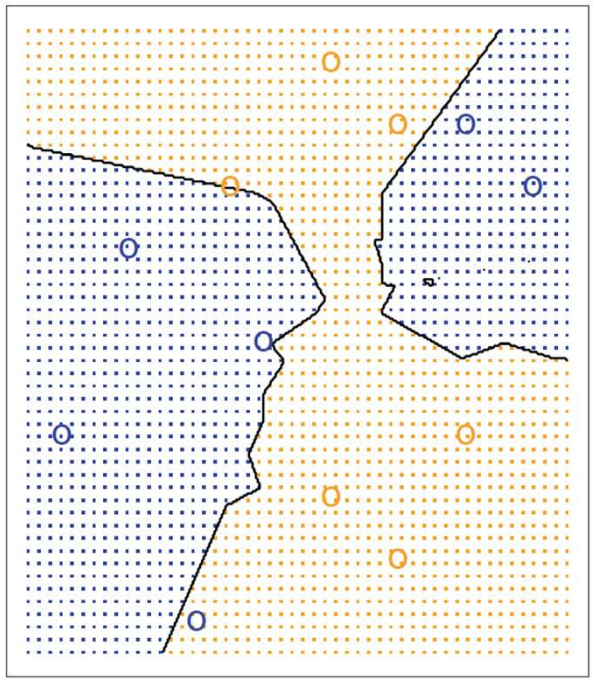
KNN Classification - Working principle



We have plotted a small training data set consisting of six blue and six orange observations. Our goal is to make a prediction for the point labeled by the black cross. Suppose that we choose K = 3. Then KNN will first identify the three observations that are closest to the cross. This neighborhood is shown as a circle. It consists of two blue points and one orange point, resulting in estimated probabilities of 2/3 for the blue class and 1/3 for the orange class.

Hence KNN will predict that the black cross belongs to the blue class.

KNN - Effect of K on the Decision Boundary

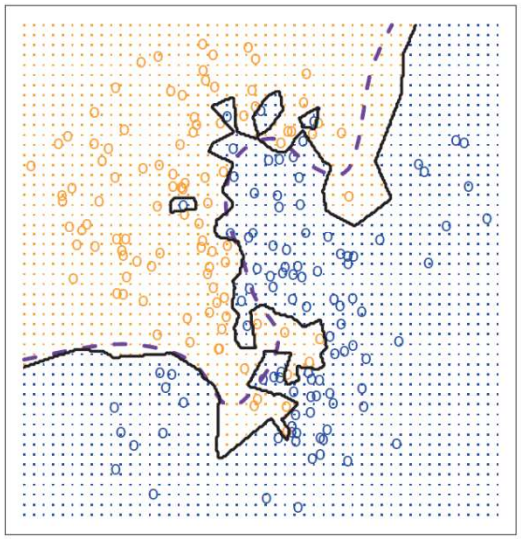


We have applied the KNN approach with K = 3 at all of the possible values for X1 and X2, and have drawn in the corresponding KNN decision boundary.

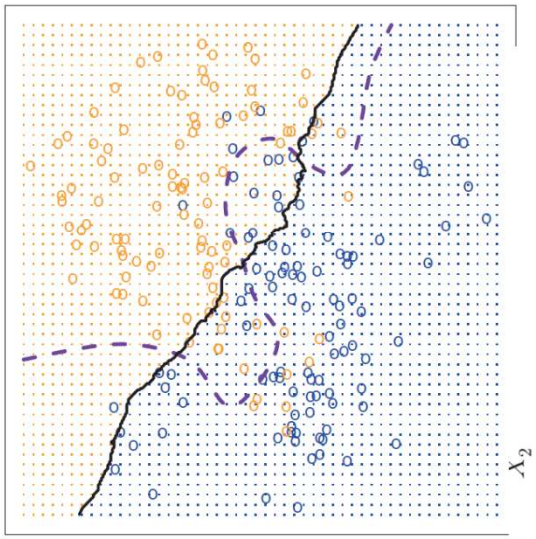
With a larger dataset, the value of K is important to find the correct balance between overfitting and predictive inflexibility.

Examples: K=1 overfits; K=100 is inflexible; K = 10 is just right

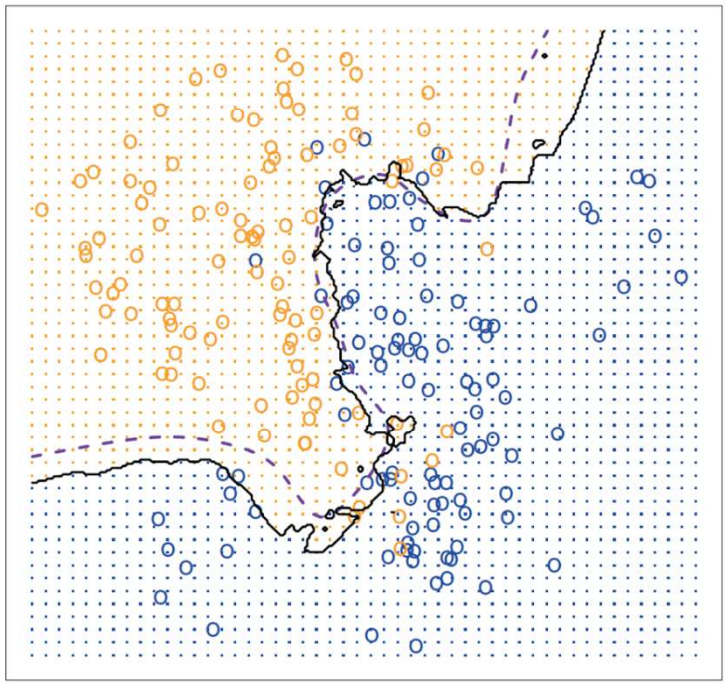
KNN: K=1



KNN: K=100



KNN: K=10



Lab: Knn with MNIST digits

Classification with multiple classes: hand-written digits

We use the popular non-linear multi-class K-nearest neighbor algorithm to predict hand-written digits from the MNIST dataset.

In [34]:
```
mnist = pandas.read_csv("data/mnist_small.csv")
mnist_train = mnist[:int(0.8*len(mnist))]
mnist_test = mnist[int(0.8*len(mnist)):]
mnist_test[:2]
```

Out[34]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	p
800	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
801	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

2 rows × 785 columns

In [66]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(mnist_train.drop("label", axis=1), mnist_train['label'])
```

Out[66]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=10, p=2, weights='uniform')
```

In [71]:
```
preds = knn.predict_proba(mnist_test.drop("label", axis=1))
pandas.DataFrame(preds[:5], index=["Digit %d"%(i+1) for i in range(5)])
```

Out[71]:

	0	1	2	3	4	5	6	7	8	9
Digit 1	0.0	0	0	0.0	0.0	1	0.0	0.0	0.0	0.0
Digit 2	0.0	0	0	1.0	0.0	0	0.0	0.0	0.0	0.0
Digit 3	0.3	0	0	0.0	0.0	0	0.6	0.0	0.1	0.0
Digit 4	0.0	0	0	0.1	0.0	0	0.0	0.5	0.0	0.4
Digit 5	0.0	0	0	0.0	0.7	0	0.0	0.0	0.0	0.3

In [73]:
```
knn.score(mnist_test.drop("label", axis=1), mnist_test['label'])
```

Out[73]:
```
0.8199999999999995
```

Regression: predicting numerical values

Not every machine-learning problem is about putting records into classes. Sometimes the target variable takes on numerical values—for example, when predicting a dollar values in a financial model.

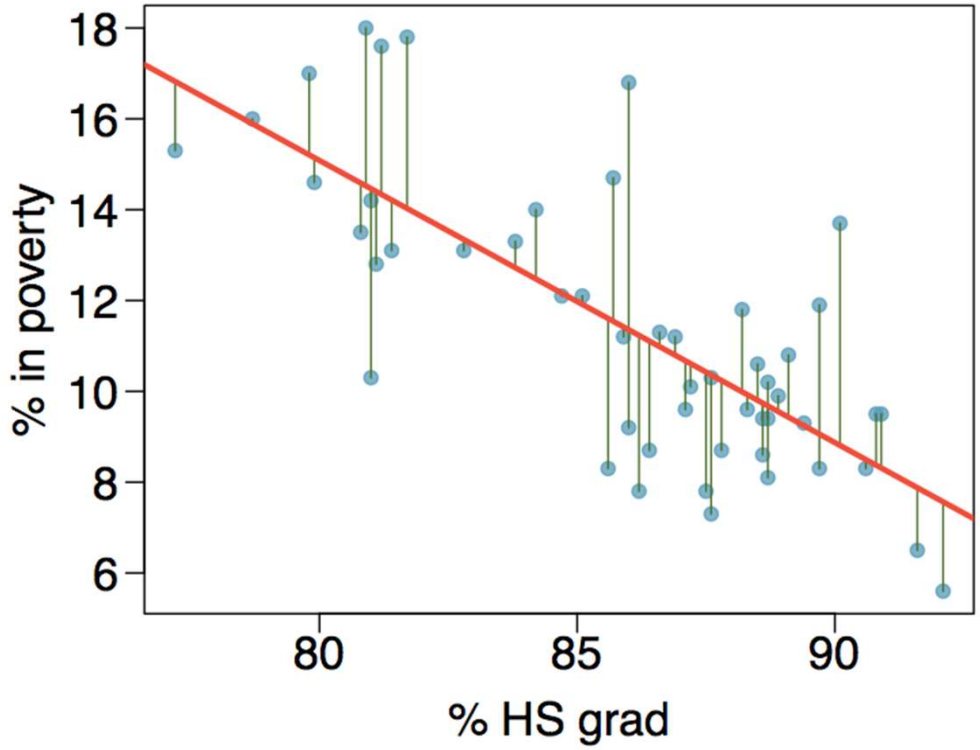
We call the act of predicting numerical values **regression**, and the model itself a regressor.

Linear regression is arguably the simplest and most widely used algorithm for building regression models. The main strengths are linear scalability and a high level of interpretability.

This algorithm plots the dataset records as points, with the target variable on the y-axis, and fits a straight line (or plane, in the case of two or more features) to these points.

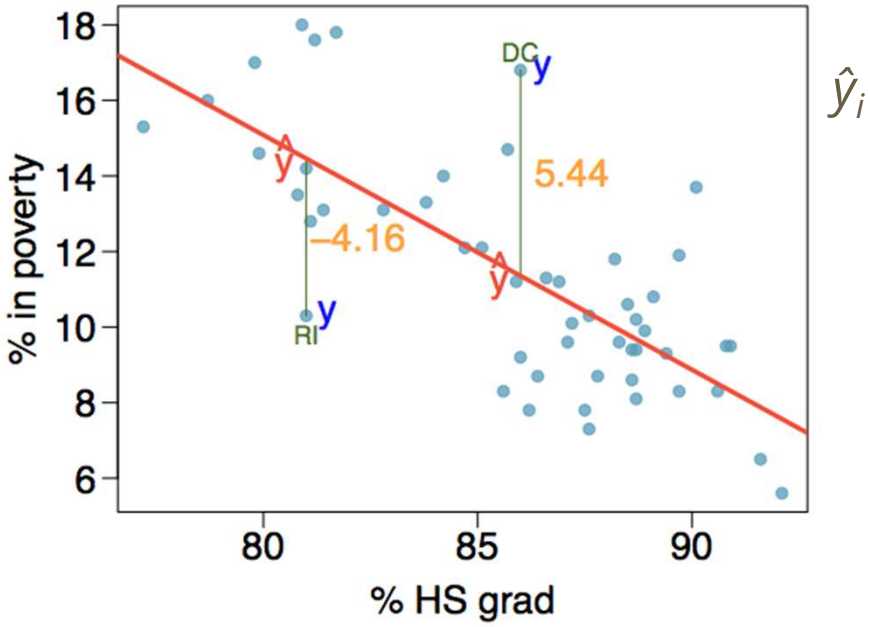
Residuals

Residuals are the leftovers from the model fit: Data = Fit + Residual



Residuals (cont.)

Residual is the difference between the observed (y_i) and predicted $\hat{y}_i$.



- % living in poverty in DC is 5.44% more than predicted.
- % living in poverty in RI is 4.16% less than predicted.

A measure for the best line

- We want a line that has small residuals
  1. Option 1: Minimize the sum of magnitudes (absolute values) of residuals

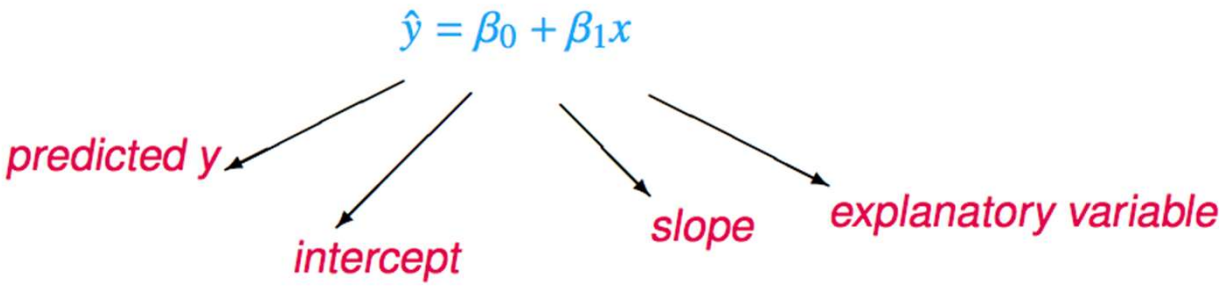
     $$|e_1| + |e_2| + \ldots + |e_n|$$

  2. Option 2: Minimize the sum of squared residuals -- *least squares*

     $$e_1^2 + e_2^2 + \ldots + e_n^2$$

- Why least squares?
  1. Most commonly used
  2. Easier to compute by hand and using software
  3. In many applications, a residual twice as large as another is usually more than twice as bad

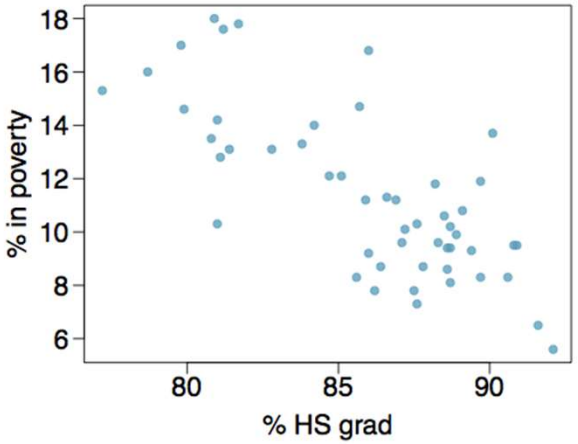
The least squares line



Notation:

- Intercept:
  - Parameter: β_0
  - Point estimate: b_0
- Slope:
  - Parameter: β_1
  - Point estimate: b_1

Given...



	% HS grad (x)	% in poverty (y)
mean	$\bar{x} = 86.01$	$\bar{y} = 11.35$
sd	$s_x = 3.73$	$s_y = 3.1$
	correlation	$R = -0.75$

Slope

The slope of the regression can be calculated as

$$b_1 = \frac{s_y}{s_x} R$$

In context...

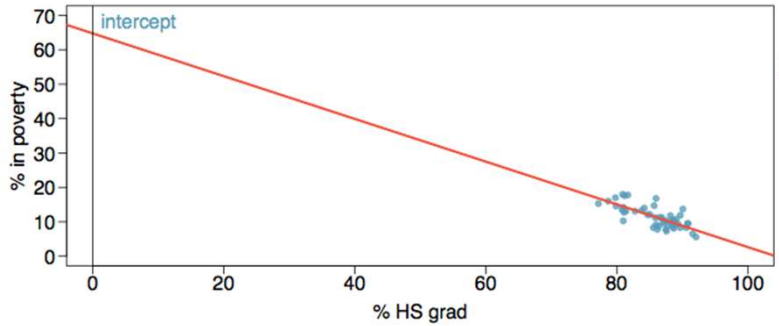
$$b_1 = \frac{3.1}{3.73} \times -0.75 = -0.62$$

Interpretation

For each additional % point in HS graduate rate, we would expect the % living in poverty to be lower on average by 0.62% points.

Intercept

The intercept is where the regression line intersects the y-axis. The calculation of the intercept uses the fact the a regression line always passes through $(\bar{x}, \bar{y})$.



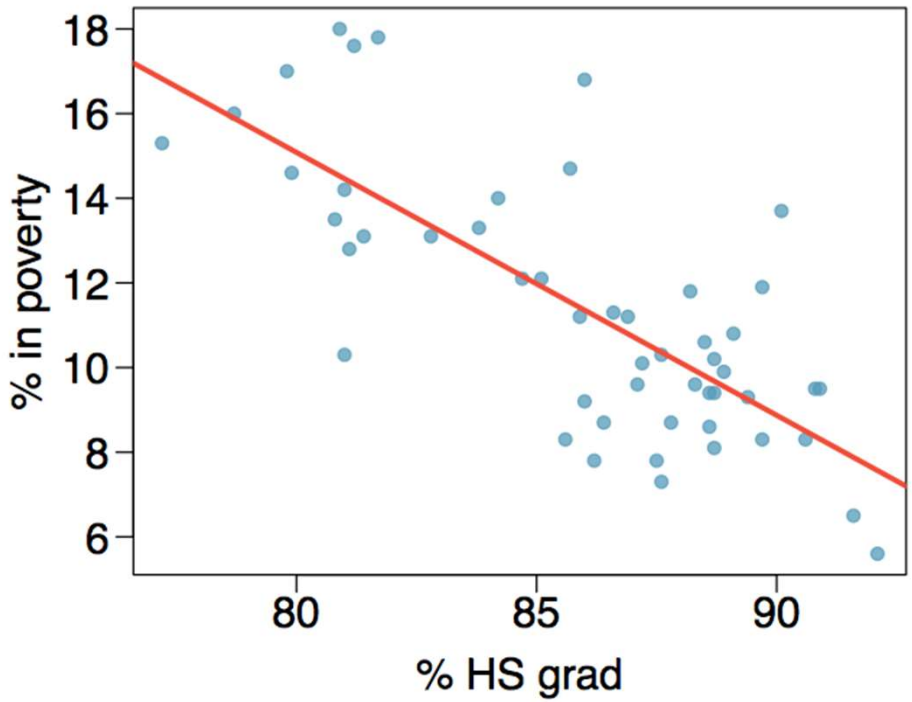
$\bar{y} - b_1 \bar{x}$

$b_0 = 11.35 - (-0.62) \times 86.01$

$= 64.68$

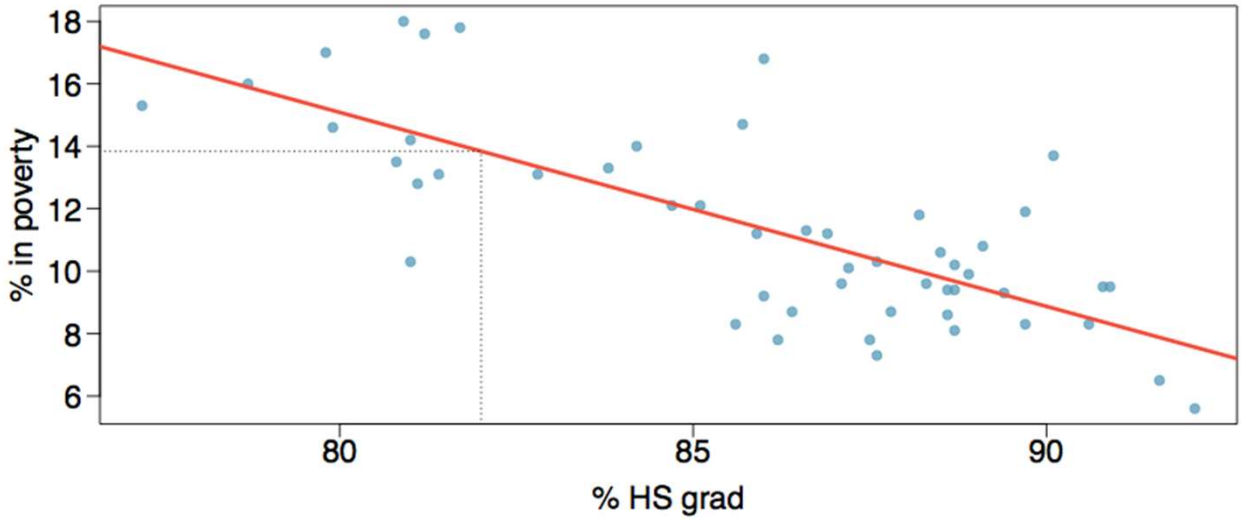
Regression line

$$\widehat{\%\ in\ poverty} = 64.68 - 0.62\ \%\ HS\ grad$$



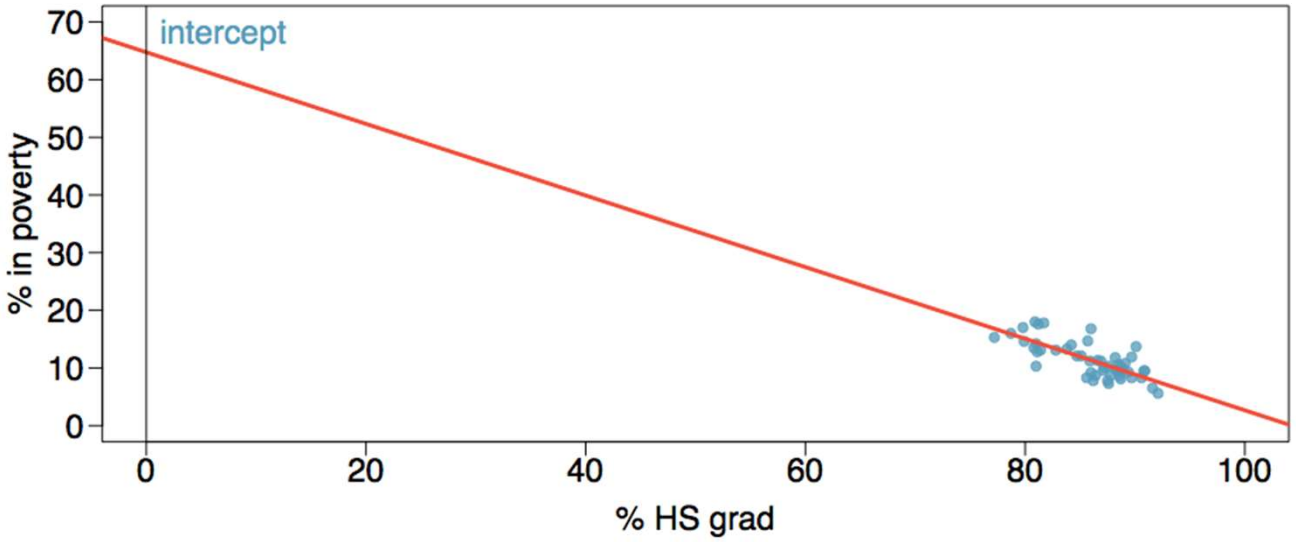
Prediction

- Using the linear model to predict the value of the response variable for a given value of the explanatory variable is called *prediction*, simply by plugging in the value of x in the linear model equation.
- There will be some uncertainty associated with the predicted value.



