
Practical Machine Learning

An introductory course
in Python and R

Course Agenda (I)

- Starting with data:
 - Inspecting data; Types of Data
 - A crash course in R
 - Visualize Data in R/RStudio; types of visualisation
 - A crash course in Python
 - Visualize Data in Python
 - 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 and R
 - A framework for regression analysis in R

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

Course Agenda (iv)

- Scalable Machine Learning
 - An introduction to big data architectures: Spark on R;
 - A brief case study
- A comprehensive example of Machine Learning implementation
 - Taxi trip data
- A collaborative exercise



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
- R Real World Examples: <https://github.com/thimotyb/rwml-R>
- R Labs on Linear Regression: <https://github.com/thimotyb/oiLabs-base-R>

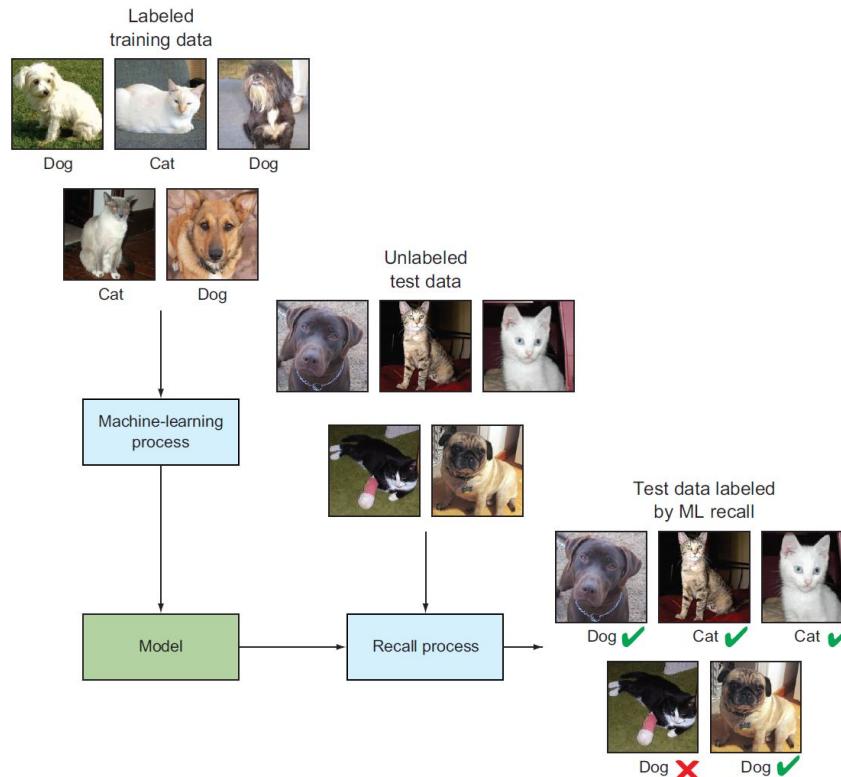
Go on your /home/user/git folder

```
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

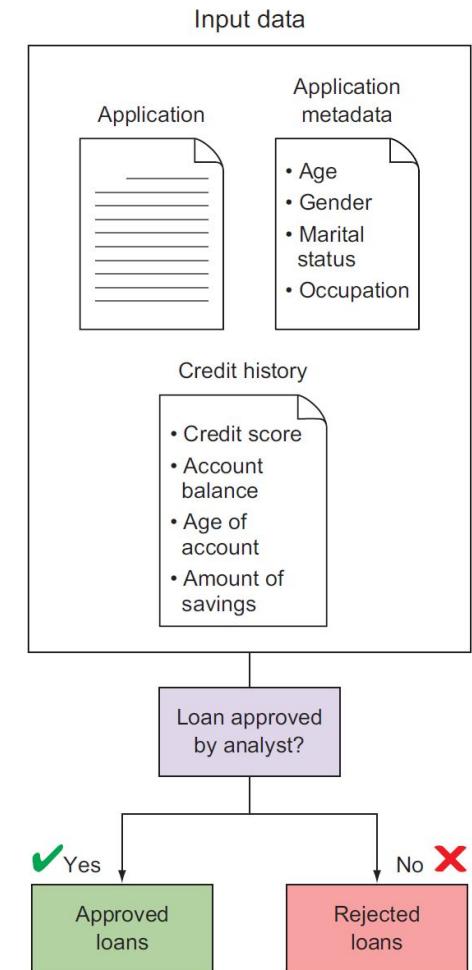


Some ML use cases for supervised learning

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

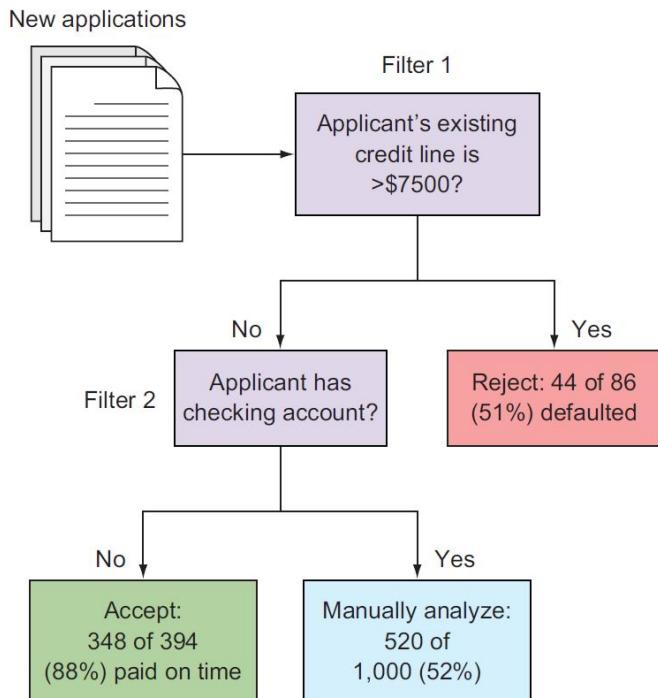
A first example: microlending process

- 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.

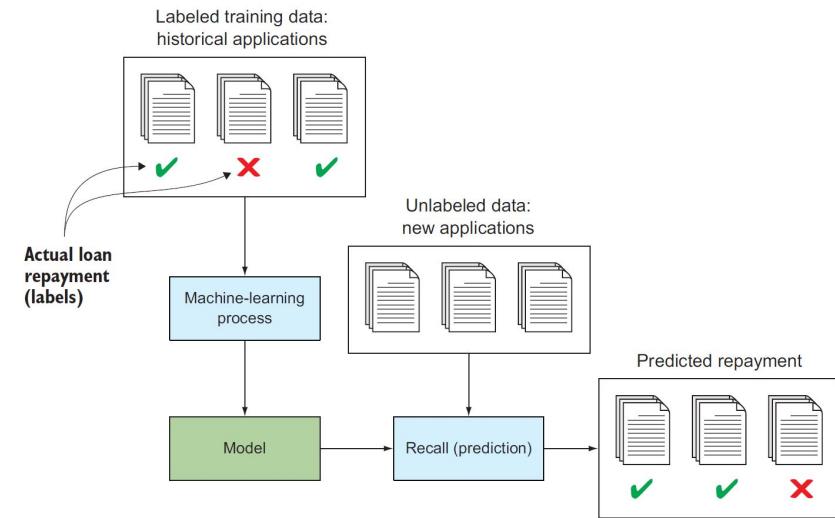
To determine whether to accept each new loan application, ML uses historical training data to predict the best course of action for each new application.