Extrapolation

- Applying a model estimate to values outside of the realm of the original data is called *extrapolation*.
- Sometimes the intercept might be an extrapolation.

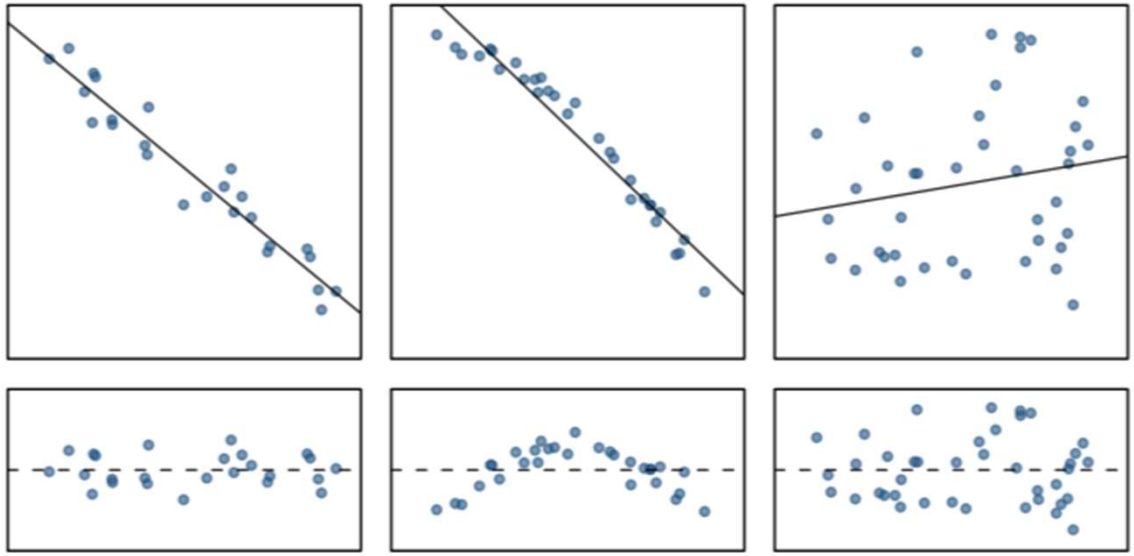


Conditions for the least squares line

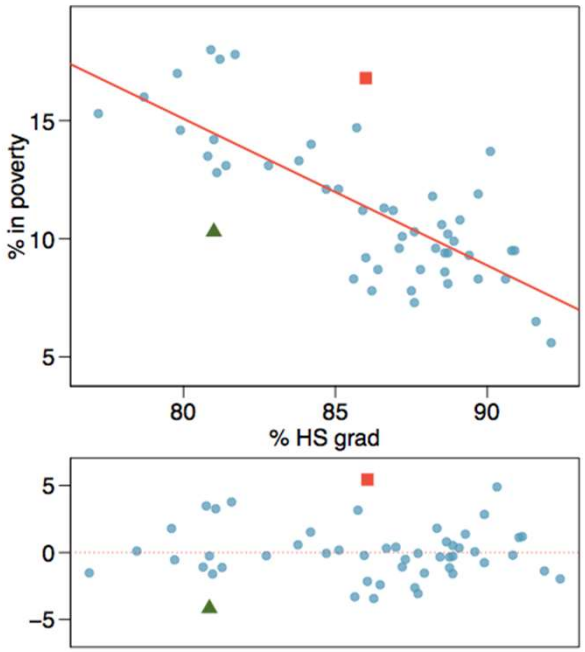
1. Linearity
2. Nearly normal residuals
3. Constant variability

Conditions: (1) Linearity

- The relationship between the explanatory and the response variable should be linear.
- Check using a scatterplot of the data, or a *residuals plot*.



Anatomy of a residuals plot



▲ *RI:*

$$\%\ HS\ grad = 81 \qquad \%\ in\ poverty = 10.3$$

$$\widehat{\%\ in\ poverty} = 64.68 - 0.62 * 81 = 14.46$$

$$e = \%\ in\ poverty - \widehat{\%\ in\ poverty}$$

$$= 10.3 - 14.46 = -4.16$$

■ *DC:*

$$\%\ HS\ grad = 86 \qquad \%\ in\ poverty = 16.8$$

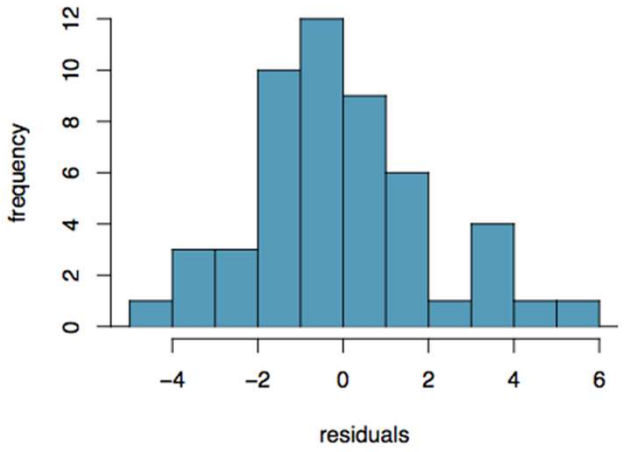
$$\widehat{\%\ in\ poverty} = 64.68 - 0.62 * 86 = 11.36$$

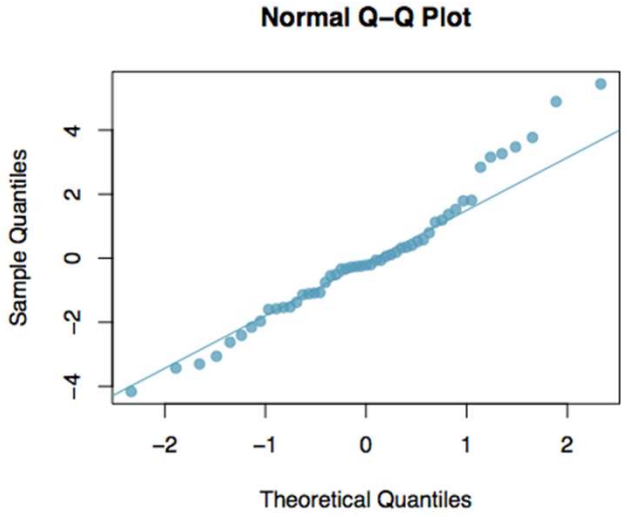
$$e = \%\ in\ poverty - \widehat{\%\ in\ poverty}$$

$$= 16.8 - 11.36 = 5.44$$

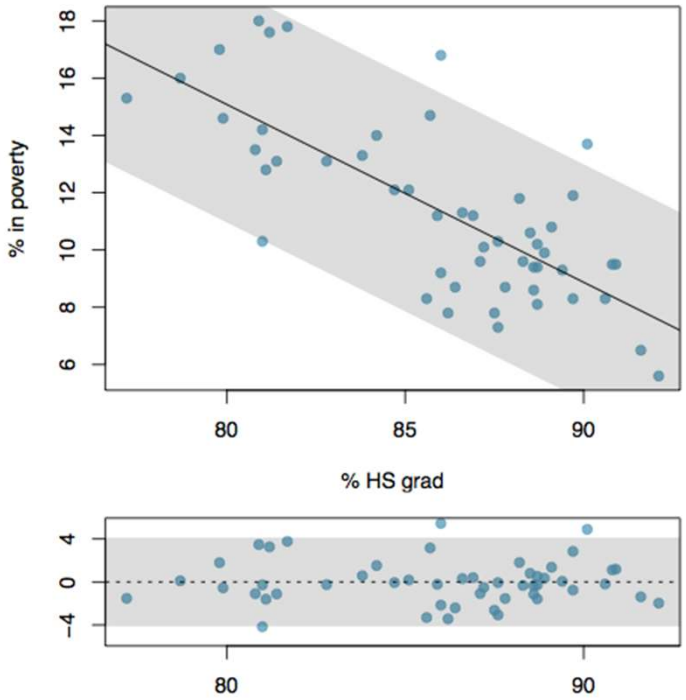
Conditions: (2) Nearly normal residuals

- The residuals should be nearly normal.
- This condition may not be satisfied when there are unusual observations that don't follow the trend of the rest of the data.
- Check using a histogram or normal probability plot of residuals.





Conditions: (3) Constant variability



- The variability of points around the least squares line should be roughly constant.
- This implies that the variability of residuals around the 0 line should be roughly constant as well.
- Also called *homoscedasticity*.
- Check using a histogram or normal probability plot of residuals.

Correlation Matrix and Correlation Coefficient (R)

- The correlation matrix is a square matrix that contains the **Pearson product-moment correlation coefficient** (often abbreviated as **Pearson's r**), which measure the linear dependence between pairs of features. The correlation coefficients are in the range -1 to 1. Two features have a perfect positive correlation if r=1, and perfect negative if r=-1. No correlation if r = 0.

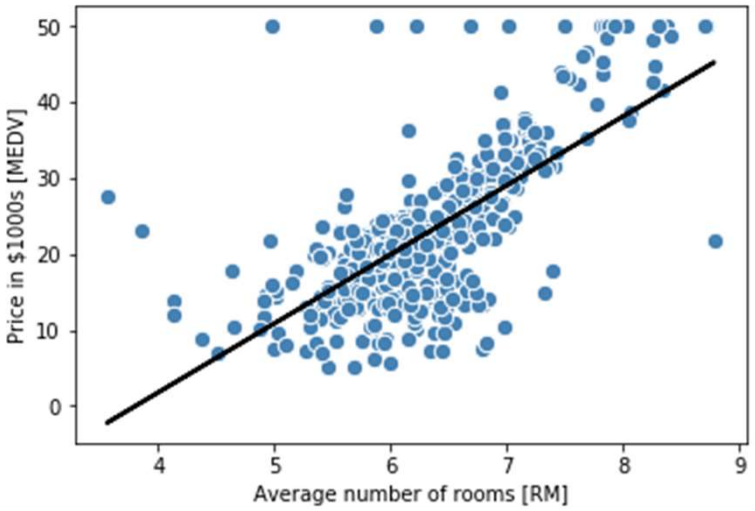
$$r = \frac{\sum_{i=1}^{n}\left[\left(x^{(i)} - \mu_x\right)\left(y^{(i)} - \mu_y\right)\right]}{\sqrt{\sum_{i=1}^{n}\left(x^{(i)} - \mu_x\right)^2}\sqrt{\sum_{i=1}^{n}\left(y^{(i)} - \mu_y\right)^2}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

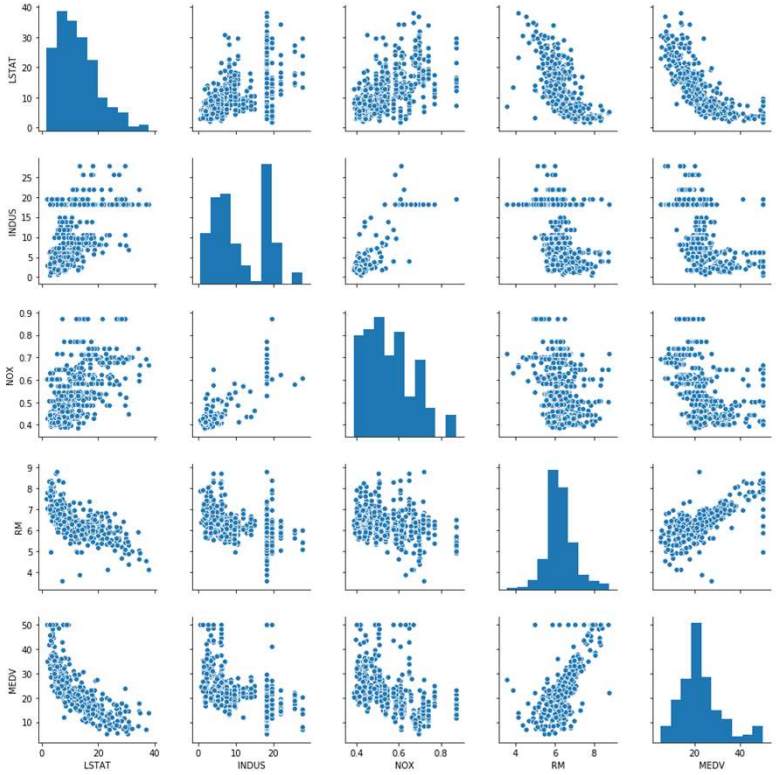
R^2

- The strength of the fit of a linear model is most commonly evaluated using R^2.
- R^2 is calculated as the square of the correlation coefficient.
- It tells us what percent of variability in the response variable is explained by the model.
- The remainder of the variability is explained by variables not included in the model or by inherent randomness in the data.
- For the model we've been working with, $R^2 = -0.62^2 = 0.38$.

Lab: Introducing Linear Regression

- http://localhost:8888/notebooks/python-machine-learning-book-2nd-edition/code/ch10/ch10-2.ipynb



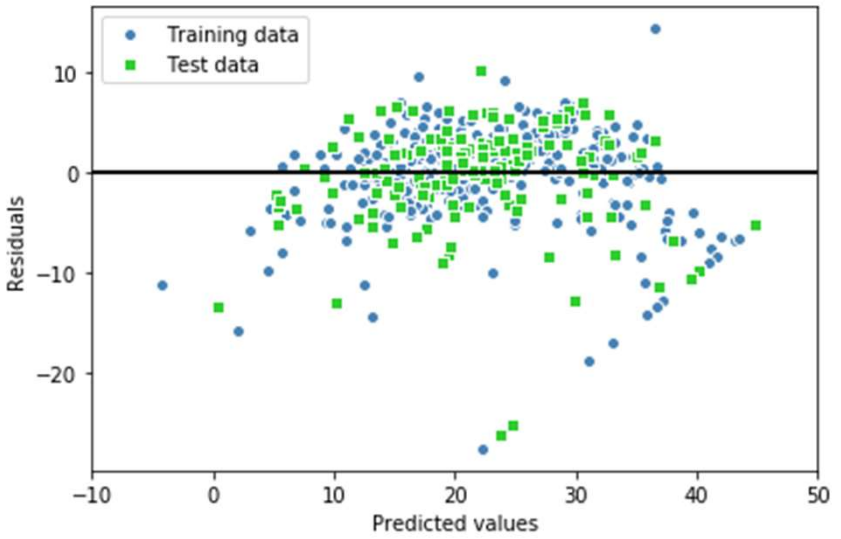


Lab: Robust Regression with RANSAC

- Linear regression models can be heavily impacted by the presence of outliers
- As an alternative to throwing out outliers, we will look at a robust method of regression using the **RANdom SAmple Consensus** (**RANSAC**) algorithm, which fits a regression model to a subset of the data, the so-called **inliers**
- We can summarize the iterative RANSAC algorithm as follows:
    - Select a random number of samples to be inliers and fit the model.
    - Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers.
    - Refit the model using all inliers.
    - Estimate the error of the fitted model versus the inliers.
    - Terminate the algorithm if the performance meets a certain user-defined threshold or if a fixed number of iterations were reached; go back to step 1 otherwise.

Lab: Evaluating the Performance of Regression Models

- **Residual plots** are a commonly used graphical tool for diagnosing regression models. They can help detect nonlinearity and outliers, and check whether the errors are randomly distributed.
- For a good regression model, we would expect that the errors are randomly distributed and the residuals should be randomly scattered around the centerline. If we see patterns in a residual plot, it means that our model is unable to capture some explanatory information, which has leaked into the residuals



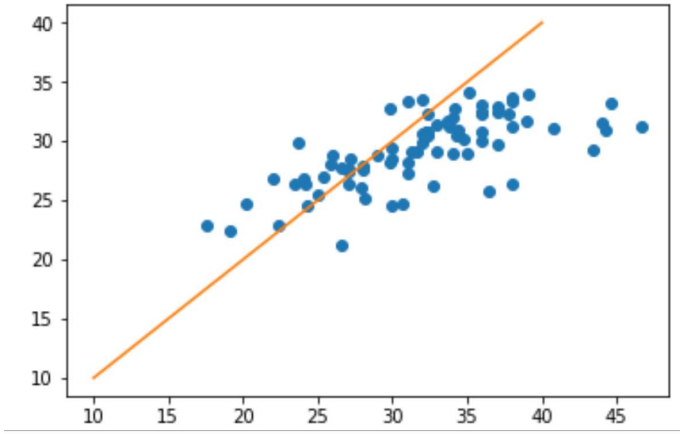
Lab: Prediction with Multivariate Linear Regression

http://localhost:8888/notebooks/real-world-machine-learning/Chapter%203%20-%20Modeling%20and%20prediction.ipynb

- Linear Regression with the MPG Model
- A useful quantitative measure of a model's performance is the so-called **Mean Squared Error**(**MSE**), which is simply the averaged value of the SSE cost that we minimized to fit the linear regression model
- **R2** is another important performance indicator (see above)

```
pred_mpg = reg.predict(auto_test.drop('mpg',axis=1))
```

```
plot(auto_test.mpg, pred_mpg, 'o')
x = linspace(10,40,5)
plot(x, x, '-');
```

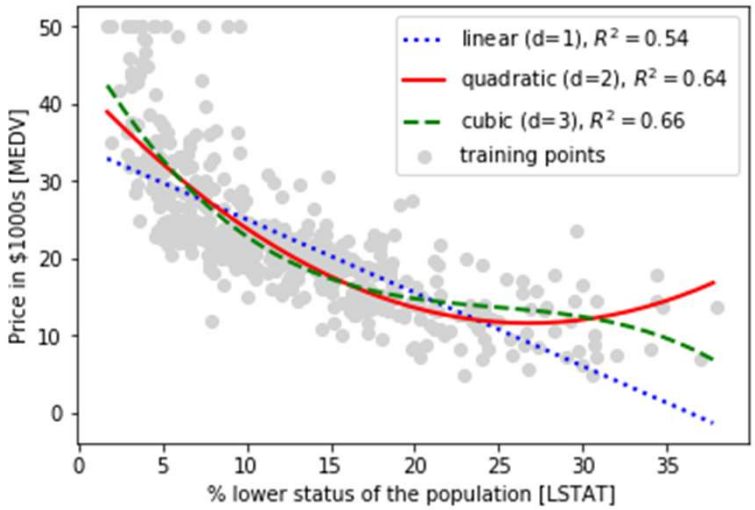


Lab: Dealing With Non-Linear Regression

- http://localhost:8888/notebooks/python-machine-learning-book-2nd-edition/code/ch10/ch10-2.ipynb#Turning-a-linear-regression-model-into-a-curve---polynomial-regression

We use a polynomial regression model by adding polynomial terms:

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d$$



Performing regression on complex, nonlinear data

In some datasets, the relationship between features can't be fitted by a linear model, and algorithms such as linear regression may not be appropriate if accurate predictions are required.

Other properties, such as scalability, may make lower accuracy a necessary trade-off.

Also, there's no guarantee that a nonlinear algorithm will be more accurate, as you risk overfitting to the data.

Trees and Random Forests

As an example of a nonlinear regression model, we introduce the random forest algorithm. Random forest is a popular method for highly nonlinear problems for which accuracy is important. RF (and trees in general can also be applied to classification problems). The basis of the RF algorithm is the decision tree. The decision tree algorithm lets the computer figure out, based on the training set, which variables are the most important, and put them in the top of the tree, and then gradually use less-important variables.

A problem with decision trees is their high variance: the top levels of the tree have a huge impact on the answer, and if the new data doesn't follow exactly the same distribution as the training set, the ability to generalize might suffer.

This is where the random forest method comes in. By building a collection of decision trees, you mitigate this risk. When making the answer, you pick the majority vote in the case of classification, or take the mean in case of regression. Because you use votes or means, you can also give back full probabilities in a natural way that not many algorithms share.

Lab: comparing Linear Regression and Random Forest Regression

Do the exercise: perform a regression using RandomForestRegression and assess any improvement in the prediction.

Predicting numerical values with a regression model

We use the the Linear Regression algorithm to predict miles-per-gallon of various automobiles.

In [37]:
```
auto = pandas.read_csv("data/auto-mpg.csv")

# Convert origin to categorical variable
auto = auto.join(cat_to_num(auto['origin']))
auto = auto.drop('origin', axis=1)

# Split in train/test set
auto_train = auto[:int(0.8*len(auto))]
auto_test = auto[int(0.8*len(auto)):]
```

Out[37]:

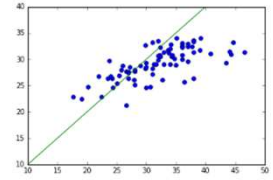
	mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	origin=1	origin=2	origin=3
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	0	0
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	0	0
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	0	0
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	0	0
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	0	0

In [38]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
reg = LinearRegression()
regrf = RandomForestRegressor()
reg.fit(auto_train.drop('mpg', axis=1), auto_train["mpg"])
```

Out[38]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

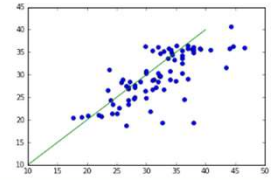
In [39]:
```
pred_mpg = reg.predict(auto_test.drop('mpg',axis=1))
```

In [40]:
```
plot(auto_test.mpg, pred_mpg, 'o')
x = linspace(10,40,5)
```



In [41]:
```
pred_mpg_rf = regrf.predict(auto_test.drop('mpg',axis=1))
```

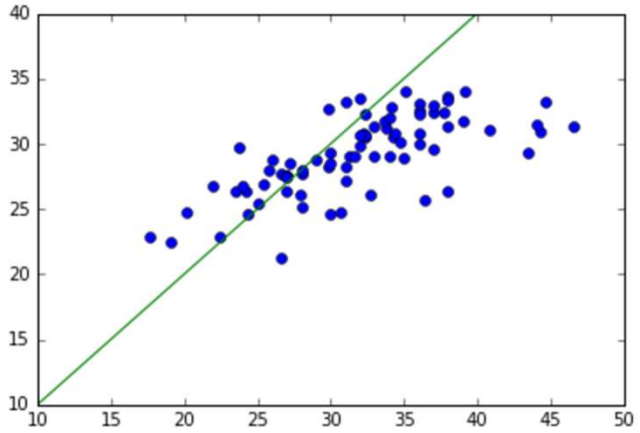
In [42]:
```
plot(auto_test.mpg, pred_mpg_rf, 'o')
x = linspace(10,40,5)
```



Comparison of R-squared (model score)

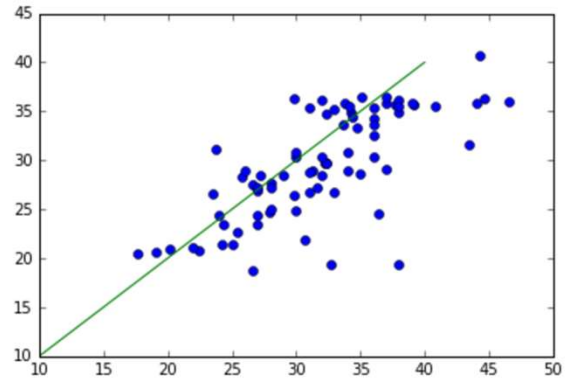
```
In [39]: pred_mpg = reg.predict(auto_test.drop('mpg',axis=1))
```

```
In [40]: plot(auto_test.mpg, pred_mpg, 'o')
         x = linspace(10,40,5)
         plot(x, x, '-');
```




```
In [41]: pred_mpg_rf = regrf.predict(auto_test.drop('mpg',axis=1))
```

```
In [42]: plot(auto_test.mpg, pred_mpg_rf, 'o')
         x = linspace(10,40,5)
         plot(x, x, '-');
```




```
In [43]: print reg.score(auto_train.drop('mpg', axis=1), auto_train["mpg"])
         print regrf.score(auto_train.drop('mpg', axis=1), auto_train["mpg"])
```

```
0.821497671896
0.978717452231
```

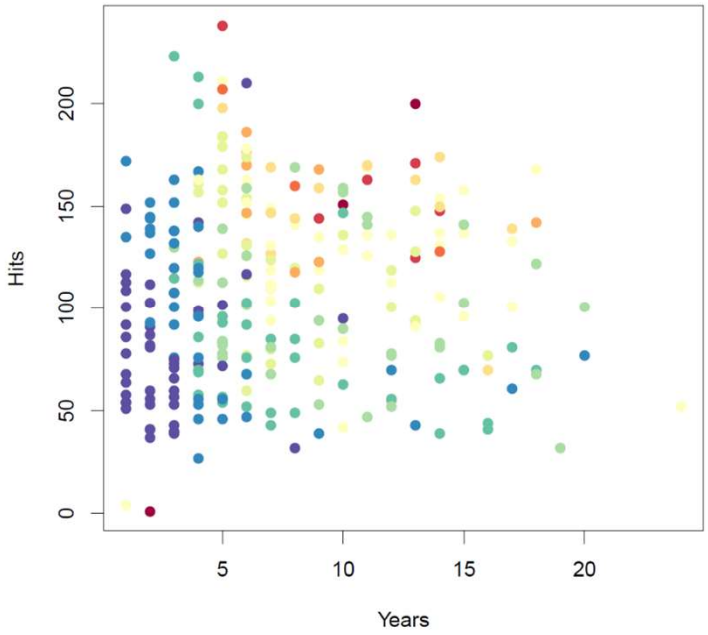
Tree-based methods

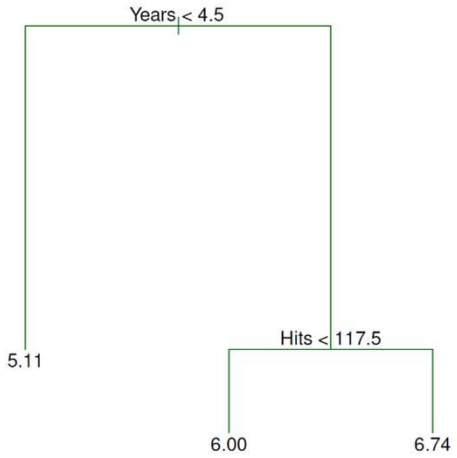
- Here we describe *tree-based* methods for regression and classification.
- These involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision-tree* methods.

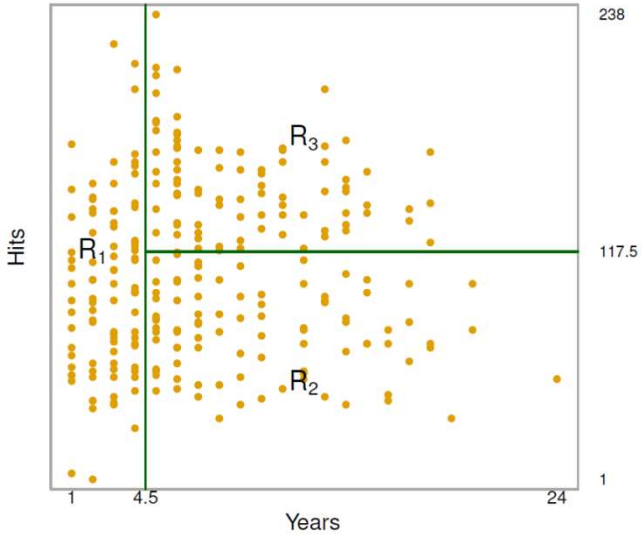
Pros and Cons of Trees

- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

Example: Baseball Salary Data







Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text `Years`<4.5 and `Hits`<117.5.

Terminology for Trees

- In keeping with the *tree* analogy, the regions R_1, R_2, and R_3 are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text `Years`<4.5 and `Hits`<117.5.

Tree-building process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Tree-building process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or *boxes*, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

* Residual Sum of Squares

Classification trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

Details of Classification Trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the *classification error rate*. this is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}).$$

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

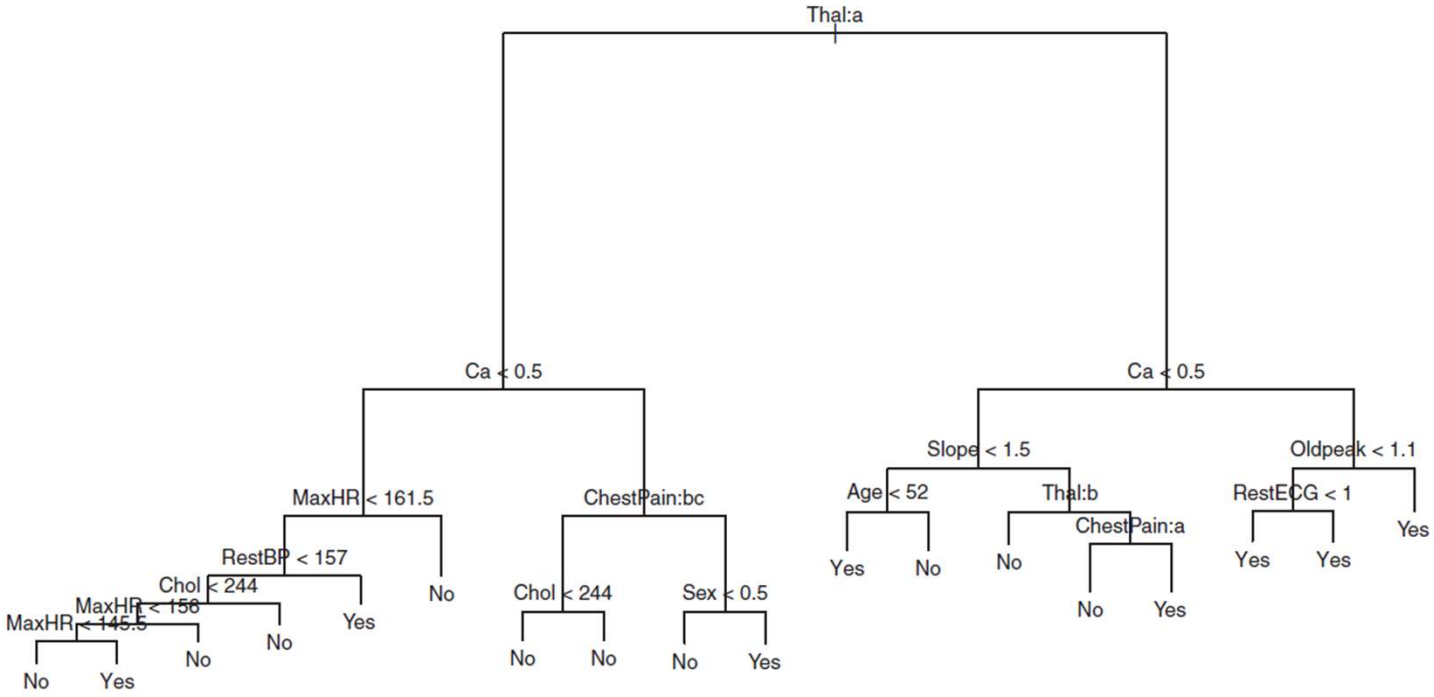
Gini index

- The *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

  a measure of total variance across the K classes. The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.

Example of Classification Tree: Heart Data



Recap

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ Trees can easily handle qualitative predictors without the need to create dummy variables.
- ▼ Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.

Bagging

- *Bootstrap aggregation*, or *bagging*, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

- In other words, *averaging a set of observations reduces variance*. Of course, this is not practical because we generally do not have access to multiple training sets.

Average of Bootstrapped Trees

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. We then average all the predictions to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

This is called *bagging*.

Bagging classification trees

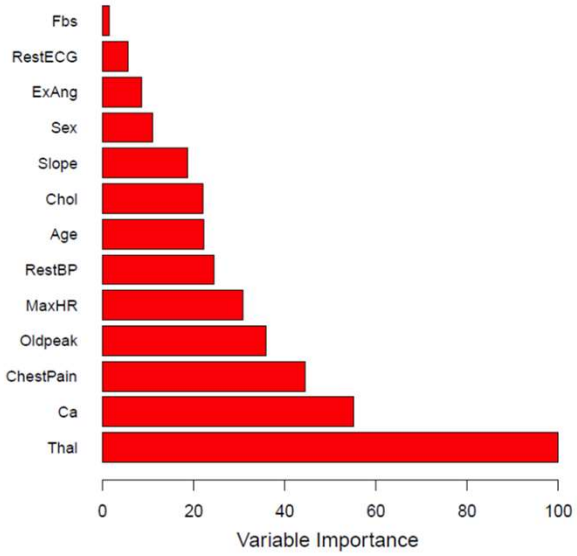
- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the B predictions.

Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, *a random selection of m predictors* is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot for the `Heart` data

Why decorrelation is needed?

In building a Random Forest, at each split in the tree, the algorithm **is not even allowed to consider a majority** of the available predictors. Why? Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, **most or all of the trees will use this strong predictor in the top** split.

Consequently, **all of the bagged trees will look quite similar to each other**, hence the predictions from the bagged trees will be highly correlated. Unfortunately, **averaging many highly correlated quantities does not lead to as large of a reduction in variance** as averaging many uncorrelated quantities.

In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting. Random forests overcome this problem by forcing each split to consider only a subset of the predictors.

Lab: Classification Tree on Titanic with Python

https://www.kaggle.com/willkoehrsen/visualize-a-decision-tree-w-python-scikit-learn

https://github.com/thimotyb/real-world-machine-learning/blob/python3/DecisionTreeDemo.ipynb

```
67 features = prepare_data(data_train)
68 features[:5]
69 model = DecisionTreeClassifier()
70 model.fit(features, data_train["Survived"])
71
72 #print model.decision_path(features)
73
74 tree_to_code(model, features.columns)
75 print model.score(prepare_data(data_test), data_test["Survived"])
76
77
78
```

```
Python                                    ~\Dropbox\corsi\machine_learning\real-world-machine-learning-master
In [38]: %run "C:/Users/thimo/Dropbox/corsi/machine_learning/real-world-machine-learning-
master/decision-tree.py"
def tree(Pclass, Age, SibSp, Parch, sqrt_Fare, Sex=female, Sex=male, Embarked=C, Embarked=Q,
Embarked=S, Embarked=nan):
  if Sex=female <= 0.5:
    if Pclass <= 1.5:
      if Age <= 53.0:
        if Age <= -0.0399999916553:
          if sqrt_Fare <= 6.2348575592:
            if sqrt_Fare <= 5.93713378906:
              if sqrt_Fare <= 5.12584447861:
                return [[ 3.  0.]]
              else:  # if sqrt_Fare > 5.12584447861
```

Lab: Comparing Tree, Forests and Gradient Boost

- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

- https://github.com/thimotyb/real-world-machine-learning/blob/python3/Forest%2C_Features_and_LightGBM.ipynb

Model Evaluation and Optimization

Model evaluation: the problem

When you evaluate the performance of a model, you want to determine how well that model will perform on new data

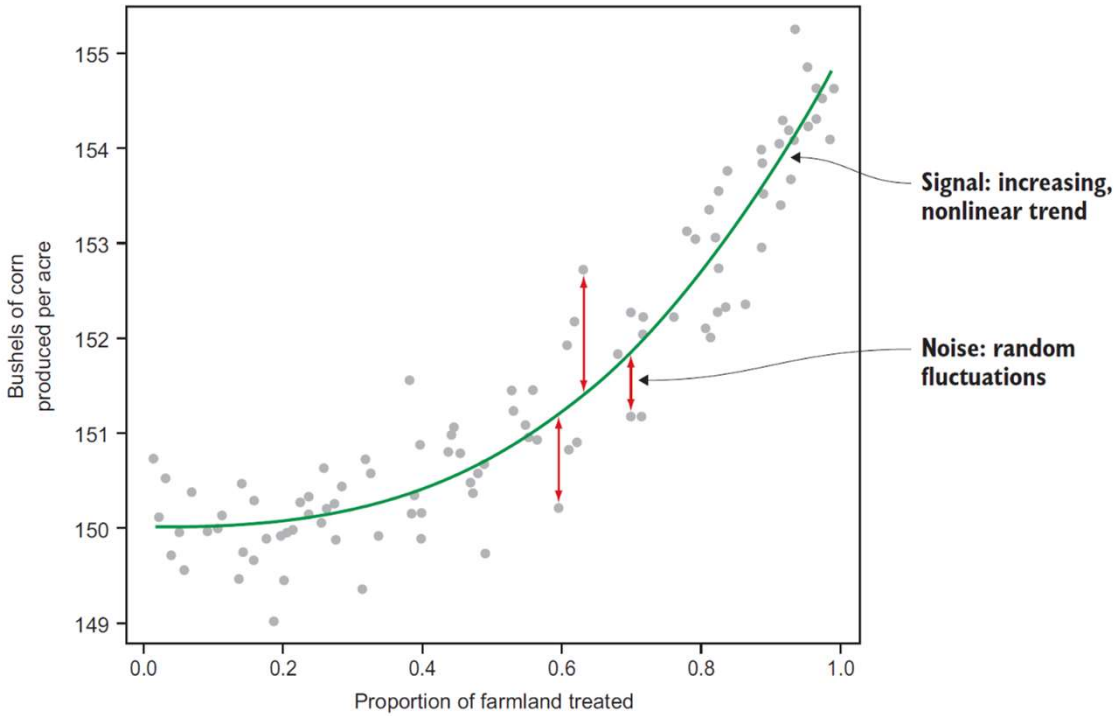
The general technique is called Cross-Validation, and it involves separating the data in training and test data.

Overfitting and Model Optimism are typical problems.

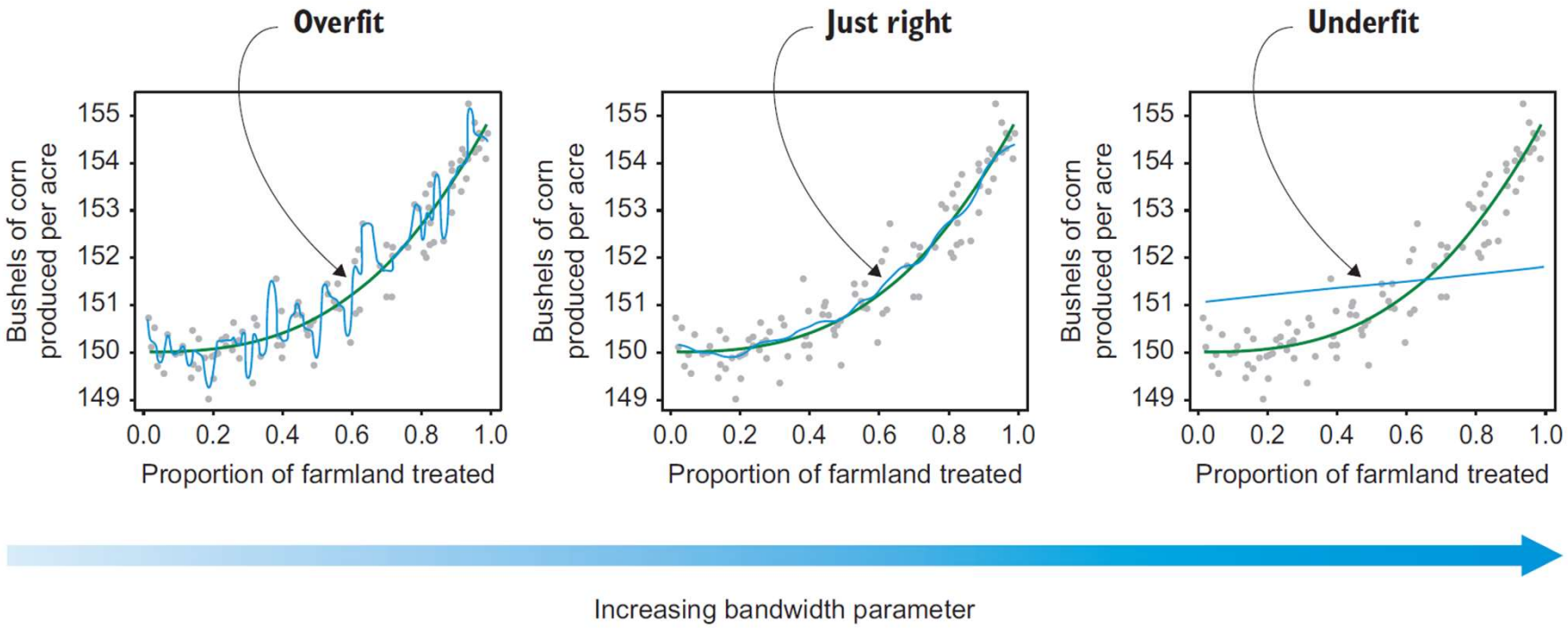
Let's look at an example

Performing a non-linear regression

We use Kernel Smoothing: a non-parametric, nonlinear regression with one tuning parameter, called *bandwidth*.

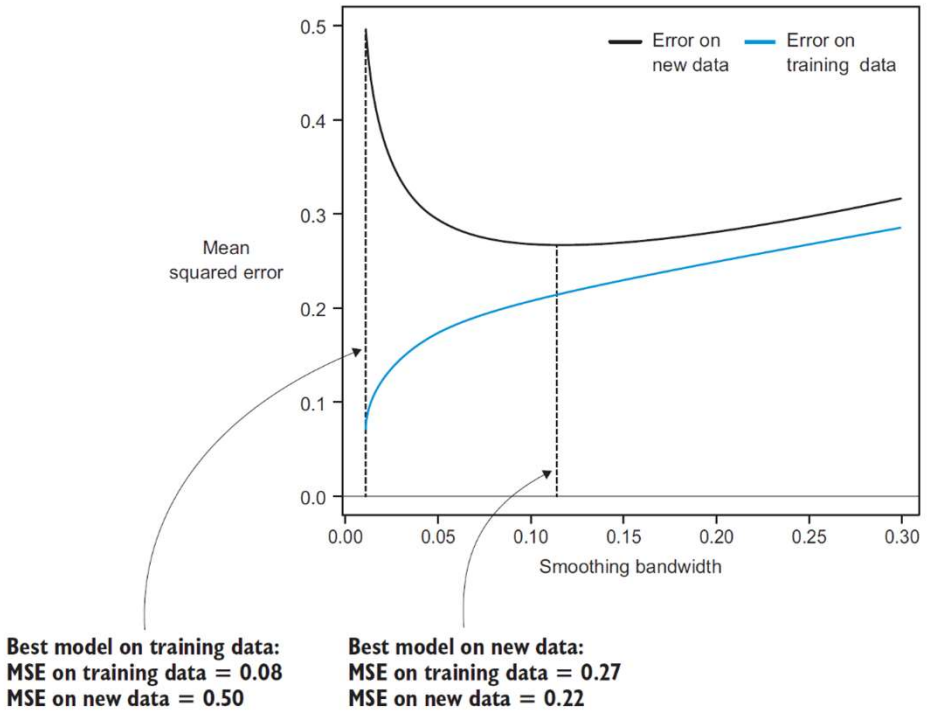


Tuning bandwidth in Kernel Smoothing



Model Optimism: MSE on training data or new data

For regression, the standard metric for evaluation is mean squared error (MSE), which is the average squared difference between the true value of the target variable and the model-predicted value



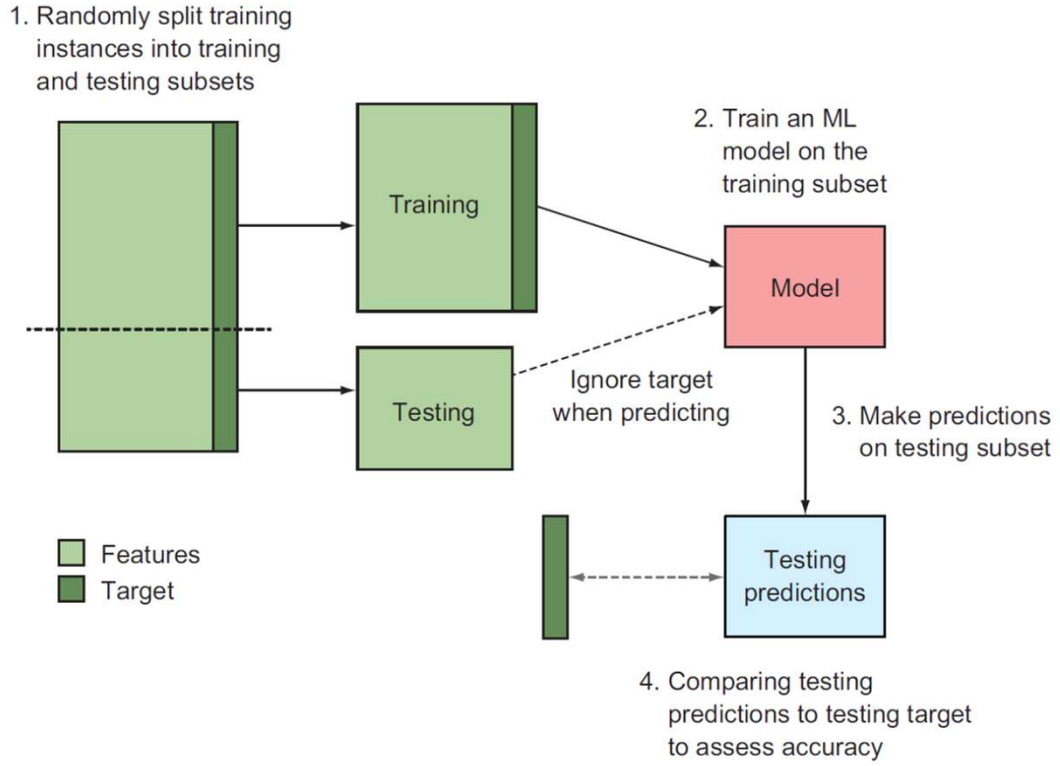
Cross Validation (CV): Holdout method

You use only the training subset to fit the model, and only the testing subset to evaluate the accuracy of the model.

This approach is referred to as the holdout method, because a random subset of the training data is held out from the training process.
Practitioners typically leave out 20–40% of the data as the testing subset.