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.



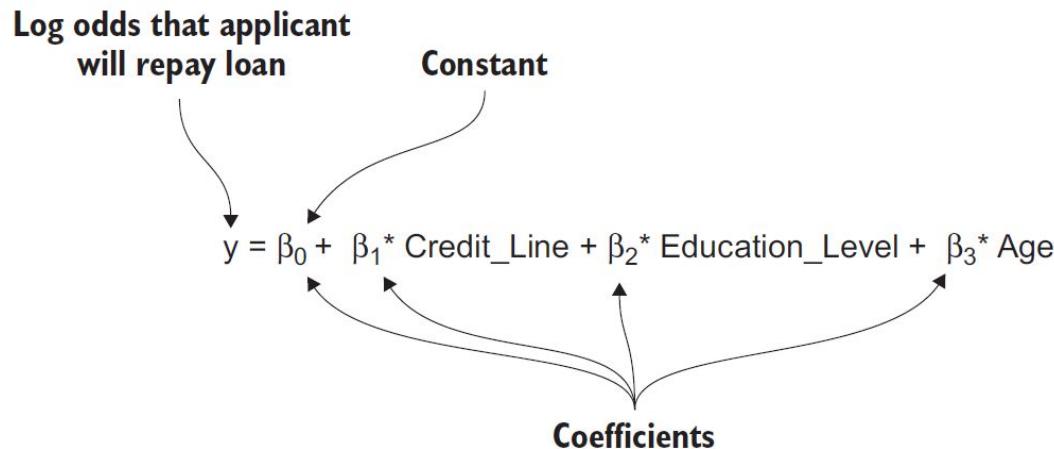
Selecting a ML algorithm

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

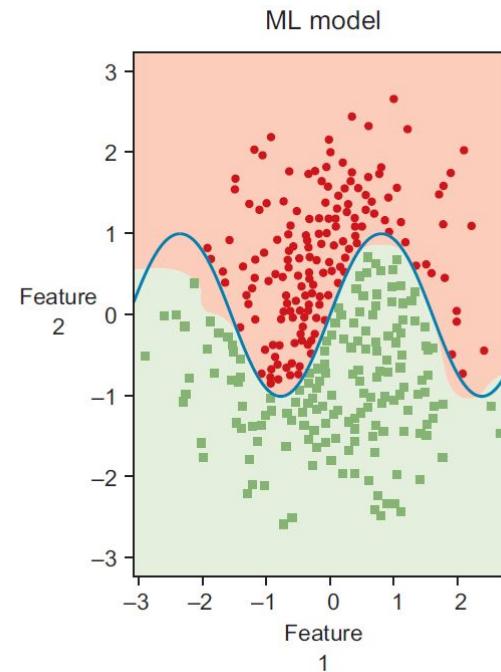
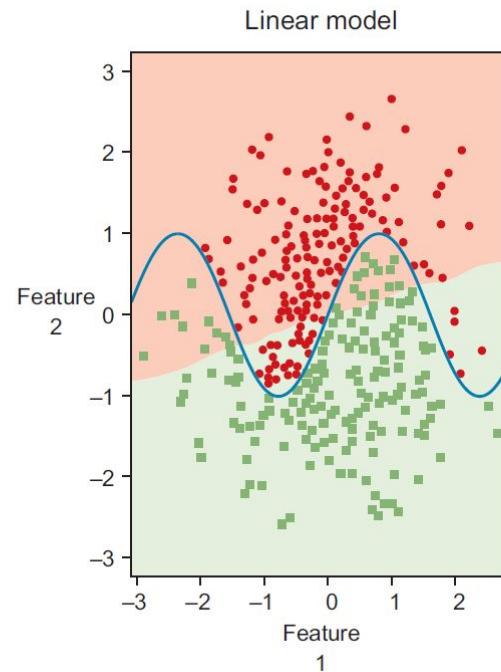
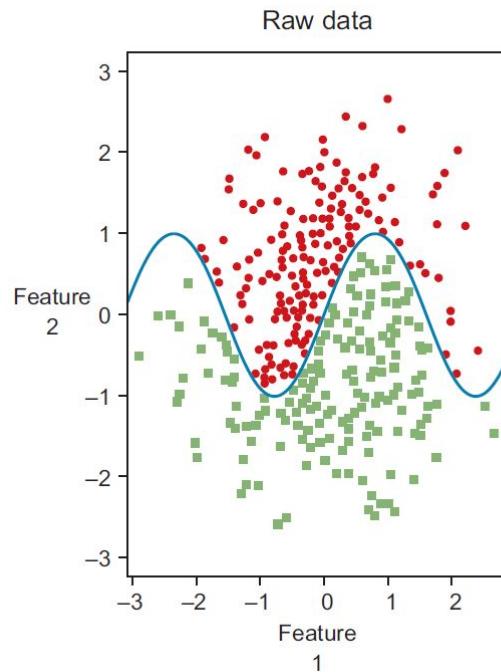
- using a simple parametric model: parametric models use simple, fixed equations to express the relationship between the outcome and the inputs. Data is then used to learn the best values of the unknown terms in the equation: linear regression, logistic regression, ...
- using a nonparametric set of 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.

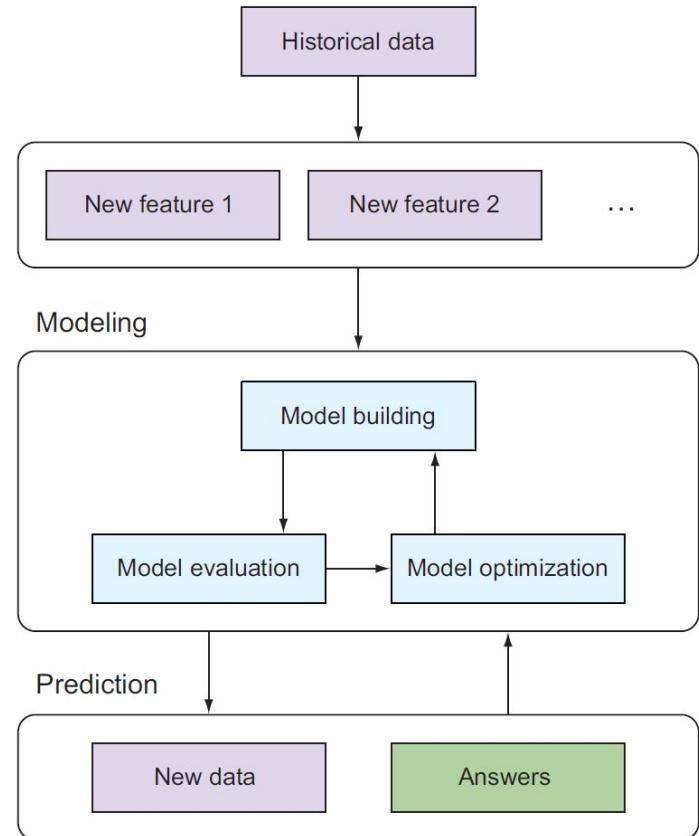
Fast—ML can generate answers in a matter of milliseconds as new data streams in, allowing systems to react in real time.