```
# assume that we begin with two inputs:
#    features - a matrix of input features
#    target - an array of target variables corresponding to those features
features = rand(100,5)
target = rand(100) > 0.5

N = features.shape[0] # The total number of instances
N_train = floor(0.7 * N) # The total number of training instances

idx = random.permutation(N)              <-- Randomizes index

idx_train = idx[:N_train]
idx_test = idx[N_train:]                 | Splits index

features_train = features[idx_train,:]
target_train = target[idx_train]
features_test = features[idx_test,:]
target_test = target[idx_test]           | Breaks your data into training and testing subsets
```

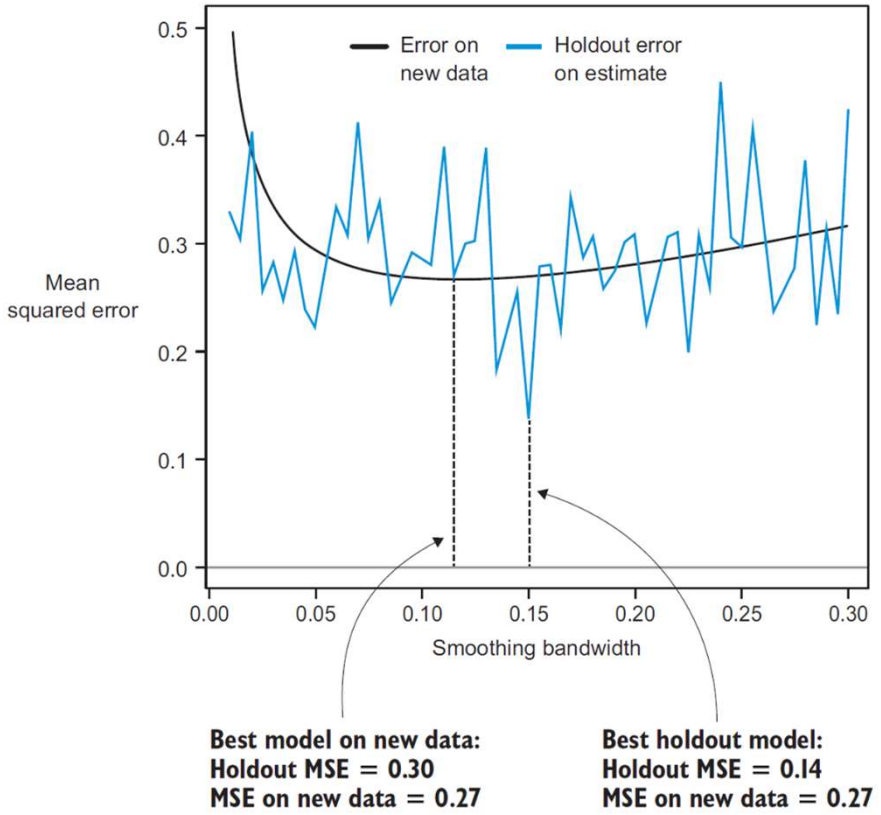
Iterating the holdout method



MSE on CV with holdout

The error estimates computed by the holdout method are close to the new-data error of the model. They're certainly much closer than the training set error estimates, particularly for small-bandwidth values.

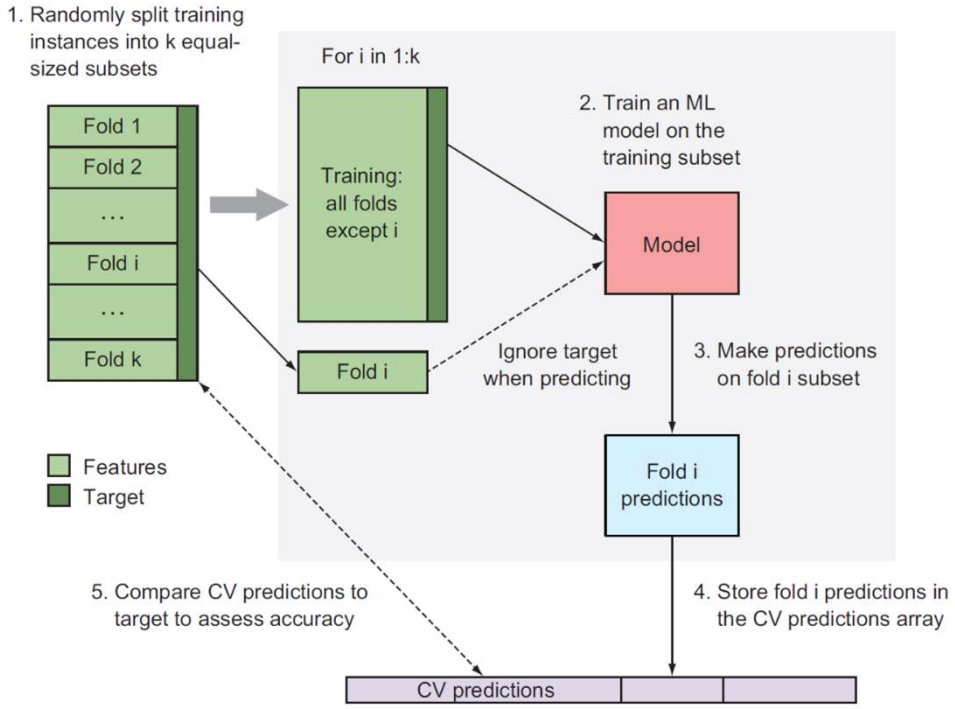
The holdout error estimates are noisy. They bounce around wildly compared to the smooth curve that represents the error on new data.



K-Fold Cross Validation

k-fold CV begins by randomly splitting the data into k disjoint subsets, called folds. For each fold, a model is trained on all the data except the data from that fold and is subsequently used to generate predictions for the data from that fold

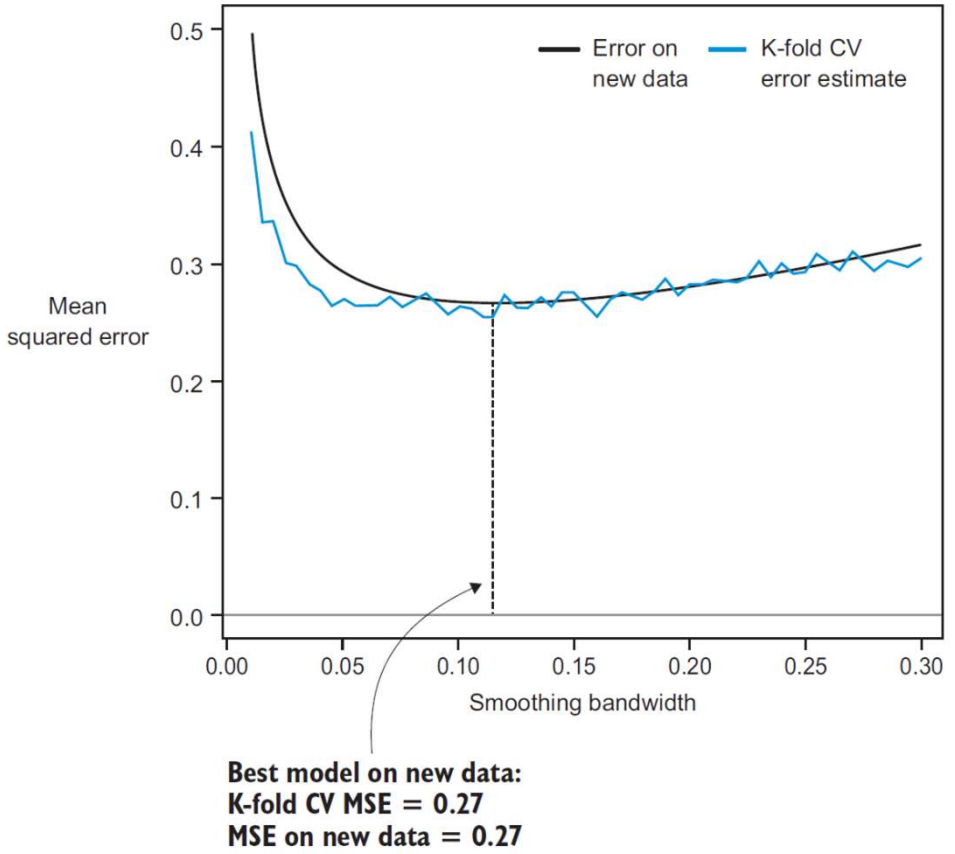
After all k-folds are cycled through, the predictions for each fold are aggregated and compared to the true target variable to assess accuracy



MSE for k-fold CV

Cross-validation methods (both holdout and k-fold) assume that the training data forms a representative sample from the population of interest: Ensure that any potential biases in the training data are addressed and minimized.

The larger the number of folds used in k-fold cross-validation, the better the error estimates will be.



Algorithm for k-fold CV

```
N = features.shape[0]
K = 10 # number of folds

preds_kfold = np.empty(N)
folds = np.random.randint(0, K, size=N)

for idx in np.arange(K):

    features_train = features[folds != idx,:]
    target_train = target[folds != idx]
    features_test = features[folds == idx,:]

    # Build and predict for CV fold (to be filled out)
    # model = train(features_train, target_train)
    # preds_kfold[folds == idx] = predict(model, features_test)

# accuracy = evaluate_acc(preds_kfold, target)
```

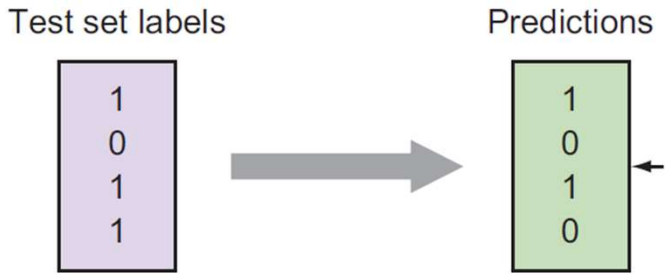
Loops over the folds

Breaks your data into training and testing subsets

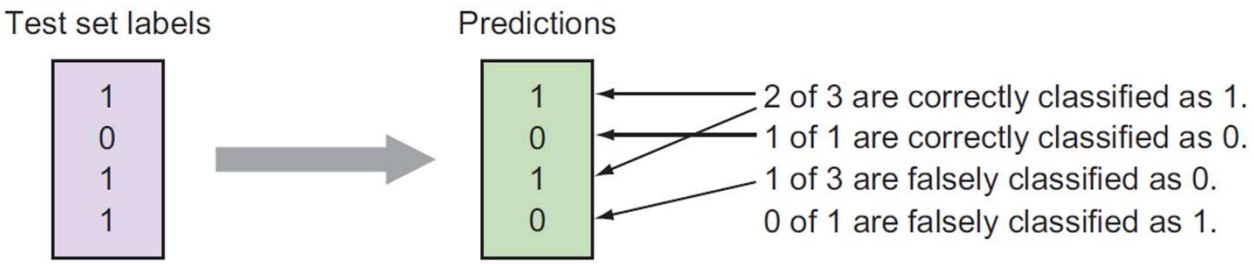
Evaluation of classification models

We begin our discussion of evaluating classification models by presenting problems with only two classes. Evaluation is **always done in CV** with test data held out from the training data.

The simplest performance measure of a classification model is to calculate the fraction of correct answers; if three out of four rows were correctly predicted, you'd say the accuracy of the model on this particular validation set is 3/4 = 0.75, or 75%.



Class-wise accuracy

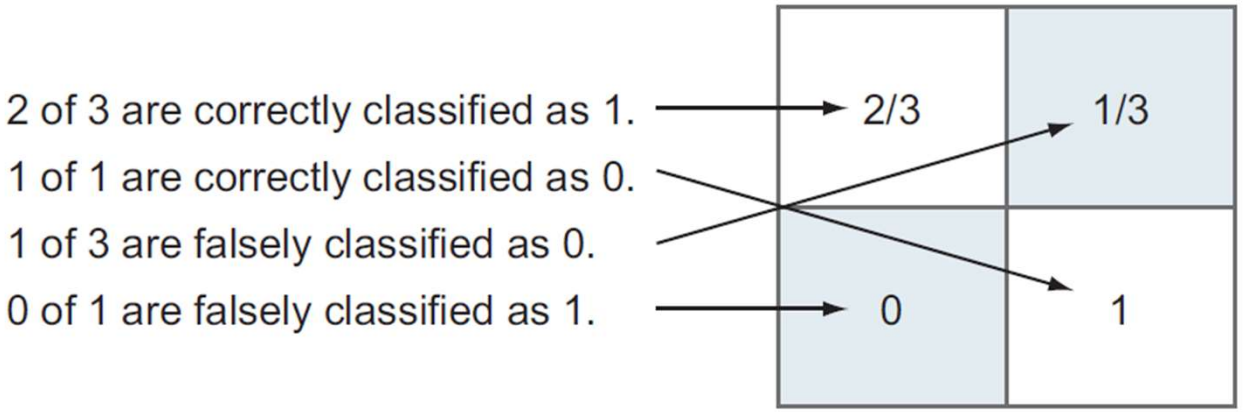


The predictions provide more information than simply being correct or not, and analyze the accuracy per class. For binary classification, you can be wrong in two ways:

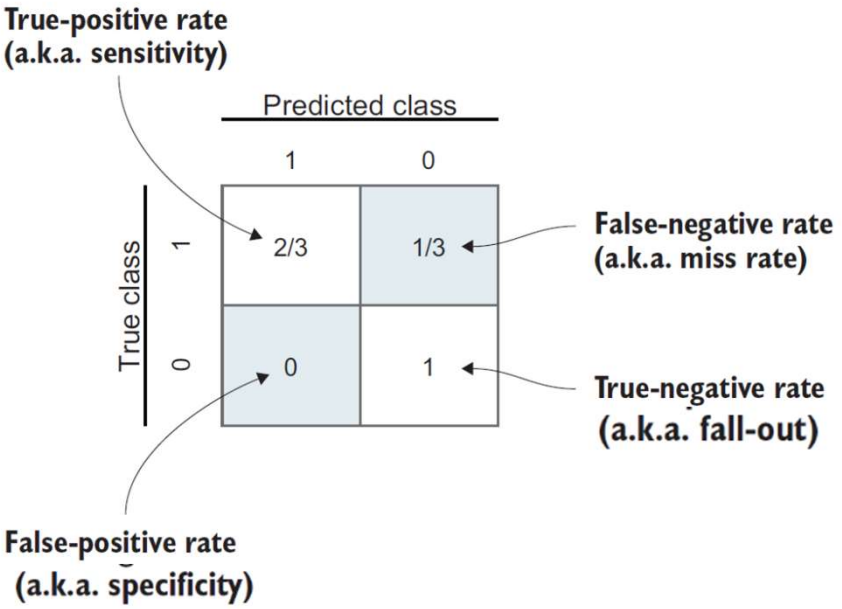
predicting 0 when the correct value is 1, or predicting 1 when the correct value is 0. In the same way, you can be correct in two ways.

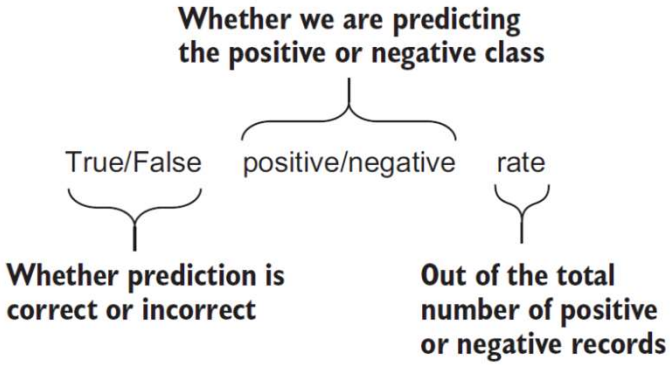
Confusion Matrix

It is useful to display these four numbers in a two-by-two diagram called a ***confusion matrix***



Receiver operating characteristics (ROCs)





Accuracy trade-offs

Output from classifier: class probabilities

	Survived	Died
15	0.092	0.908
16	0.904	0.096
17	0.646	0.354
18	0.740	0.260
19	0.460	0.540

Sorted probabilities

	Survived	Died
308	0.705	0.295
215	0.703	0.297
217	0.700	0.300
54	0.698	0.302
169	0.698	0.302

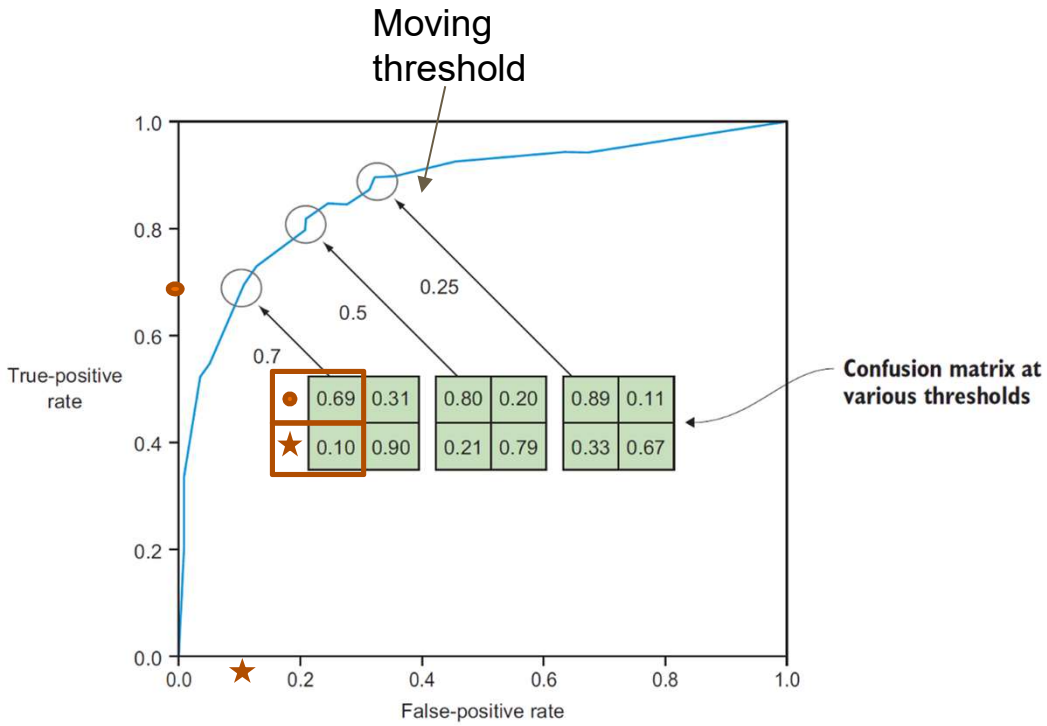
Threshold: "survived" probabilities > 0.7

Many classification algorithms output not only the zero-one predictions, but the full prediction probabilities. The output of a probabilistic classifier is what we call the probability vectors or class probabilities. If we use a 0.5 threshold probability to distinguish classes, we say that the threshold that determines the class is 0.5. Using any other threshold would get different values for all of your metrics.

ROC curves

Moving the trade-off threshold for classification, you get the confusion matrix and the ROC metrics at this particular threshold.

If you follow this process for all thresholds from 0 to 1, you define the ***ROC curve***



Algorithm for ROC curve

```
import numpy as np

def roc_curve(true_labels, predicted_probs, n_points=100, pos_class=1):

  thr = np.linspace(0,1,n_points)
  tpr = np.zeros(n_points)
  fpr = np.zeros(n_points)

  pos = true_labels == pos_class
  neg = np.logical_not(pos)
  n_pos = np.count_nonzero(pos)
  n_neg = np.count_nonzero(neg)

  for i,t in enumerate(thr):
    tpr[i] = np.count_nonzero(np.logical_and(
             predicted_probs >= t, pos)) / n_pos
    fpr[i] = np.count_nonzero(np.logical_and(
             predicted_probs >= t, neg)) / n_neg
  return fpr, tpr, thr
```

Returns the false-positive and true-positive rates at n_points thresholds for the given true and predicted labels

Allocates the threshold and ROC lists

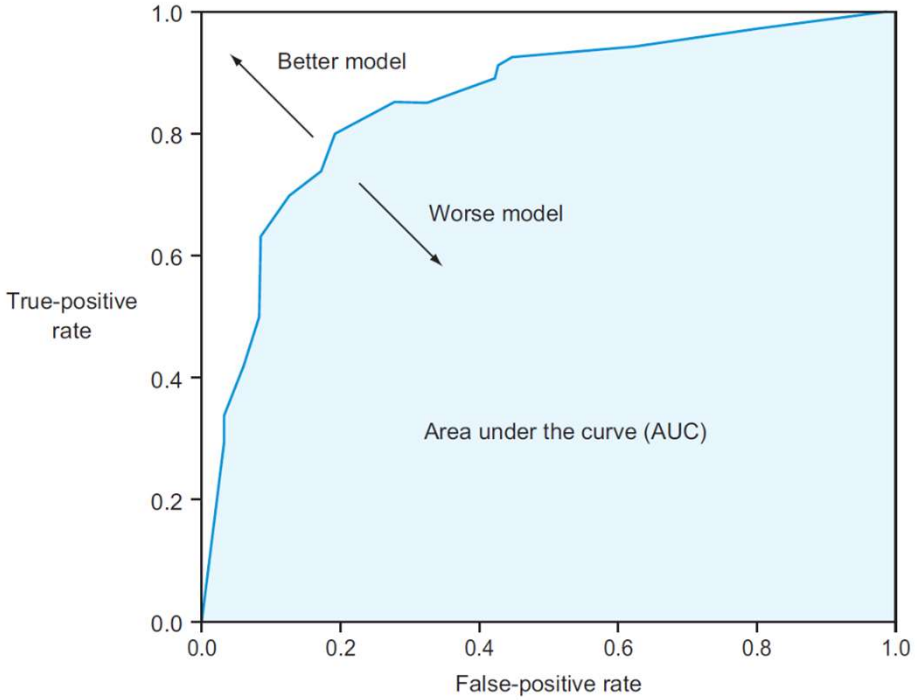
Precalculates values for the positive and negative cases, used in the loop

For each threshold, calculates the rate of true and false positives

AUC (Area Under Curve)

A perfect classifier would have no false positives and no missed detections, so the curve would be pushed to the top-left corner.

This leads us naturally to another evaluation metric: the area under the ROC curve (AUC). The larger this area, the better the classification performance.

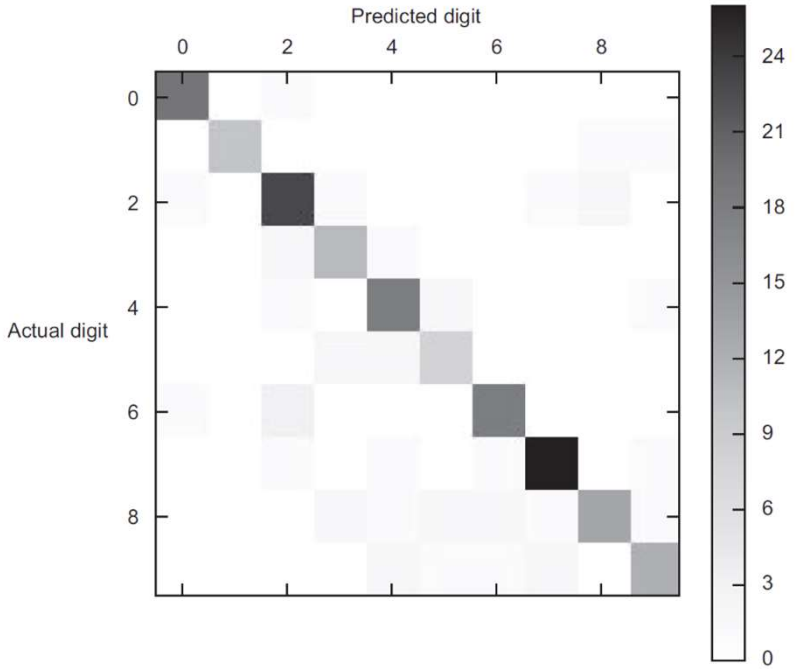


Multiclass Classification

In a multiclass confusion matrix every element in the matrix is the class on the row versus the class on the column.

The cell contains the number of predicted instances for the actual instance.

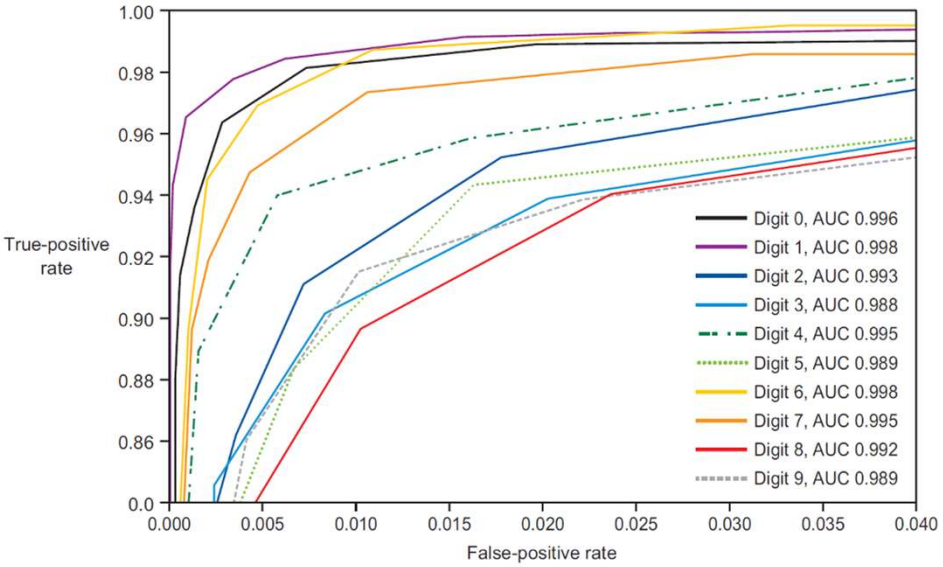
Areas shaded outside the diagonal provide a visual idea of the confusion in prediction



Multiclass ROC

The ROC curve is in principle applicable to only binary classification problems, because you divide the predictions into positive and negative classes in order to get ROC metrics such as the true-positive rate and false-positive rate commonly used on the ROC curve axis.

To simulate binary classification in a multiclass problem, you use the one-versus-all trick. For each class, you denote the particular class as the positive class, and everything else as the negative class, and you draw the ROC curve as usual.



Lab: Evaluation of a Model in Python

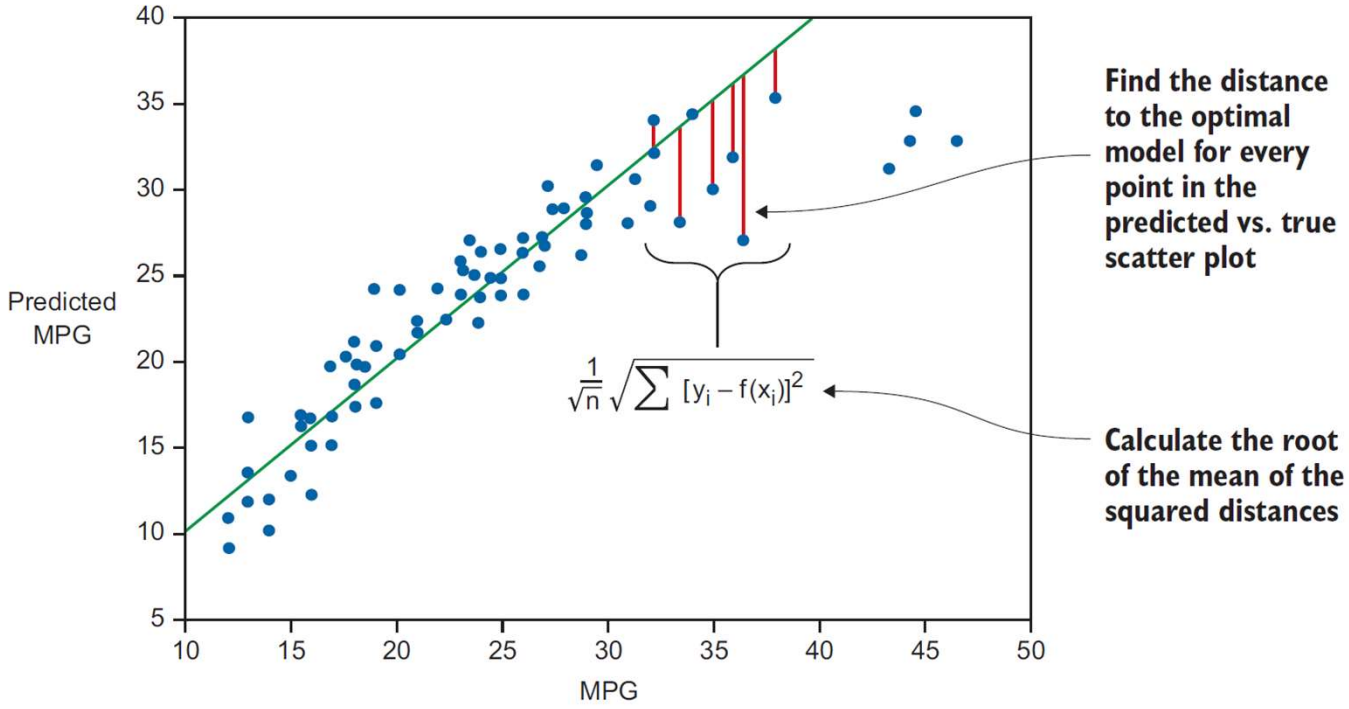
- https://www.kaggle.com/sgus1318/titanic-analysis-learning-to-swim-with-python
- https://www.kaggle.com/mnassrib/titanic-logistic-regression-with-python

Evaluation of regression models

In contrast to classification models, regression carries no simple notion of a correct prediction: there are mainly two simple metrics to measure the regression performance: the root mean-square error (the square root of the MSE) and the R-squared value

```
def rmse(true_values, predicted_values):
  n = len(true_values)
  residuals = 0
  for i in range(n):
    residuals += (true_values[i] - predicted_values[i])**2.
  return np.sqrt(residuals/n)
```

RMSE


Predicted MPG
MPG
40
35
30
25
20
15
10
5
10
15
20
25
30
35
40
45
50
$\frac{1}{\sqrt{n}}\sqrt{\sum [y_i - f(x_i)]^2}$
Find the distance to the optimal model for every point in the predicted vs. true scatter plot
Calculate the root of the mean of the squared distances

RMSE

The advantage of RMSE is that the result is in the same units as the values themselves, but it's also a disadvantage in the sense that the RMSE value depends on the scale of the problem, and thus isn't easily comparable across datasets.

If the predicted or actual values are larger numbers, the RMSE will be correspondingly higher. Although this isn't a problem when comparing models in the same project, it can be a challenge to understand the overall model performance and compare it to other models in general.

R2

the R-squared, or R2, metric whose response is relative and always in the 0–1 range. If the model can predict the data better, the R-squared value is closer to 1.

```
def r2(true_values, predicted_values):
  n = len(true_values)
  mean = np.mean(true_values)
  residuals = 0
  total = 0
  for i in range(n):
    residuals += (true_values[i] - predicted_values[i])**2.
    total += (true_values[i] - mean)**2.
  return 1.0 - residuals/total
```

Parameter Tuning

Most machine-learning models come endowed with one or more tuning parameters that control the inner workings of the learning algorithm.

These tuning parameters typically control the complexity of the relationship between the input features and target variable. As a result, the tuning parameters can have a strong influence on the fitted model and its predictive accuracy on new data.

The standard way to optimize the choice of tuning parameters for an ML model is via a brute-force grid search.

http://localhost:8888/notebooks/real-world-machine-learning/Chapter%204%20-%20Evaluation%20and%20Optimization.ipynb

Lab: Grid search on SVM on the Titanic Dataset

Use AUC as your optimization metric and SVM with a radial basis function (RBF) kernel as your classification algorithm. For kernel SVM with an RBF kernel, you have two standard tuning parameters: the kernel coefficient, gamma; and the penalty parameter, C.

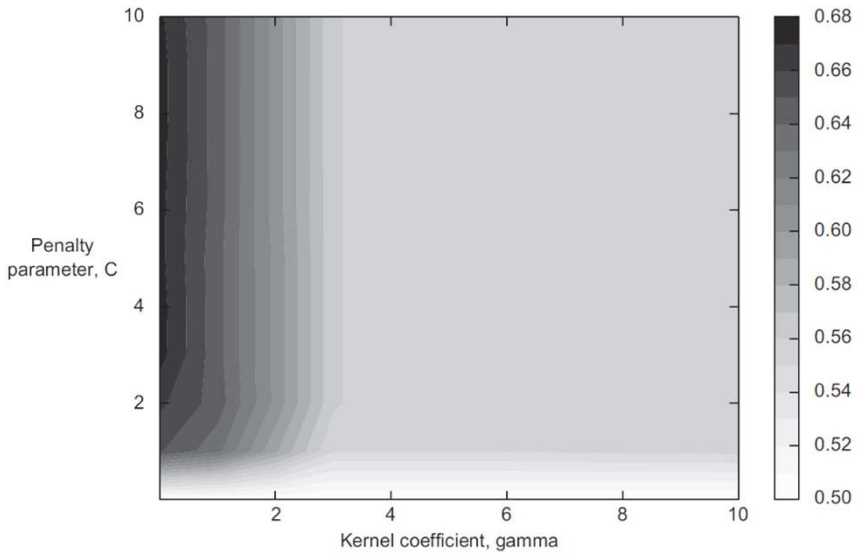
```
# Inputs: X - features, y - target

import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC

# grid of (gamma, C) values to try
gam_vec, cost_vec = np.meshgrid(np.linspace(0.01, 10., 11),
                    np.linspace(1., 10., 11))

AUC_all = []                                  <- Initializes empty array to store AUC results

# set up cross-validation folds
N = len(y)
K = 10                                        <- Number of cross-validation folds
folds = np.random.randint(0, K, size=N)

# search over every value of the grid
for param_ind in np.arange(len(gam_vec.ravel())):

    # initialize cross-validation predictions
    y_cv_pred = np.empty(N)

    # loop through the cross-validation folds
    for ii in np.arange(K):
        # break your data into training and testing subsets
        X_train = X.ix[folds != ii,:]
        y_train = y.ix[folds != ii]
        X_test = X.ix[folds == ii,:]

        # build a model on the training set
        model = SVC(gamma=gam_vec.ravel()[param_ind],
     C=cost_vec.ravel()[param_ind])
        model.fit(X_train, y_train)

        # generate and store model predictions on the testing set
        y_cv_pred[folds == ii] = model.predict(X_test)

    # evaluate the AUC of the predictions
    AUC_all.append(roc_auc_score(y, y_cv_pred))
```

Grid search

1. Choose the evaluation metric that you want to maximize (for example, AUC for classification, R2 for regression).
2. Choose which ML algorithm you want to use (for example, random forest).
3. Select which tuning parameters you want to optimize over (for example, number of trees and number of features per split) and the array of values to test for each parameter.
4. Define the grid as the Cartesian product between the arrays of each tuning parameter.
5. For each combination of tuning parameters in the grid, use the training set to perform cross-validation (using either the hold-out or k-fold-CV method) and compute the evaluation metric on the cross-validated predictions.
6. Finally, select the set of tuning parameters corresponding to the largest value of the evaluation metric. This is the optimized model.

Results of Grid Search

- The maximum occurs at the boundary of the grid (gamma = 0.01), meaning that you'd want to rerun the grid search on an expanded grid.
- A high amount of sensitivity exists in the accuracy of the predictions to the numerical value of the gamma parameter, meaning that you need to increase the granularity of sampling of that parameter.
- The maximum value occurs near gamma = 0, so expressing the grid on a log scale (for example, 10-4, 10-3, 10-2, 10-1) is sensible.
- There's not much sensitivity of the AUC as a function of C, so you can use a coarse sampling of that parameter.

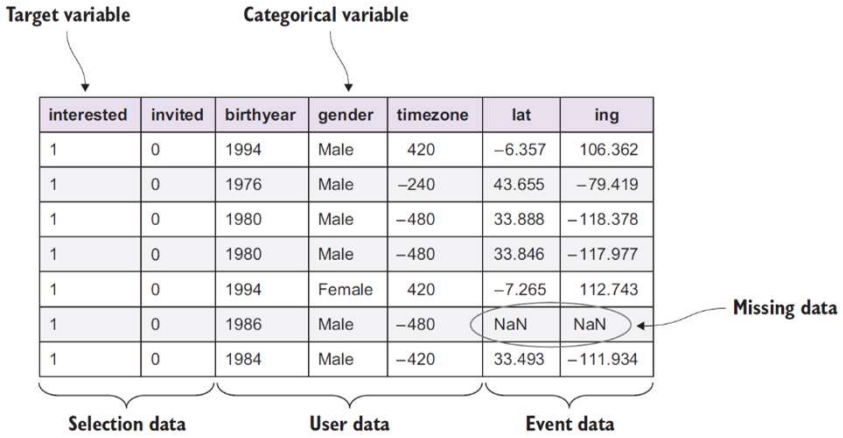


Feature Engineering

Feature engineering is the practice of using mathematical transformations of raw input data to create new features to be used in an ML model.

- You can use feature engineering to produce transformations of your original data that are more closely related to the target variable.
- Instead of relying on the model to infer such subtle relationships from the raw location data, engineered predictive factors become easier to deduce
- Use unstructured data sources in ML models

Examples of Feature Engineering



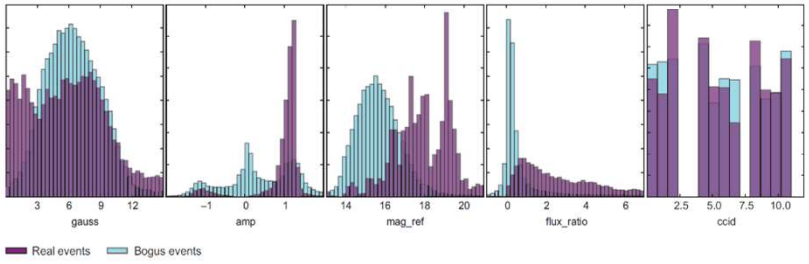
interested	invited	birthyear	gender	timezone	lat	ing
1	0	1994	Male	420	–6.357	106.362
1	0	1976	Male	–240	43.655	–79.419
1	0	1980	Male	–480	33.888	–118.378
1	0	1980	Male	–480	33.846	–117.977
1	0	1994	Female	420	–7.265	112.743
1	0	1986	Male	–480	NaN	NaN
1	0	1984	Male	–420	33.493	–111.934

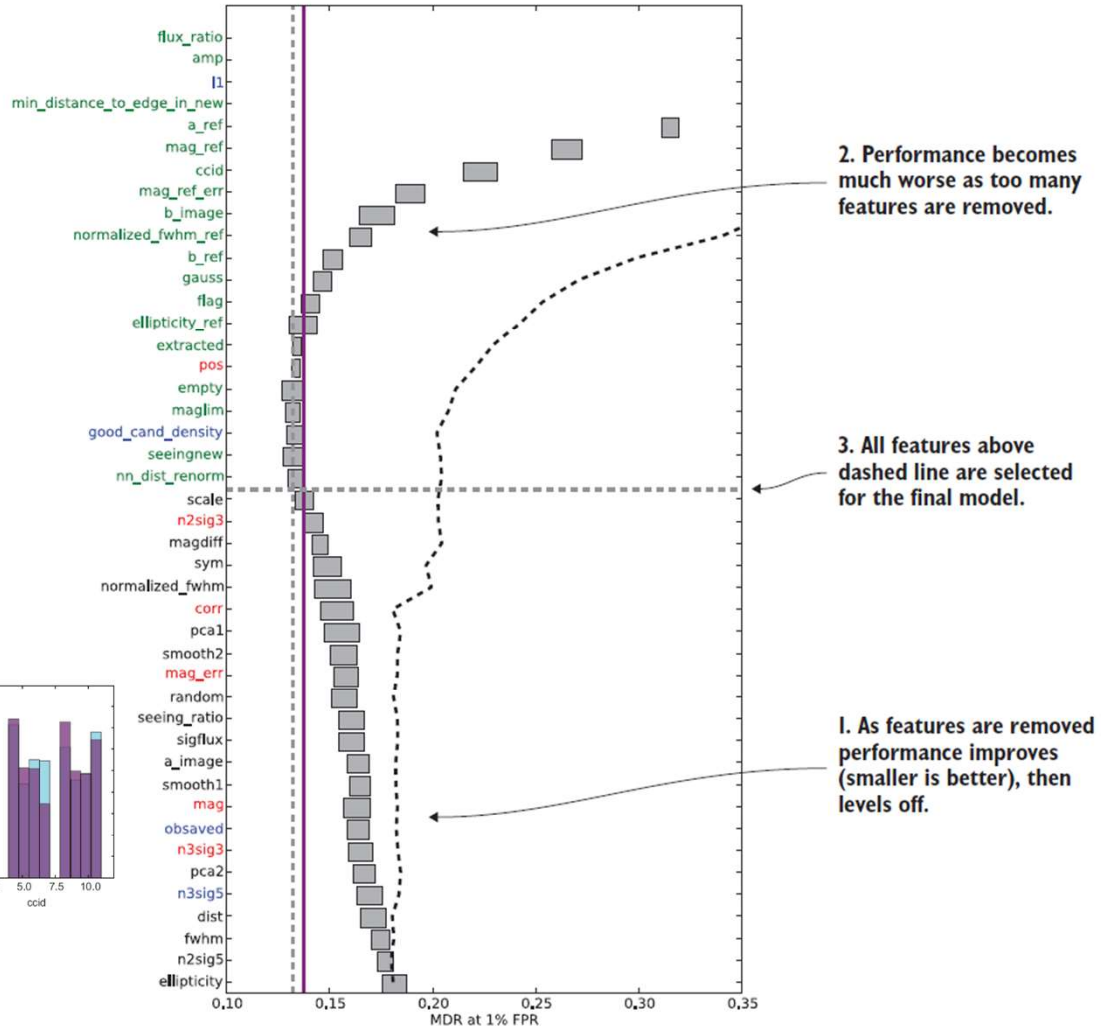
datetime_hour_of_day	datetime_day_of_week	datetime_day_of_month	datetime_day_of_year	datetime_month_of_year
13	4	26	300	10
13	4	26	300	10
13	4	26	300	10
13	4	26	300	10
13	4	26	300	10

datetime_minute_of_hour	datetime_second_of_minute	datetime_year	datetime_quarter_of_year	datetime_week_of_year
30	0	2012	4	43
30	0	2012	4	43
30	0	2012	4	43
30	0	2012	4	43
30	0	2012	4	43

Feature Selection

Feature selection can be perform with forward selection or backward elimination. Some models already have built-in proces for feature selection.

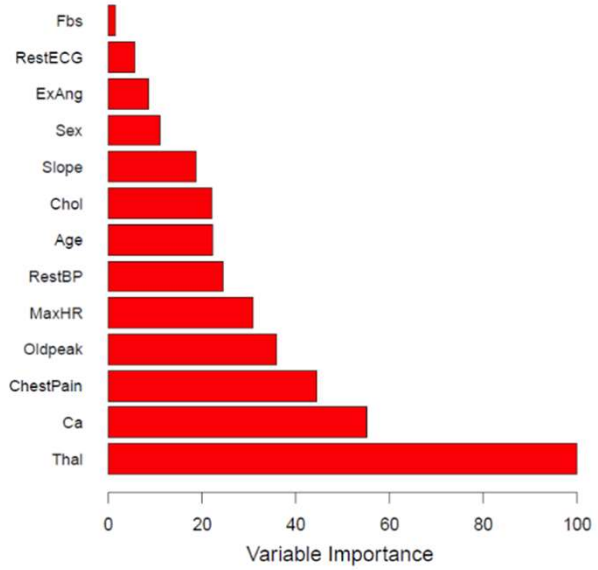




Models with built-in feature selection

Examples of built-in feature-selection methods are the weights assigned to features in linear and logistic regression algorithms

Feature importances in decision trees and ensemble variants such as random forests, which capture (in a computationally efficient manner) the amount that predictive accuracy is expected to decrease if a feature were replaced with random noise.



Bibliography

- Open Intro Statistics: https://www.openintro.org/stat/teachers.php
- Real World Machine Learning: https://www.amazon.it/Real-World-Machine-Learning-Henrik-Brink/dp/1617291927
- Applied Predictive Modeling (Kuhn): http://appliedpredictivemodeling.com/
- Introduction to Statistical Learning With R: http://www-bcf.usc.edu/~gareth/ISL/
- Pandas: http://pandas.pydata.org/pandas-docs/stable/cookbook.html
- Python: https://docs.python.org/2/tutorial/index.html
- Scikit Learn: http://scikit-learn.org/stable/

Neural Networks

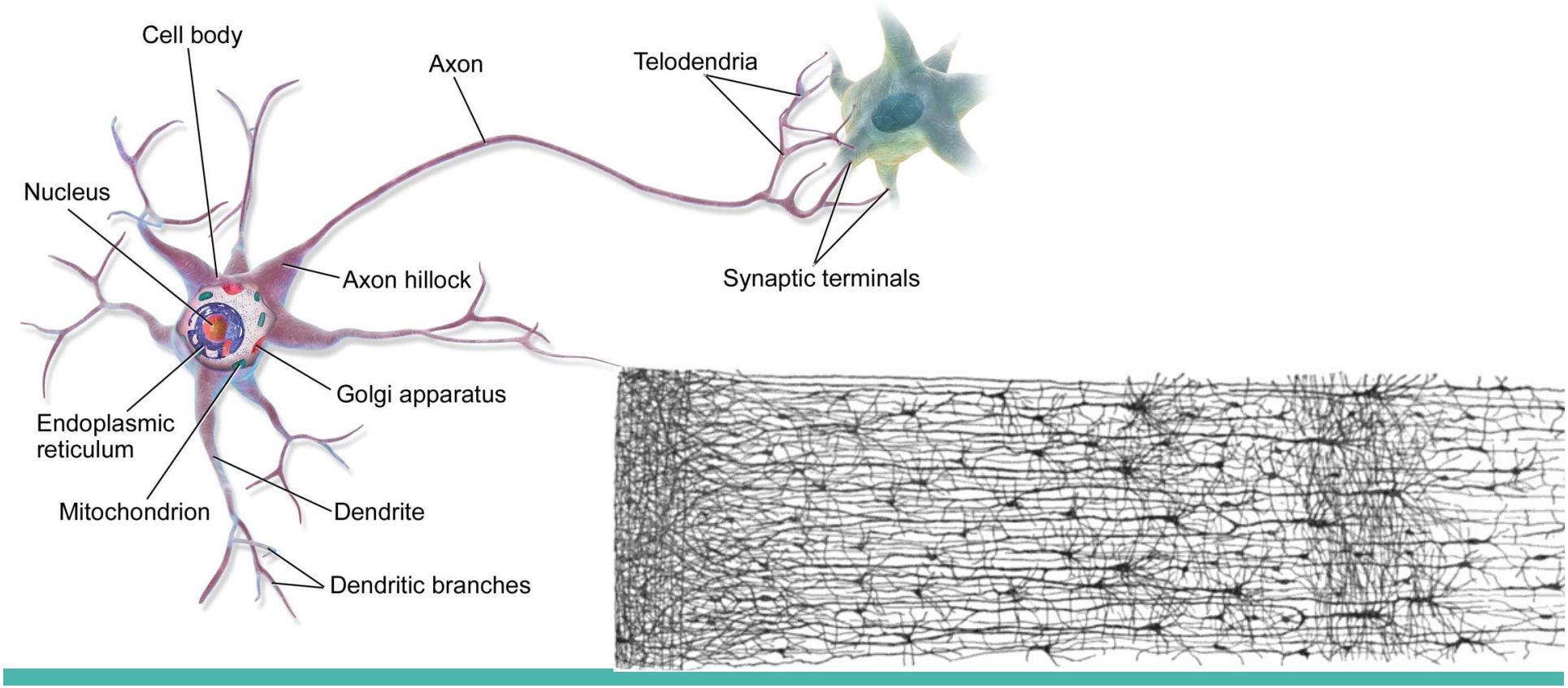
Installing TensorFlow in Anaconda

- https://www.tensorflow.org/beta
- https://towardsdatascience.com/step-by-step-guide-to-install-tensorflow-2-0-67bc73e79b82?sk=5af912c052882d0c8d3651cd76b846a9