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.

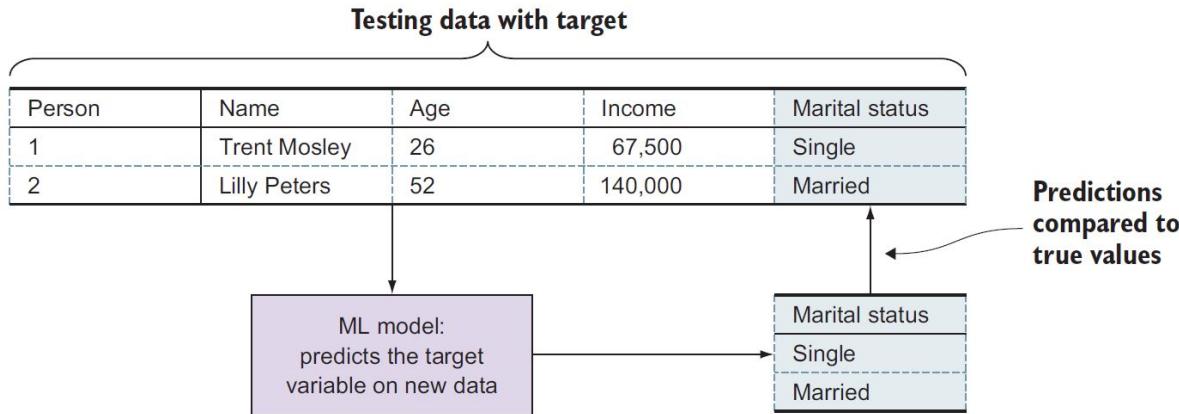
Scalable—As your business grows, ML easily scales to handle increased data rates. Some ML algorithms can scale to handle large amounts of data on many machines in the cloud.

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 fraudulent 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)?

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

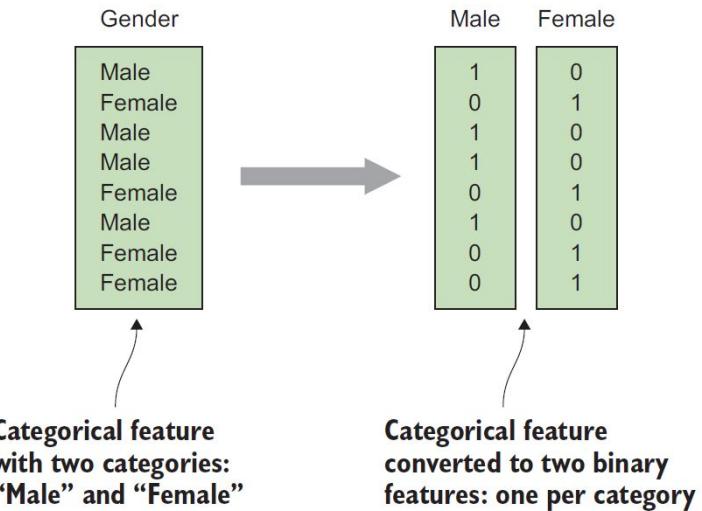
- The first issue to deal with is handling categorical features
- A feature is categorical if values can be placed in buckets and the order of values isn't important

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

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/O2. 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



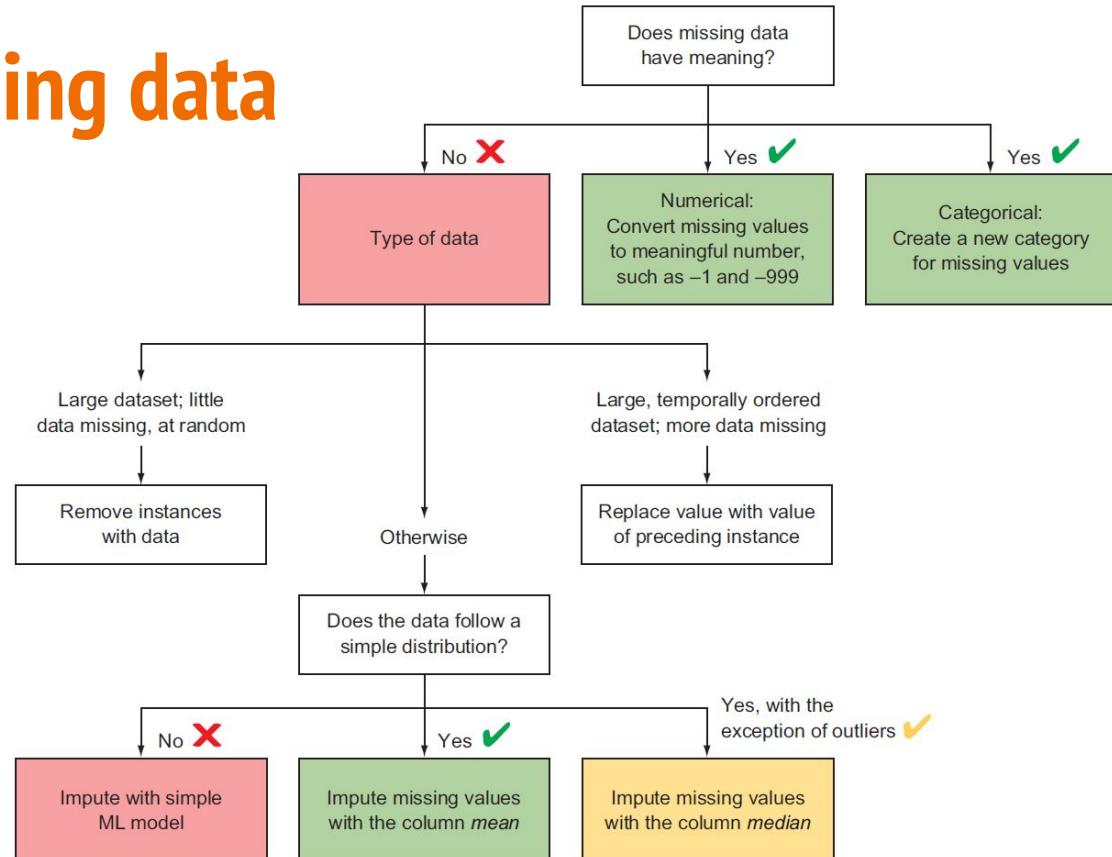
Conversion of categorical features in binary form

- 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
```

Dealing with missing data

- Usually NaN entries
- See pandas exercise



Lab 1: Introducing Python and Pandas

Setting up Canopy and running the example

Let's analyze the 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])]
```

A Python primer

<https://docs.python.org/2/tutorial/>

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

A Pandas primer

- Pandas tutorial: <https://github.com/thimotyb/pandas-cookbook>
 - Reading data from a csv file
 - Handling dataframes and plotting
 - Data and String manipulation
 - Data cleaning
- Some pandas exercises: https://github.com/thimotyb/pandas_exercises

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.

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

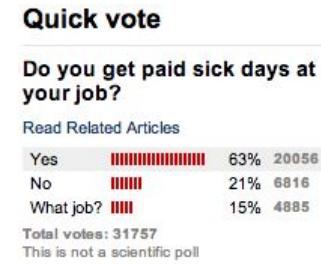
- **Non-response:** If only a small fraction of the randomly sampled people choose to respond to a survey, the sample may no longer be representative of the population.
- **Voluntary response:** Occurs when the sample consists of people who volunteer to respond because they have strong opinions on the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

Quick vote

Do you get paid sick days at your job?

Yes No
 What job?

VOTE or view results



- **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:

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

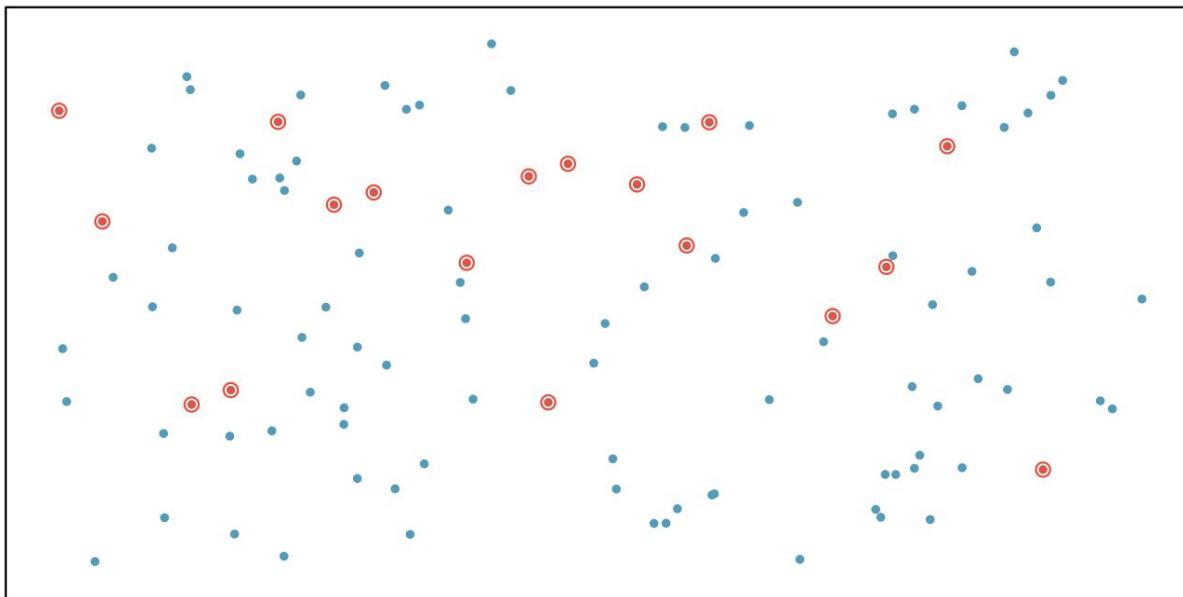
- Labeling variables as explanatory and response does not guarantee the relationship between the two is actually causal, even if there is an association identified between the two variables. We use these labels only to keep track of which variable we suspect affects the other.

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.

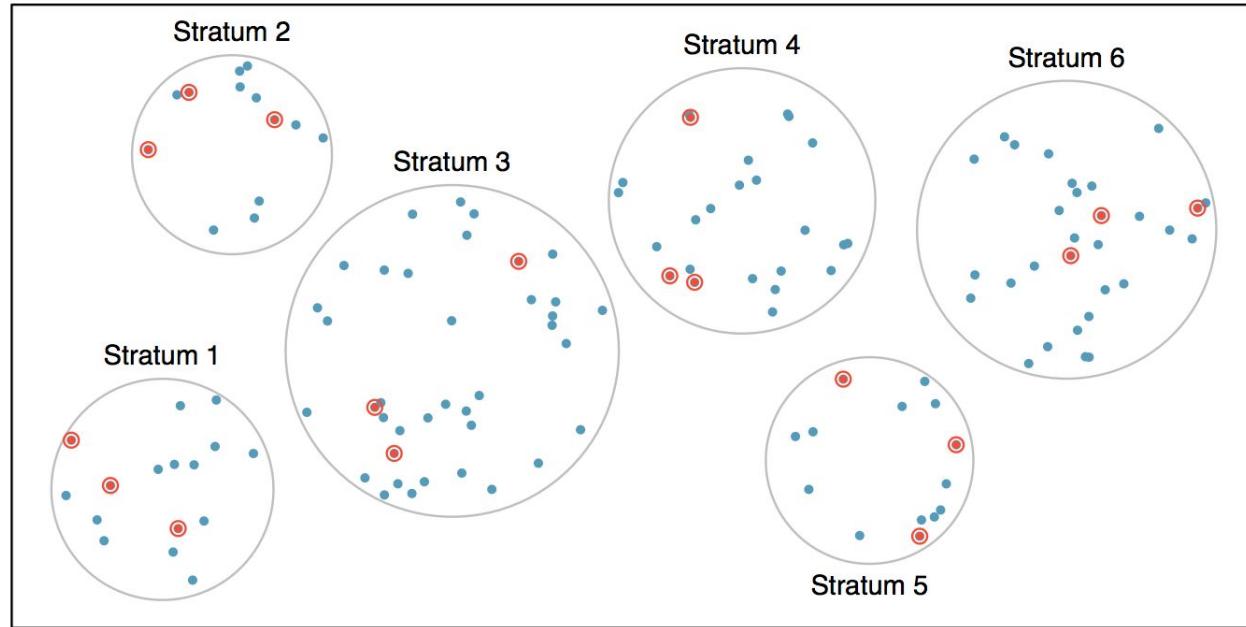
Simple Random Sample