```
conda create -n tf_2
conda activate tf_2
conda install pip
pip install tensorflow==2.0.0-beta1
conda install jupyter scikit-learn matplotlib pandas numpy pydot graphviz
python -c "import tensorflow as tf; x = [[2.]]; print('tensorflow version', tf.__version__);
print('hello, {}'.format(tf.matmul(x, x)))"
```

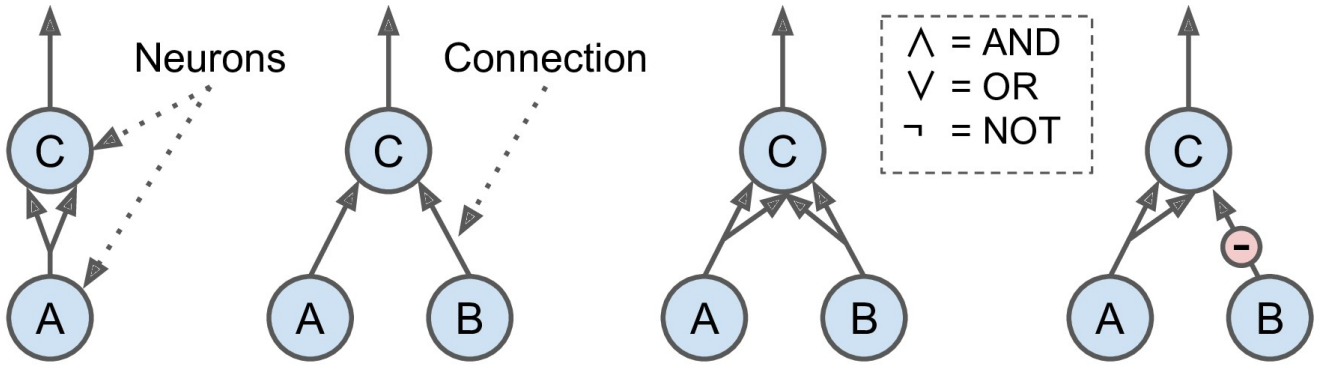
```
(tf_2) C:\Users\thimo>python -c "import tensorflow as tf; x = [[2.]]; print('tensorflow version', tf.__version__); print
('hello, {}'.format(tf.matmul(x, x)))"
tensorflow version 2.0.0-beta1
2019-06-16 15:05:26.285930: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
hello, [[4.]]
```

Introduction to Neural Networks



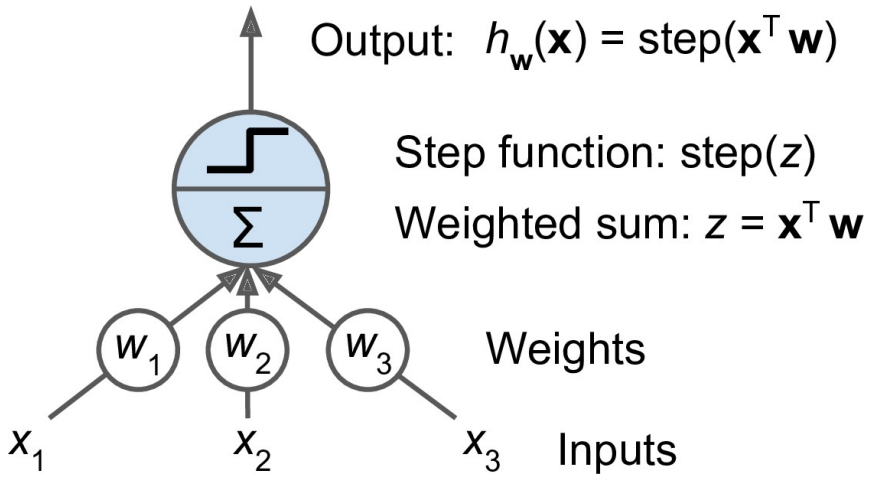
Artificial Neural Networks

- Warren McCulloch and Walter Pitts proposed a very simple model of the biological neuron, which later became known as an *artificial neuron*: it has one or more binary (on/off) inputs and one binary output.
- The artificial neuron simply activates its output when more than a certain number of its inputs are active.
- McCulloch and Pitts showed that even with such a simplified model it is possible to build a network of artificial neurons that computes any logical proposition



Threshold Logic Unit

The *Perceptron* is one of the simplest ANN architectures, invented in 1957 by Frank Rosenblatt. It is based on a slightly different artificial neuron called a *threshold logic unit* (TLU): the inputs and output are now numbers (instead of binary on/off values) and each input connection is associated with a weight. The TLU computes a weighted sum of its inputs ($z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{x}^T \mathbf{w}$), then applies a *step function* to that sum and outputs the result

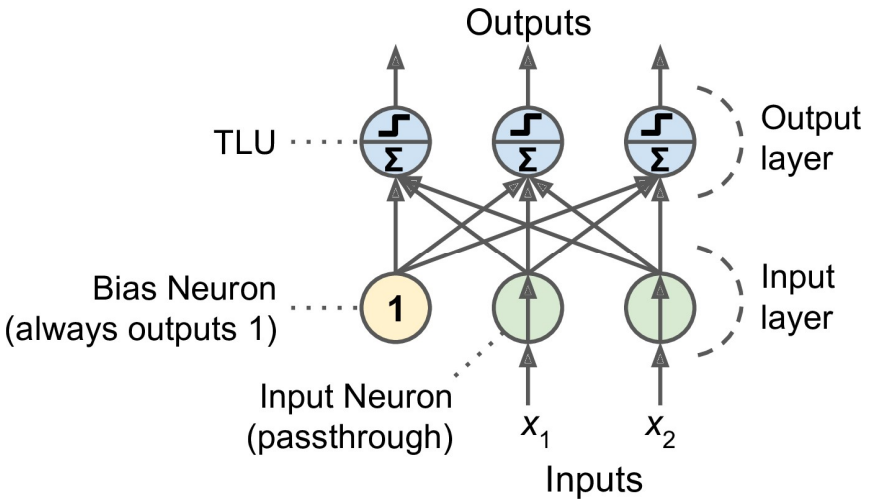


TLU: Simple Binary Classifier

- A single TLU can be used for simple linear binary classification. It computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class or else outputs the negative class (just like a Logistic Regression classifier or a linear SVM).
- For example, you could use a single TLU to classify iris flowers based on the petal length and width.
- Training a TLU in this case means finding the right values for w_0, w_1, and w_2.

Perceptron

- A Perceptron is simply composed of a single layer of TLUs,[6] with each TLU connected to all the inputs.
- When all the neurons in a layer are connected to every neuron in the previous layer (i.e., its input neurons), it is called a *fully connected layer* or a *dense layer*
- This Perceptron can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier



Compute a Perceptron Output Layer

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

- As always, X represents the matrix of input features. It has one row per instance, one column per feature.
- The weight matrix W contains all the connection weights except for the ones from the bias neuron. It has one row per input neuron and one column per artificial neuron in the layer.
- The bias vector b contains all the connection weights between the bias neuron and the artificial neurons. It has one bias term per artificial neuron.
- The function is called the activation function: when the artificial neurons are TLUs, it is a step function.

Training a Perceptron

- The connection weight between two neurons is increased whenever they have the same output.
- Perceptrons are trained using a variant of this rule that takes into account the error made by the network; it reinforces connections that help reduce the error.
- More specifically, the Perceptron is fed one training instance at a time, and for each instance it makes its predictions. For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.

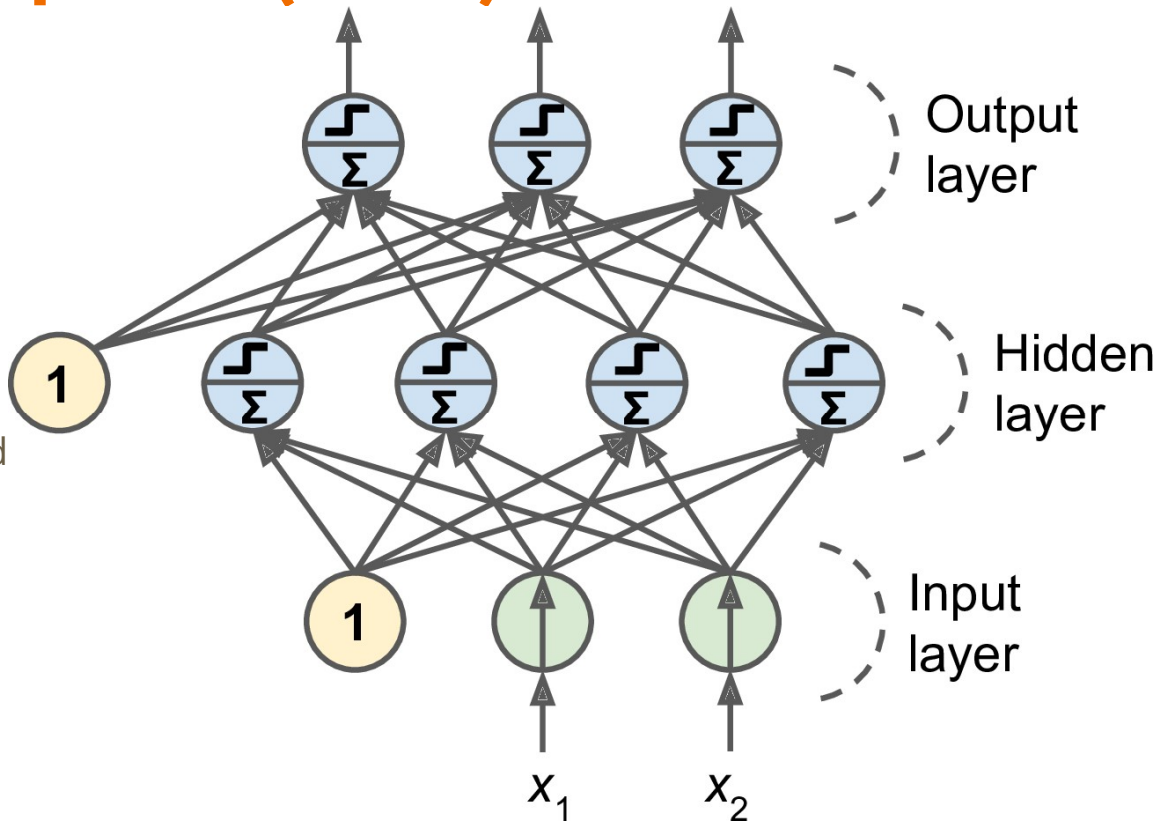
$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta \left(y_j - \hat{y}_j\right) x_i$$

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

Try it! https://playground.tensorflow.org/

Multi-Layer Perceptrons (MLPs)

- A single-layer preceptron has only a linear boundary and does not emit classification probabilities: it is incapable of adapting to complex patterns and it cannot be optimised
- Some of the limitations of Perceptrons can be eliminated by stacking multiple Perceptrons
- Every layer except the output layer includes a bias neuron and is fully connected to the next layer.



Deep Learning

- When an ANN contains a deep stack of hidden layers[8], it is called a *deep neural network* (DNN)
- In 1986, David Rumelhart, Geoffrey Hinton and Ronald Williams published a groundbreaking paper[9] introducing the *backpropagation* training algorithm, which is still used today.
- In short, it is simply Gradient Descent: in just two passes through the network (one forward, one backward), the backpropagation algorithm is able to compute the gradient of the network's error with regards to every single model parameter.
- In other words, it can find out how each connection weight and each bias term should be tweaked in order to reduce the error.

Backpropagation Algorithm

- It handles one mini-batch at a time (for example containing 32 instances each), and **it goes through the full training set multiple times**. Each pass is called an ***epoch***.
- Each mini-batch is **passed to the network's input layer**, which just sends it to the **first hidden layer**. The algorithm then **computes the output of all the neurons in this layer** (for every instance in the mini-batch). The result is passed on to the next layer, its output is computed and passed to the next layer, and so on until we get the output of the last layer, the output layer. **This is the *forward pass***: it is exactly like making predictions, except **all intermediate results are preserved since they are needed for the backward pass**.
- Next, **the algorithm measures the network's output error** (i.e., it uses a loss function that compares the desired output and the actual output of the network, and returns some measure of the error).
- Then it computes **how much each output connection contributed to the error**. This is done analytically by simply applying the *chain rule* (perhaps the most fundamental rule in calculus), which makes this step fast and precise.
- The algorithm then measures **how much of these error contributions came from each connection in the layer** below, again using the chain rule—and so on until the algorithm reaches the input layer. As we explained earlier, **this reverse pass efficiently measures the error gradient across all the connection weights** in the network by propagating the error gradient backward through the network (hence the name of the algorithm).
- Finally, **the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed**.

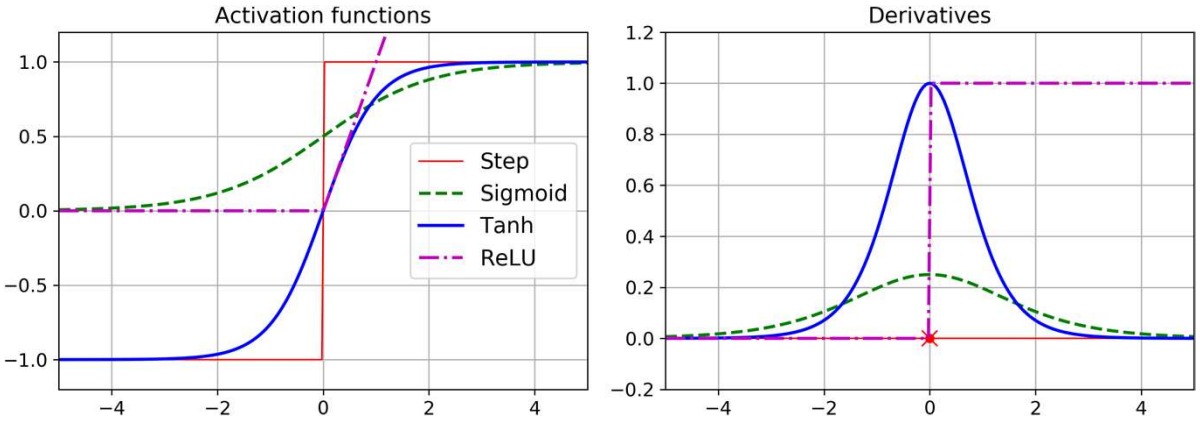
https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html

Replacing the Activation Function

- In order for this algorithm to work properly, the authors made a key change to the MLP's architecture: they replaced the step function with the logistic function, $\sigma(z) = 1 / (1 + \exp(-z))$.
- This was essential because the step function contains only flat segments, so there is no gradient to work with (Gradient Descent cannot move on a flat surface), while the logistic function has a well-defined nonzero derivative everywhere, allowing Gradient Descent to make some progress at every step.
- In fact, the backpropagation algorithm works well with many other *activation functions*, not just the logistic function.

Other activation functions

- The *hyperbolic tangent* function $\tanh(z) = 2\sigma(2z) - 1$
- The Rectified Linear Unit function: $\text{ReLU}(z) = \max(0, z)$
    - It is continuous but unfortunately not differentiable at $z = 0$ (the slope changes abruptly, which can make Gradient Descent bounce around), and its derivative is 0 for $z < 0$. However, in practice it works very well and has the advantage of being fast to compute



Why the Activation Function?

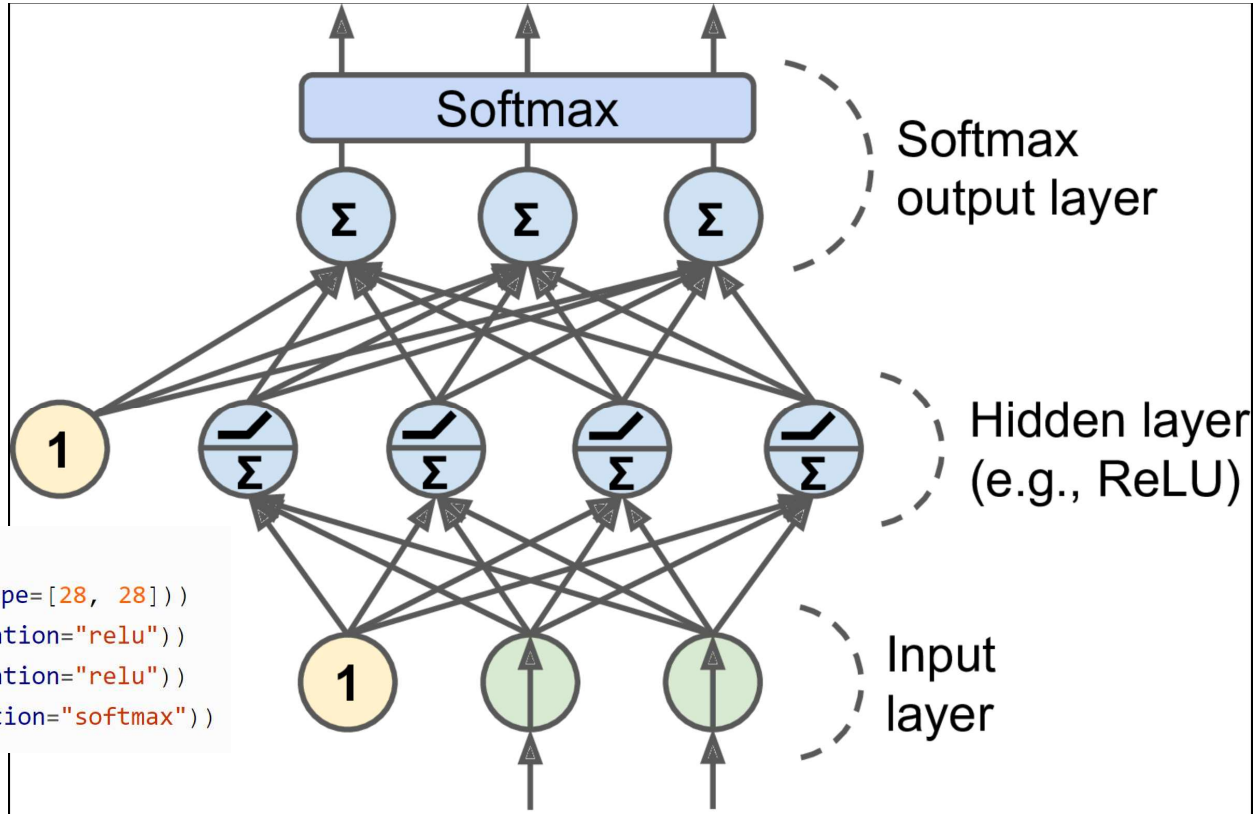
- Why do we need activation functions in the first place?
- If you chain several linear transformations, all you get is a linear transformation. For example, say $f(x) = 2x + 3$ and $g(x) = 5x - 1$, then chaining these two linear functions gives you another linear function: $f(g(x)) = 2(5x - 1) + 3 = 10x + 1$.
- Therefore if you don't have some non-linearity between layers, then even a deep stack of layers is equivalent to a single layer: you cannot solve very complex problems with that.
- Conversely, a large enough DNN with non-linear activations can theoretically approximate any continuous function.

Classification MPLs

- MLPs can be used for classification tasks. For a binary classification problem, you just need a single output neuron using the logistic activation function: the output will be a number between 0 and 1, which you can interpret as the estimated probability of the positive class.
- MLPs can also easily handle multilabel binary classification tasks: you would dedicate one output neuron for each positive class.
- If each instance can belong only to a single class, out of 3 or more possible classes (e.g., classes 0 through 9 for digit image classification), then you need to have one output neuron per class, and you should use the *softmax* activation function for the whole output layer (to sum up probs to 1)
- Regarding the loss function, since we are predicting probability distributions, the cross-entropy (Log Loss) is a good choice.

Multiclass MLP NN for Classification

The softmax function will ensure that all the estimated probabilities are between 0 and 1 and that they add up to one (which is required if the classes are exclusive).