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



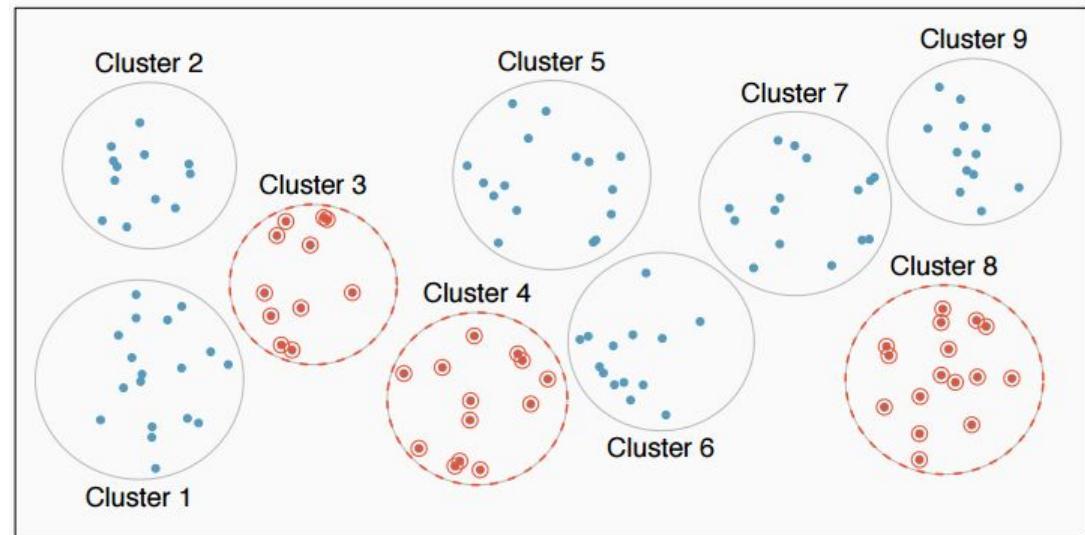
Stratified Sample

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



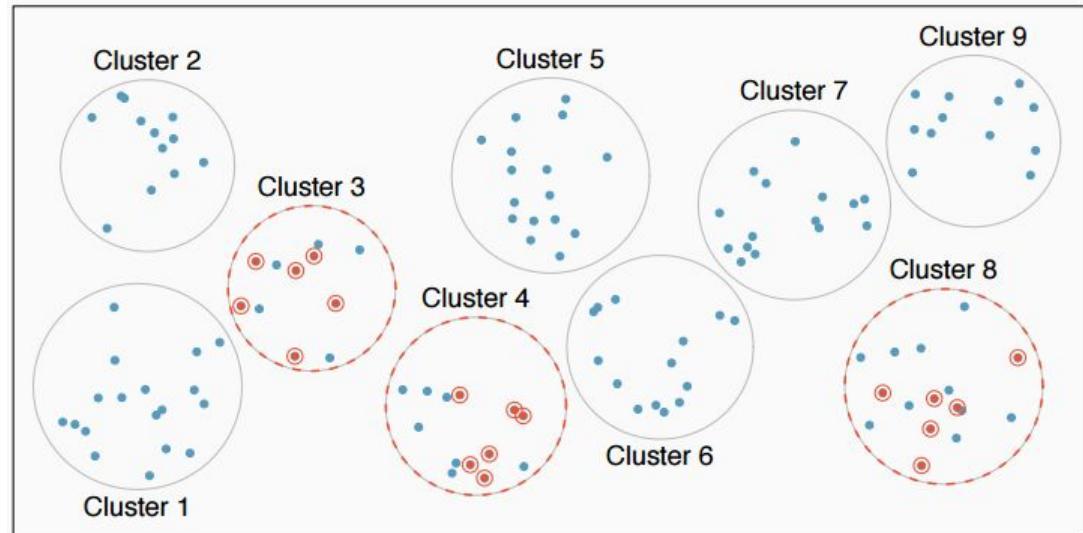
Cluster Sample

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

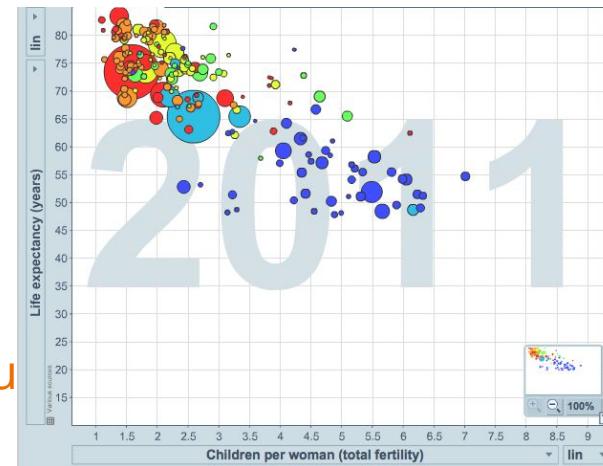
Scatterplots are useful for visualizing the relationship between two numerical variables.

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 throughout 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

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

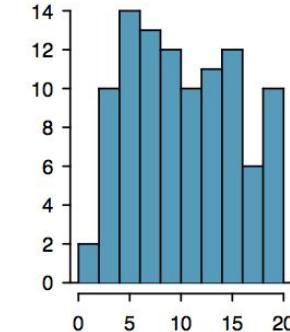
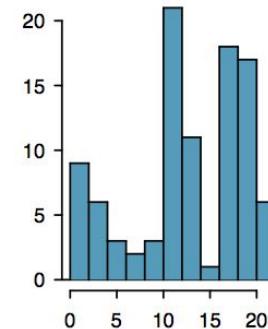
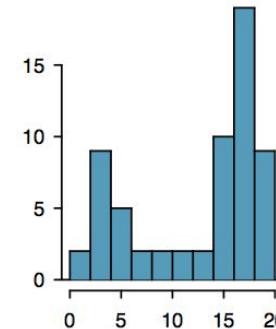
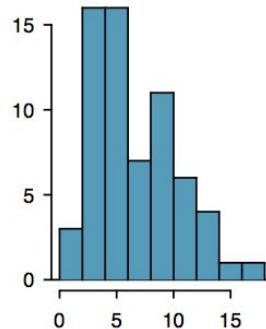
where x_1, x_2, \dots, x_n represent the n observed values.

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

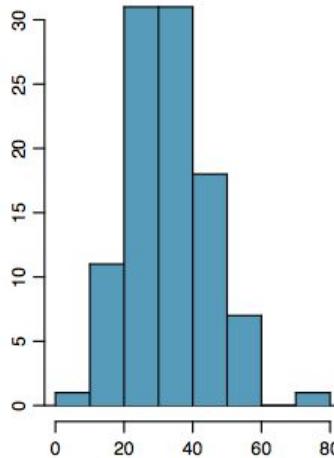
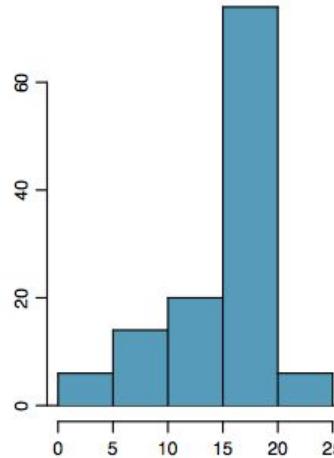
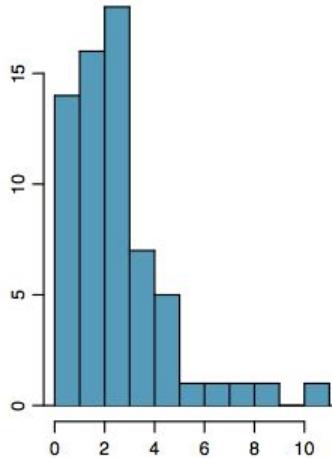
Does the histogram have a single prominent peak (*unimodal*), several prominent peaks (*bimodal/multimodal*), or no apparent peaks (*uniform*)?



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

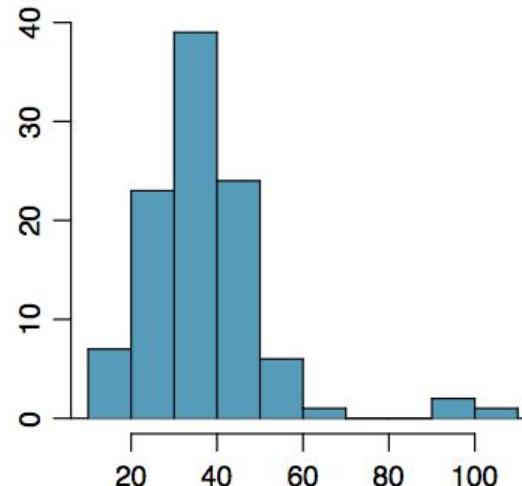
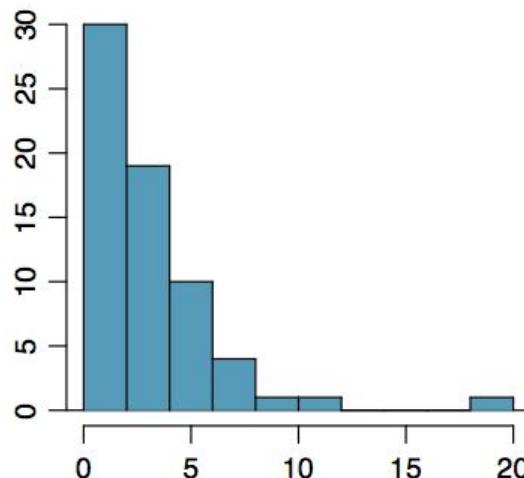
Is the histogram *right skewed*, *left skewed*, or *symmetric*?



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

unimodal



bimodal



multimodal



uniform



Skewness

right skew



left skew



symmetric

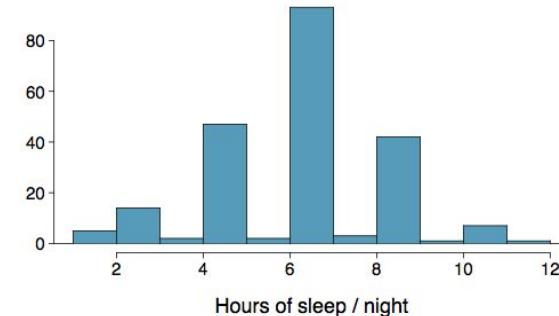


Variance

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:



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

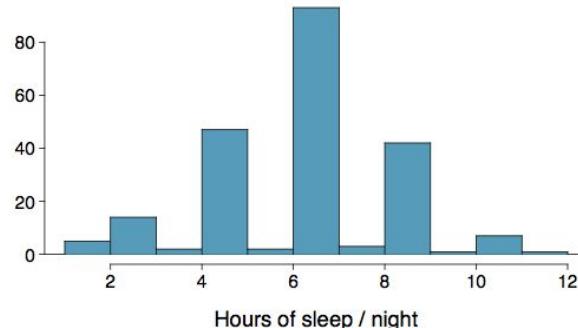
Standard Deviation

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:

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



- 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.

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

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

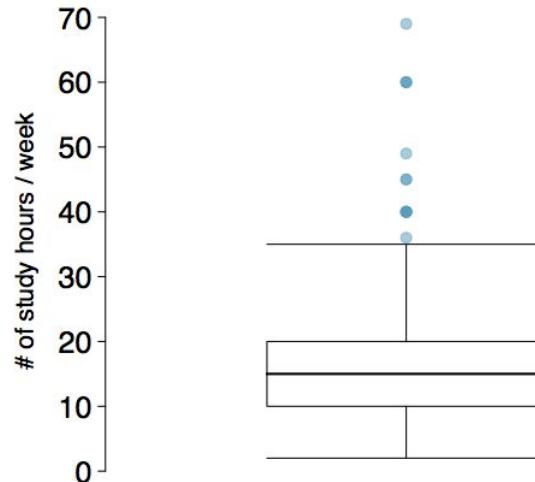
Q1, Q3, and IQR

- 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 .

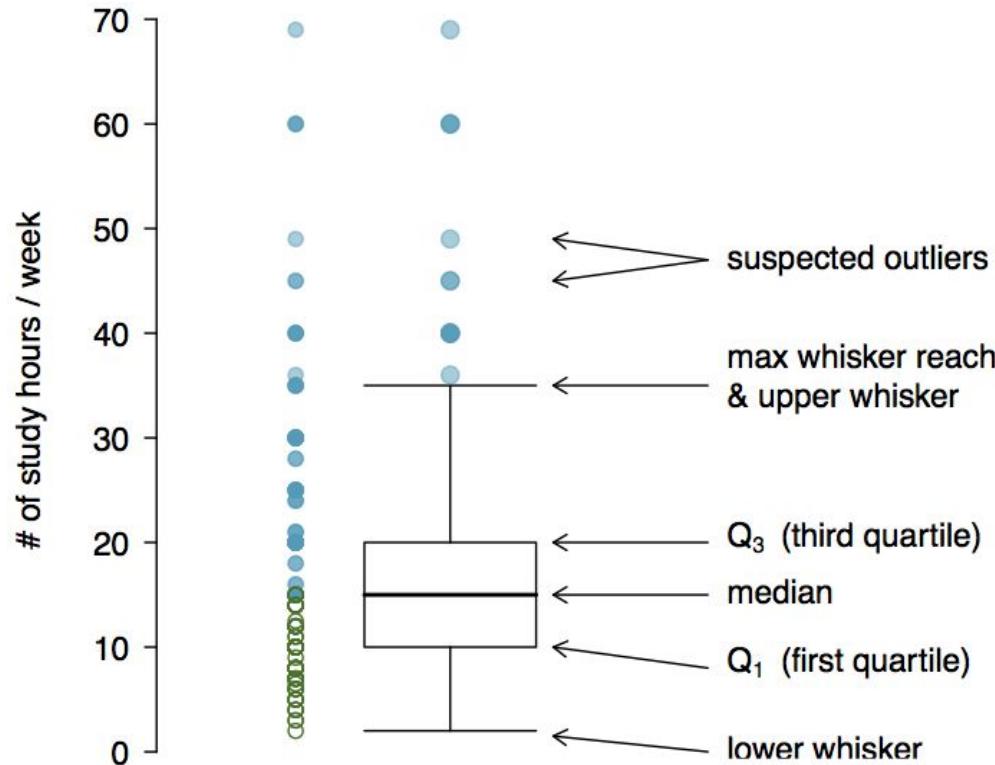
$$IQR = Q3 - Q1$$

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 $1.5 \times \text{IQR}$ away from the quartiles.