```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
Trainable params: 266,610
```

Regression MLPs

- MLPs can be used for regression tasks. If you want to predict a single value (e.g., the price of a house given many of its features), then you just need a single output neuron: its output is the predicted value. For multivariate regression (i.e., to predict multiple values at once), you need one output neuron per output dimension.
- In general, when building an MLP for regression, you do not want to use any activation function for the output neurons, so they are free to output any range of values. However, if you want to guarantee that the output will always be positive, then you can use the ReLU activation function.
- The loss function to use during training is typically the mean squared error, but if you have a lot of outliers in the training set, you may prefer to use the mean absolute error instead.

Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Typical Classification MLP Architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

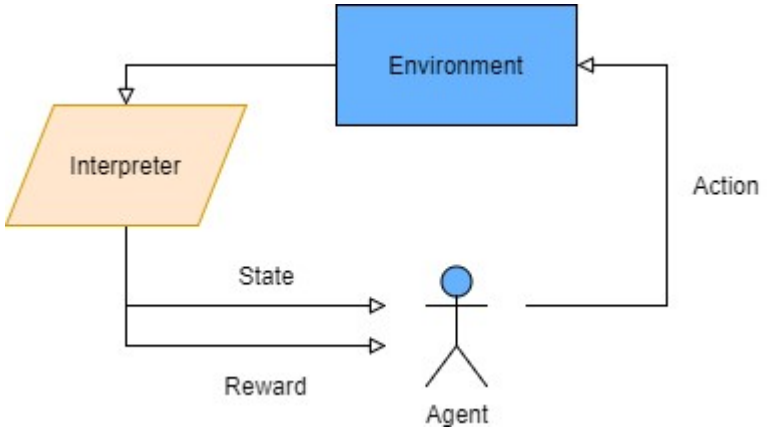
Lab: Using NNs in Keras for Classifiers and Regressions

http://localhost:8888/notebooks/handson-ml2/10_neural_nets_with_keras.ipynb

Reinforcement Learning

https://github.com/thimotyb/real-world-machine-learning/blob/python3/ReinforcementLearning_NChain_OpenGym.ipynb

- Reinforcement learning can be considered the third genre of the machine learning triad – unsupervised learning, supervised learning and reinforcement learning. In supervised learning, we supply the machine learning system with curated (x, y) training pairs, where the intention is for the network to learn to map x to y.
- In reinforcement learning, we create an *agent* which performs *actions* in an *environment* and the agent receives various *rewards* depending on what *state* it is in when it performs the action. In other words, an agent explores a kind of game, and it is trained by trying to maximize rewards in this game.



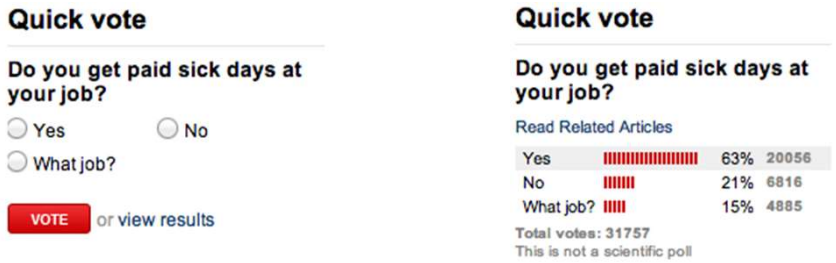
Foundations of Statistics

Exploratory analysis to inference

- Sampling is natural.
- Think about sampling something you are cooking - you taste (examine) a small part of what you're cooking to get an idea about the dish as a whole.
- When you taste a spoonful of soup and decide the spoonful you tasted isn't salty enough, that's *exploratory analysis*.
- If you generalize and conclude that your entire soup needs salt, that's an *inference*.
- For your inference to be valid, the spoonful you tasted (the sample) needs to be *representative* of the entire pot (the population).
    - If your spoonful comes only from the surface and the salt is collected at the bottom of the pot, what you tasted is probably not representative of the whole pot.
    - If you first stir the soup thoroughly before you taste, your spoonful will more likely be representative of the whole pot.

Sampling bias

- Non-response: If only a small fraction of the randomly sampled people choose to respond to a survey, the sample may no longer be representative of the population.
- Voluntary response: Occurs when the sample consists of people who volunteer to respond because they have strong opinions on the issue. Such a sample will also not be representative of the population.



- Convenience sample: Individuals who are easily accessible are more likely to be included in the sample.

Explanatory and Response Variables

- To identify the explanatory variable in a pair of variables, identify which of the two is suspected of affecting the other:

$$\text{explanatory variable} \xrightarrow{\textit{might affect}} \text{response variable}$$

- Labeling variables as explanatory and response does not guarantee the relationship between the two is actually causal, even if there is an association identified between the two variables. We use these labels only to keep track of which variable we suspect affects the other.

Obtaining Good Samples

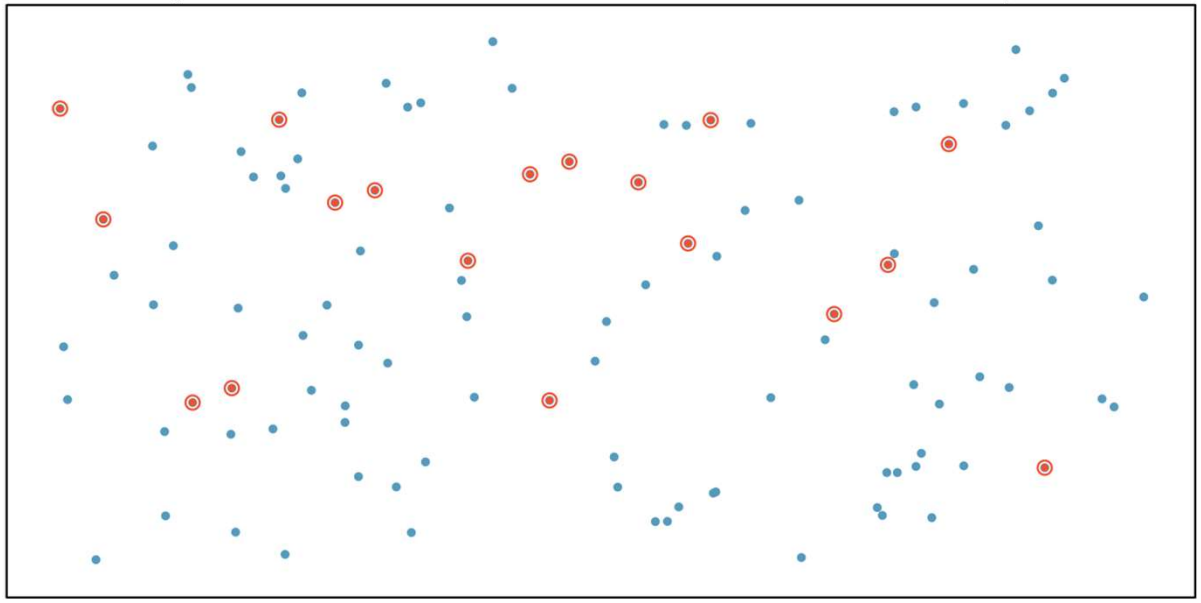
- Almost all statistical methods are based on the notion of implied randomness.

- If observational data are not collected in a random framework from a population, these statistical methods -- the estimates and errors associated with the estimates -- are not reliable.

- Most commonly used random sampling techniques are *simple*, *stratified*, and *cluster* sampling.

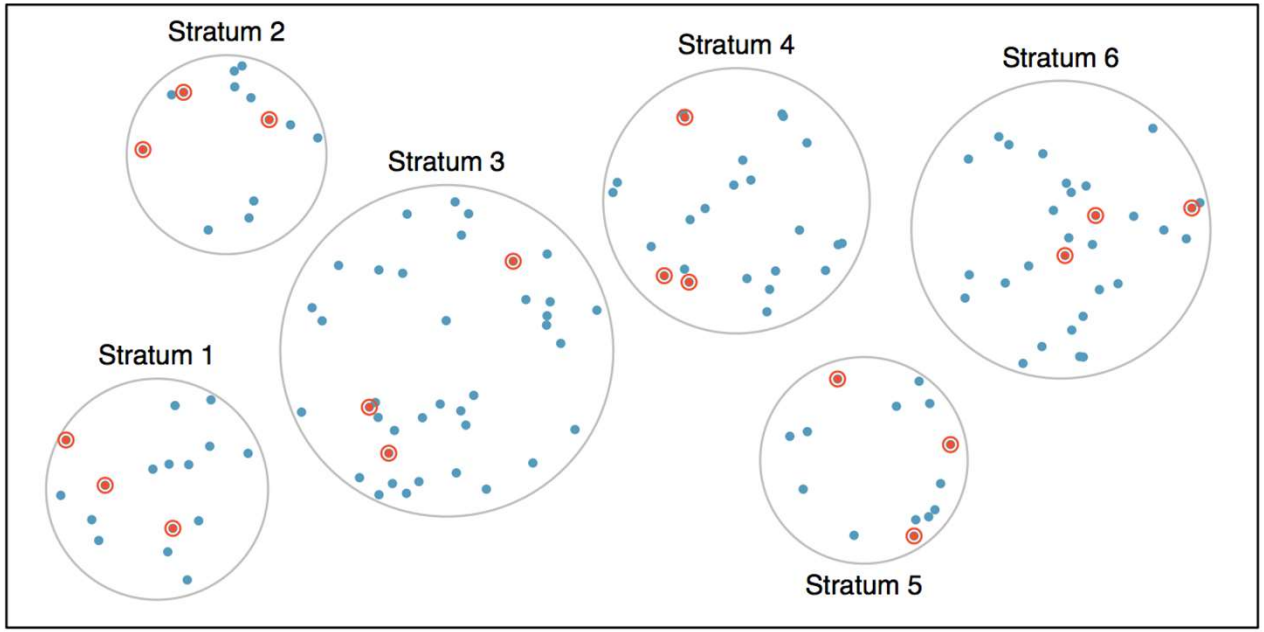
Simple Random Sample

Randomly select cases from the population, where there is no implied connection between the points that are selected.



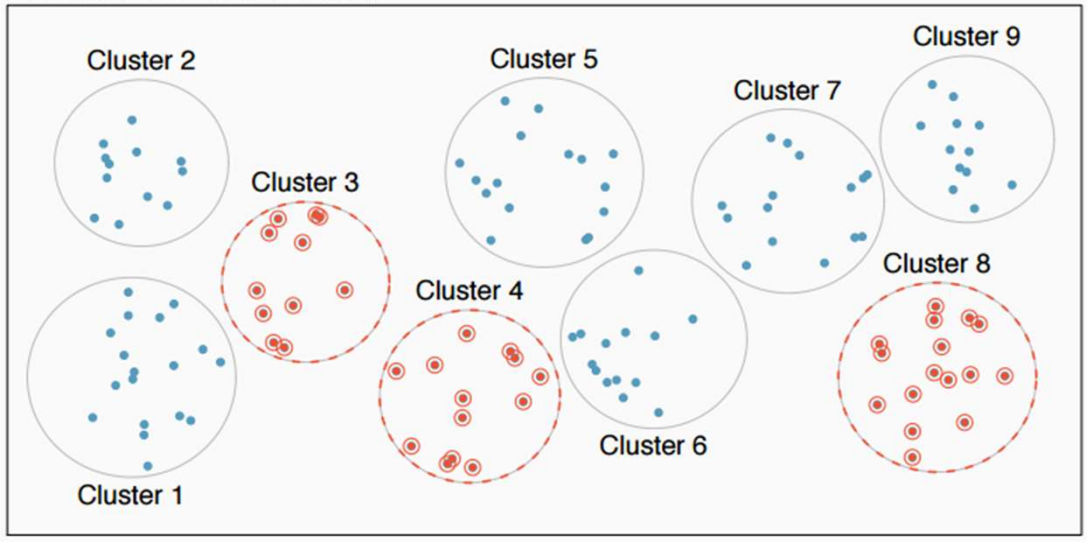
Stratified Sample

Strata are made up of similar observations. We take a simple random sample from each stratum.



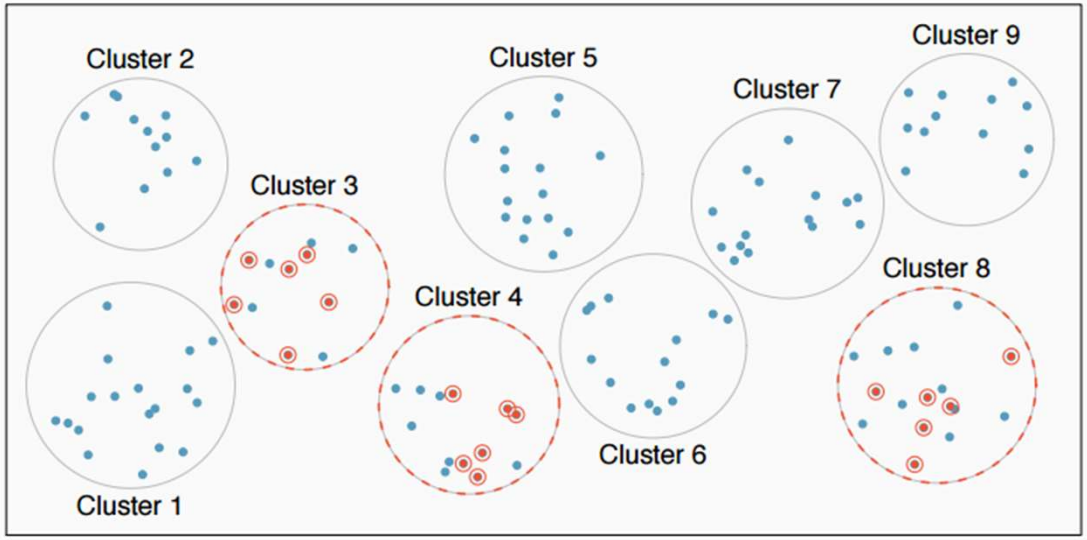
Cluster Sample

Clusters are usually not made up of homogeneous observations. We take a simple random sample of clusters, and then sample all observations in that cluster. Usually preferred for economical reasons.



Multistage Sample

Clusters are usually not made up of homogeneous observations. We take a simple random sample of clusters, and then take a simple random sample of observations from the sampled clusters



Scatterplot

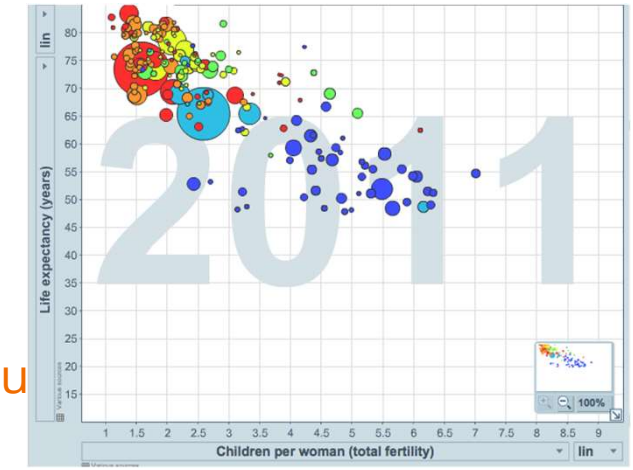
Scatterplots are useful for visualizing the relationship between two numerical variables.

Do life expectancy and total fertility appear to be *associated* or *independent*?

They appear to be linearly and negatively associated: as fertility increases, life expectancy decreases.

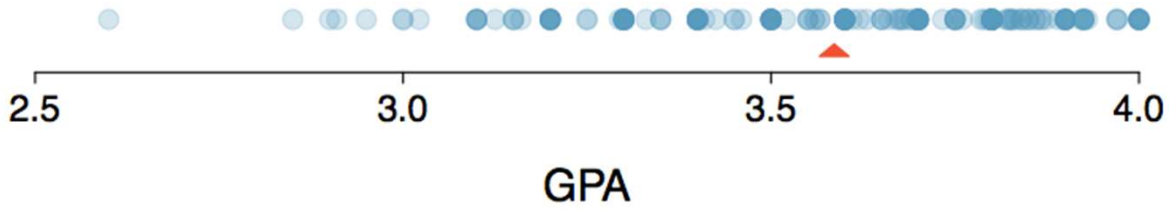
Was the relationship the same throughou the years, or did it change?

The relationship changed over the years.



http://www.gapminder.org/world

Dot Plots & Mean



The *mean*, also called the *average* (marked with a triangle in the above plot), is one way to measure the center of a *distribution* of data.

The mean GPA is 3.59.

Mean

The *sample mean*, denoted as $\bar{x}$, can be calculated as

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n},$$

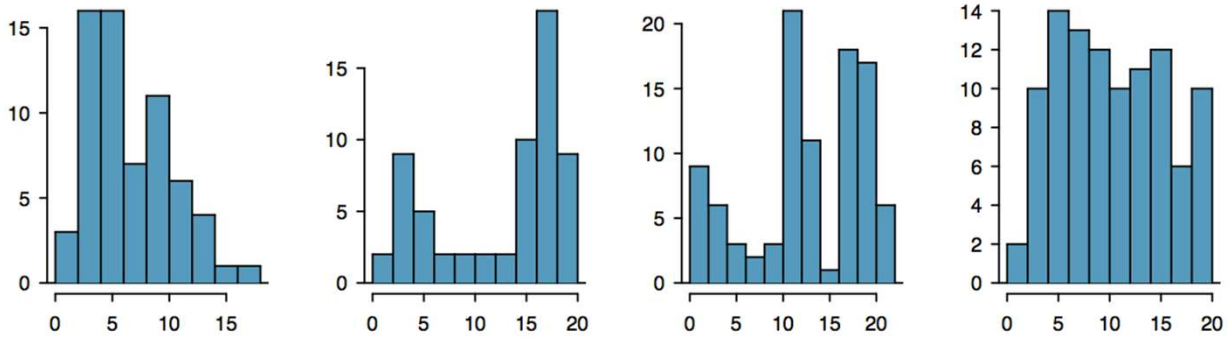
where x_1, x_2, ..., x_n represent the *n* observed values.

The *population mean* is also computed the same way but is denoted as *μ*. It is often not possible to calculate μ since population data are rarely available.

The sample mean is a *sample statistic*, and serves as a *point estimate* of the population mean. This estimate may not be perfect, but if the sample is good (representative of the population), it is usually a pretty good estimate.

Shape of a Distribution: Modality

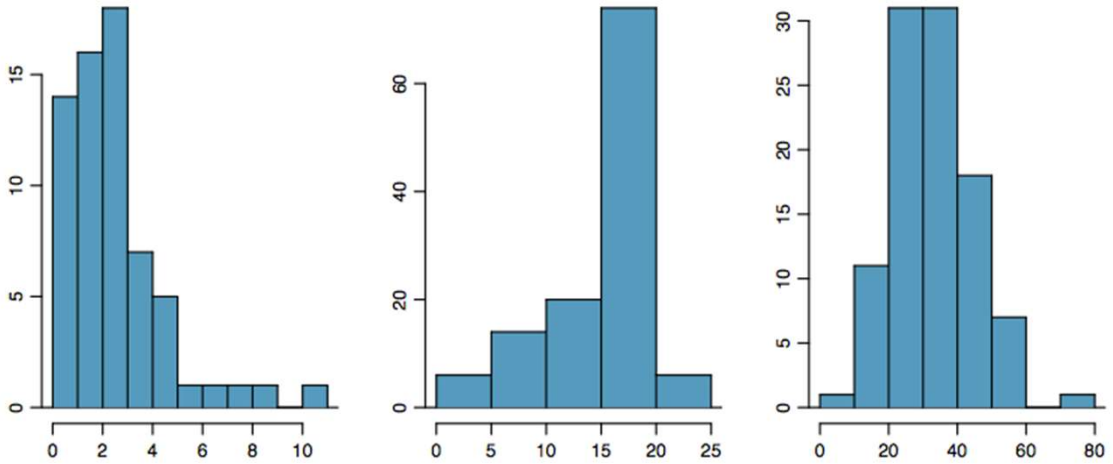
Does the histogram have a single prominent peak (*unimodal*), several prominent peaks (*bimodal/multimodal*), or no apparent peaks (*uniform*)?



Note: In order to determine modality, step back and imagine a smooth curve over the histogram -- imagine that the bars are wooden blocks and you drop a limp spaghetti over them, the shape the spaghetti would take could be viewed as a smooth curve.

Shape of a Distribution: Skewness

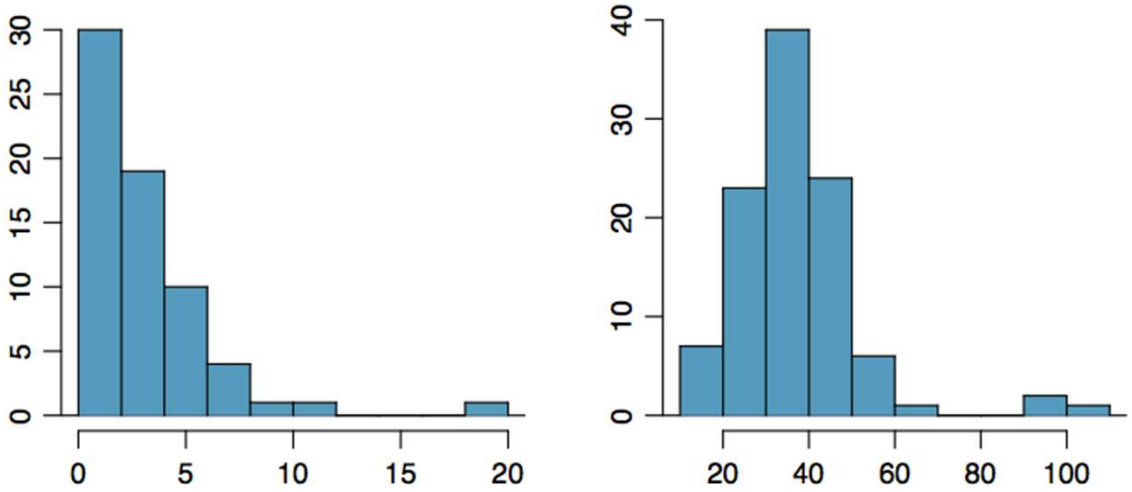
Is the histogram *right skewed*, *left skewed*, or *symmetric*?



Note: Histograms are said to be skewed to the side of the long tail.

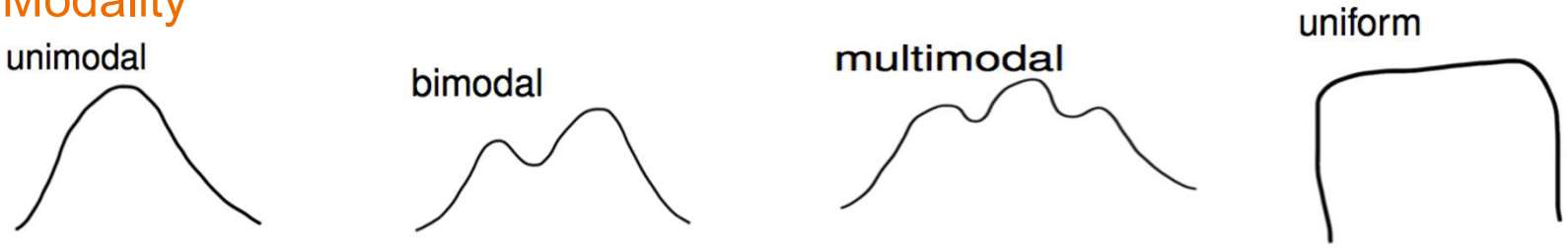
Shape of a Distribution: Unusual Observations

Are there any unusual observations or potential *outliers*?

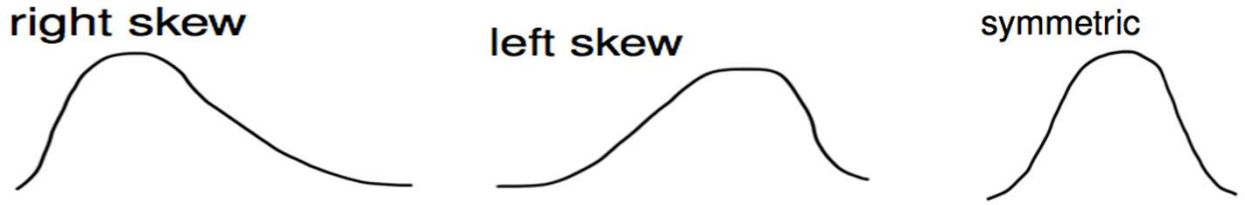


Commonly observed shapes of distributions

Modality



Skewness

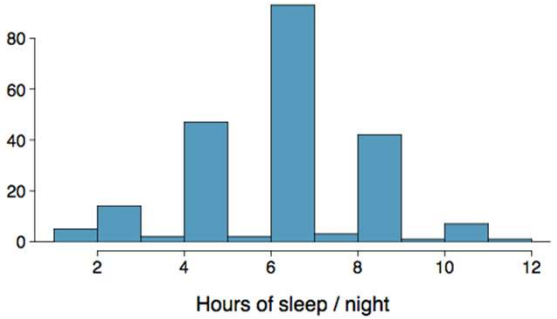


Variance

Variance is roughly the average squared deviation from the mean.

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

- The sample mean is $\bar{x} = 6.71$, and the sample size is n = 217.
- The variance of amount of sleep students get per night can be calculated as:



$$s^2 = \frac{(5-6.71)^2 + (9-6.71)^2 + \cdots + (7-6.71)^2}{217-1} = 4.11\ hours^2$$

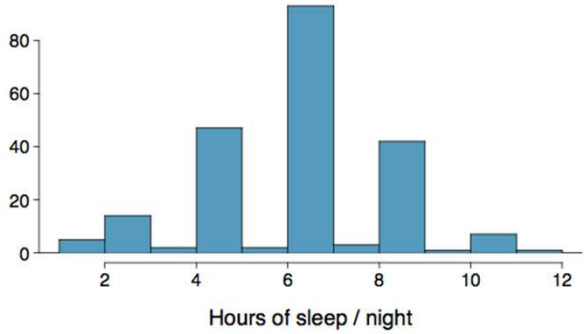
Standard Deviation

The *standard deviation* is the square root of the variance, and has the same units as the data.

$$s = \sqrt{s^2}$$

- The standard deviation of amount of sleep students get per night can be calculated as:

  $s = \sqrt{4.11} = 2.03\ hours$



- We can see that all of the data are within 3 standard deviations of the mean.

Median

The *median* is the value that splits the data in half when ordered in ascending order.

$$0, 1, 2, 3, 4$$

If there are an even number of observations, then the median is the average of the two values in the middle.

$$0, 1, \underline{2, 3}, 4, 5 \rightarrow \frac{2+3}{2} = 2.5$$

Since the median is the midpoint of the data, 50% of the values are below it. Hence, it is also the 50th percentile.

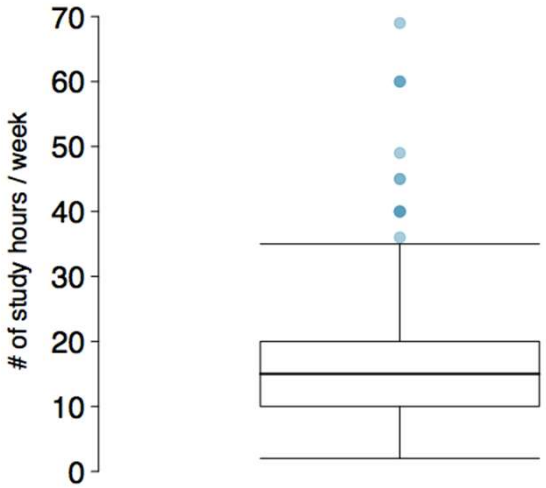
Q1, Q3, and IQR

- The 25th percentile is also called the first quartile, *Q1*.
- The 50th percentile is also called the median.
- The 75th percentile is also called the third quartile, *Q3*.
- Between Q1 and Q3 is the middle 50% of the data. The range these data span is called the *interquartile range*, or the *IQR*.

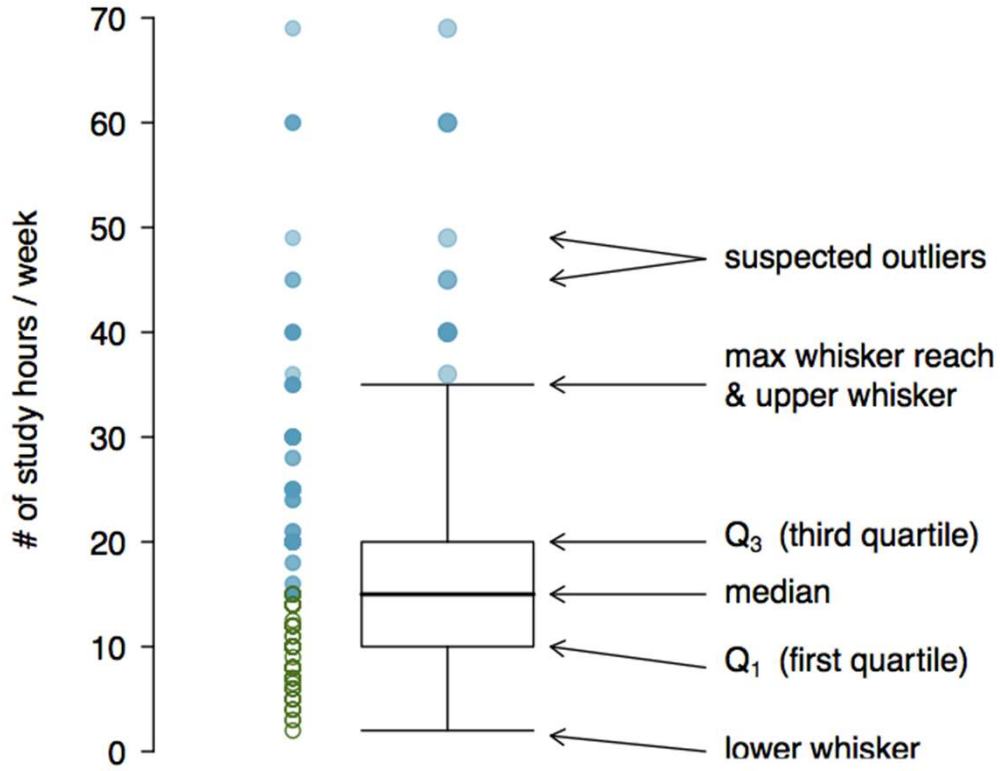
$$IQR = Q3 - Q1$$

Box Plot

The box in a *box plot* represents the middle 50% of the data, and the thick line in the box is the median.



Anatomy of a Box Plot



Whiskers and Outliers

Whiskers of a box plot can extend up to 1.5 x IQR away from the quartiles.

max upper whisker reach = Q3 + 1.5 x IQR

max lower whisker reach = Q1 - 1.5 x IQR

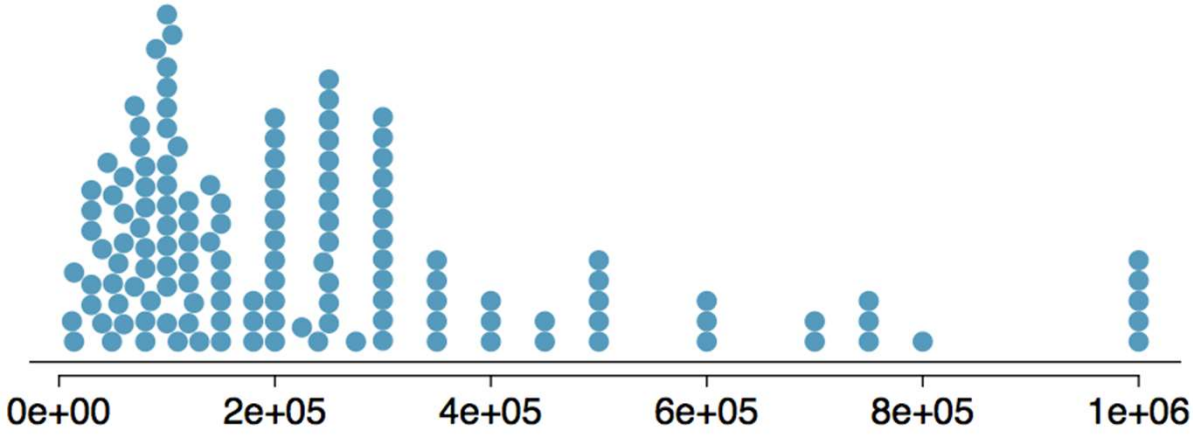
IQR: 20 - 10 = 10

max upper whisker reach = 20 + 1.5 x 10 = 35

max lower whisker reach = 10 - 1.5 x 10 = -5

A potential *outlier* is defined as an observation beyond the maximum reach of the whiskers. It is an observation that appears extreme relative to the rest of the data.

Robust Statistics

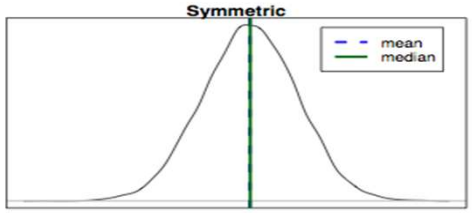


Annual Household Income

	robust		not robust	
scenario	median	IQR	$\bar{x}$	s
original data	190K	200K	245K	226K
move largest to $10 million	190K	200K	309K	853K
move smallest to $10 million	200K	200K	316K	854K

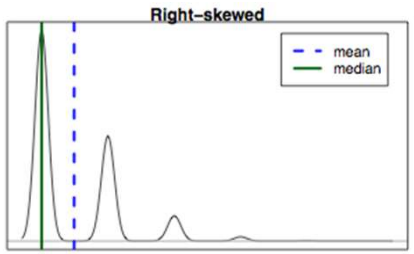
Mean vs. Median

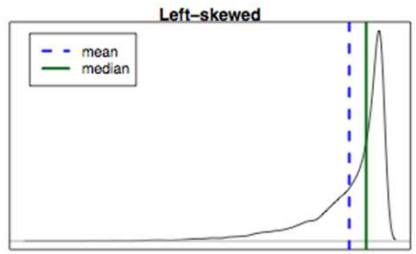
If the distribution is symmetric, center is often defined as the mean: mean ~ median



If the distribution is skewed or has extreme outliers, center is often defined as the median

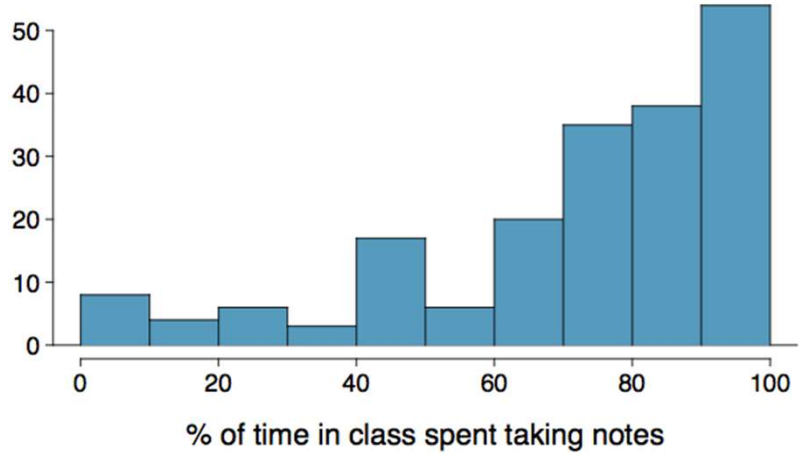
- Right-skewed: mean > median
- Left-skewed: mean < median





Practice

Which is most likely true for the distribution of percentage of time actually spent taking notes in class versus on Facebook, Twitter, etc.?



(a) mean > median

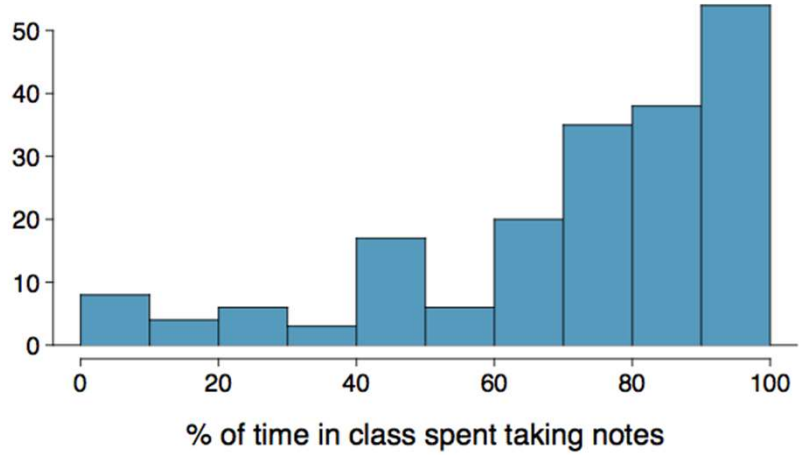
(b) mean ~ median

(c) mean < median

(d) impossible to tell

Practice

Which is most likely true for the distribution of percentage of time actually spent taking notes in class versus on Facebook, Twitter, etc.?



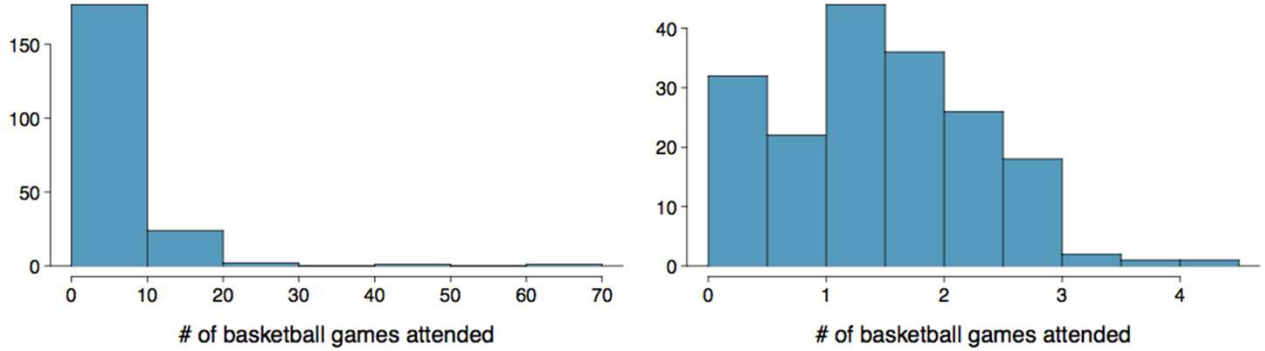
median: 80%
mean: 76%

(a) mean > median
(c) mean < median
(b) mean ~ median
(d) impossible to tell

Extremely Skewed Data

When data are extremely skewed, transforming them might make modeling easier. A common transformation is the log transformation.

The histograms on the left shows the distribution of number of basketball games attended by students. The histogram on the right shows the distribution of log of number of games attended.



Categorical data: Contingency Tables

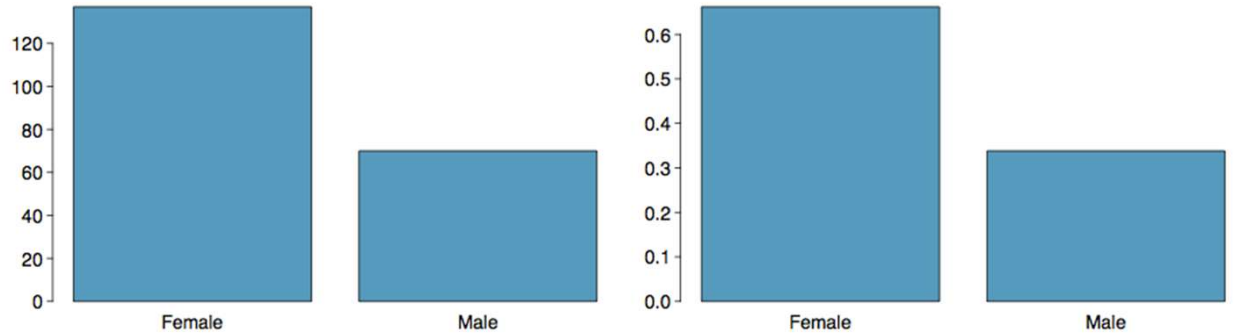
A table that summarizes data for two categorical variables is called a *contingency table*.

The contingency table below shows the distribution of students' genders and whether or not they are looking for a spouse while in college.

		looking for spouse		
		No	Yes	Total
gender	Female	86	51	137
	Male	52	18	70
	Total	138	69	207

Bar Plots

A *bar plot* is a common way to display a single categorical variable. A bar plot where proportions instead of frequencies are shown is called a *relative frequency bar plot*.



How are bar plots different than histograms?

Bar plots are used for displaying distributions of categorical variables, while histograms are used for numerical variables. The x-axis in a histogram is a number line, hence the order of the bars cannot be changed, while in a bar plot the categories can be listed in any order (though some orderings make more sense than others, especially for ordinal variables.)

Choosing the Appropriate Proportion

Does there appear to be a relationship between gender and whether the student is looking for a spouse in college?

		looking for spouse		
		No	Yes	Total
gender	Female	86	51	137
	Male	52	18	70
	Total	138	69	207

To answer this question we examine the row proportions:

- % Females looking for a spouse: 51 / 137 ~ 0.37
- % Males looking for a spouse: 18 / 70 ~ 0.26