$$\text{max upper whisker reach} = Q3 + 1.5 \times \text{IQR}$$

$$\text{max lower whisker reach} = Q1 - 1.5 \times \text{IQR}$$

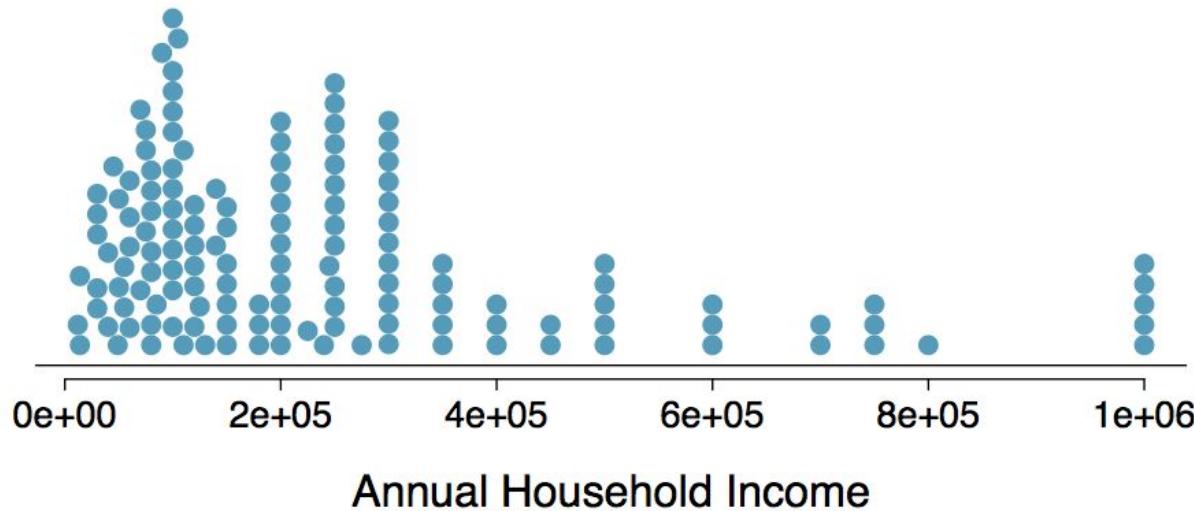
$$\text{IQR: } 20 - 10 = 10$$

$$\text{max upper whisker reach} = 20 + 1.5 \times 10 = 35$$

$$\text{max lower whisker reach} = 10 - 1.5 \times 10 = -5$$

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.

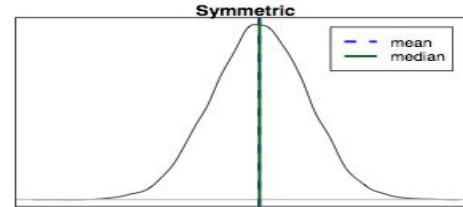
Robust Statistics



scenario	robust		not robust	
	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

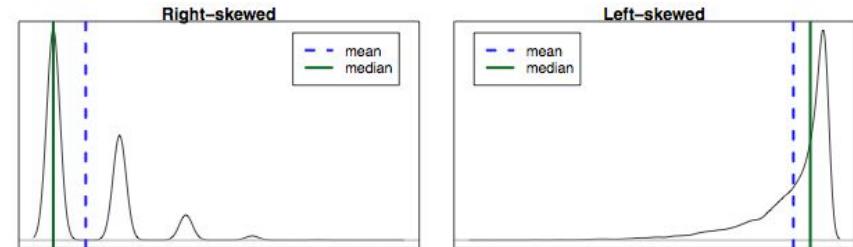
Mean vs. Median

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



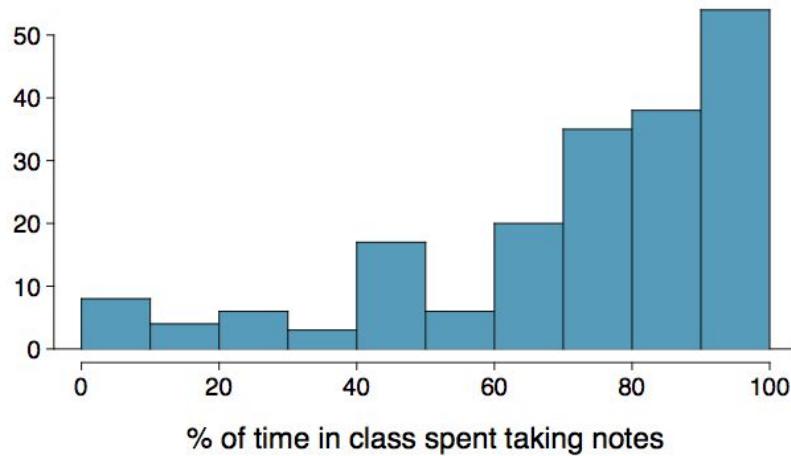
If the distribution is skewed or has extreme outliers, center is often defined as the median

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



Practice

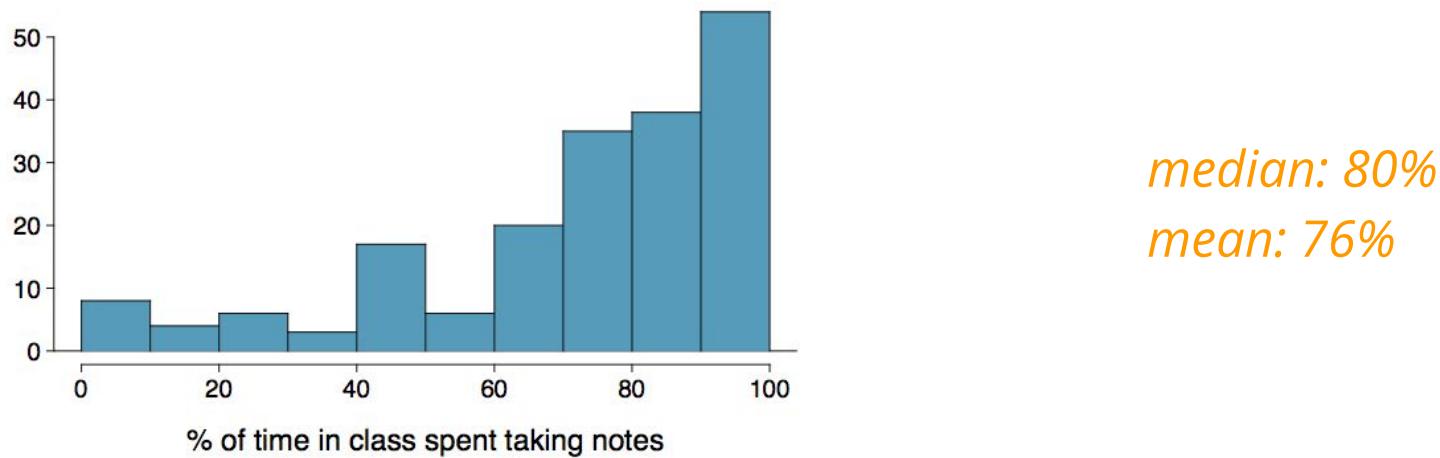
Which is most likely true for the distribution of percentage of time actually spent taking notes in class versus on Facebook, Twitter, etc.?



- (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.?

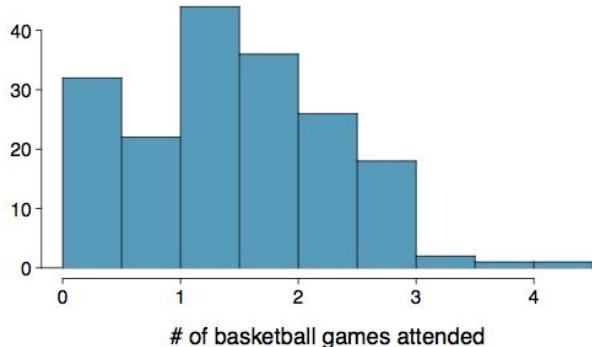
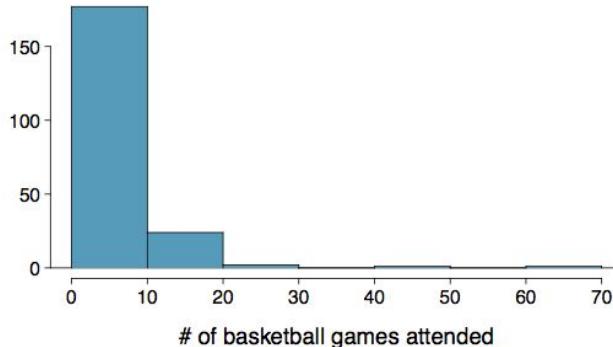


- (a) mean > median
- (b) mean ~ median
- (c) mean < median*
- (d) impossible to tell

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

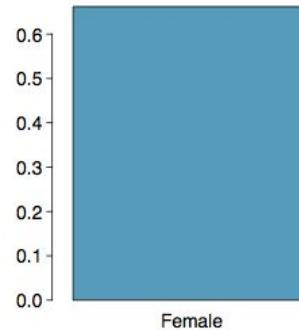
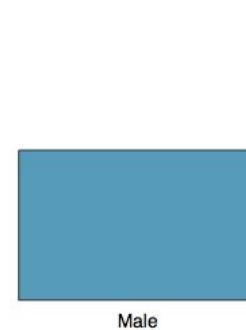
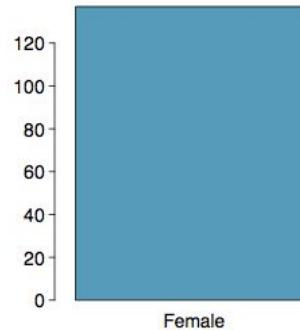
A table that summarizes data for two categorical variables is called a *contingency table*.

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

gender	looking for spouse		
	No	Yes	Total
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?

gender	looking for spouse		
	No	Yes	Total
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: $51 / 137 \sim 0.37$
- % Males looking for a spouse: $18 / 70 \sim 0.26$

Lab 2: Visualisation on some datasets using R

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.

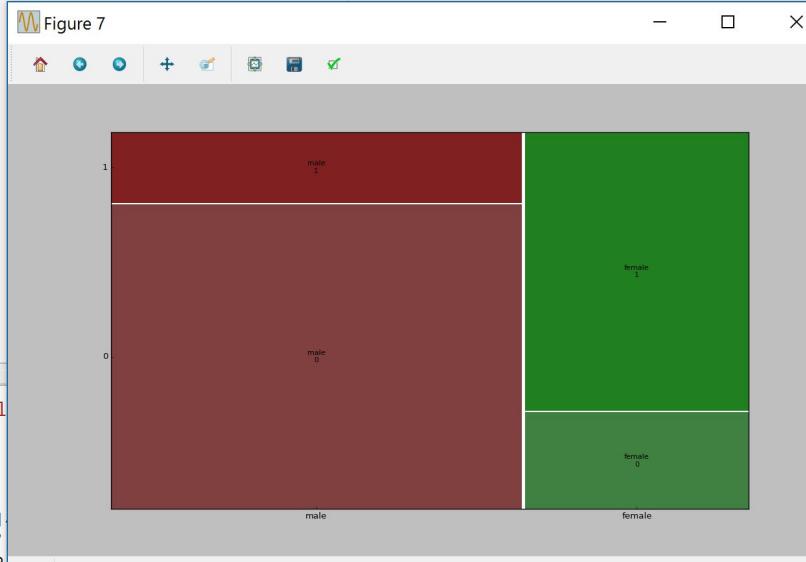
The type of plot depends from the types of exploratory and response variable.

		Input feature	
		Categorical	Numerical
Response variable	Categorical	Mosaic plots Section 2.3.1	Box plots Section 2.3.2
	Numerical	Density plots Section 2.3.3	Scatterplots Section 2.3.4

A mosaic plot on Titanic data in Python

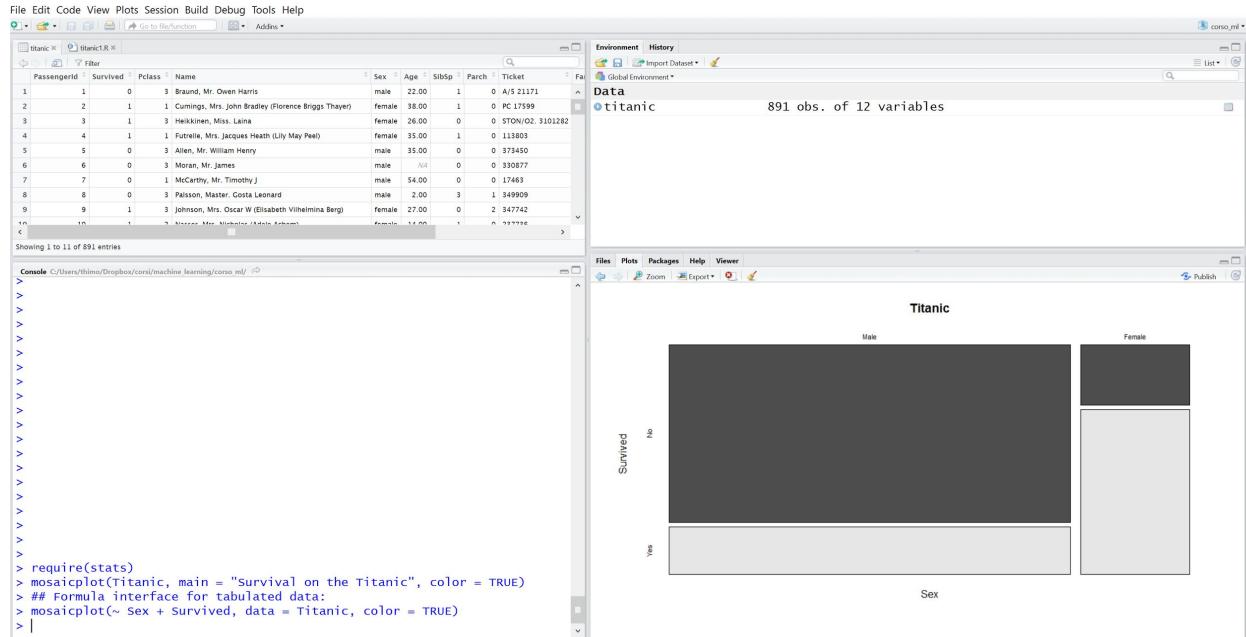
titanic_example.py svm_gui.py

```
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
```

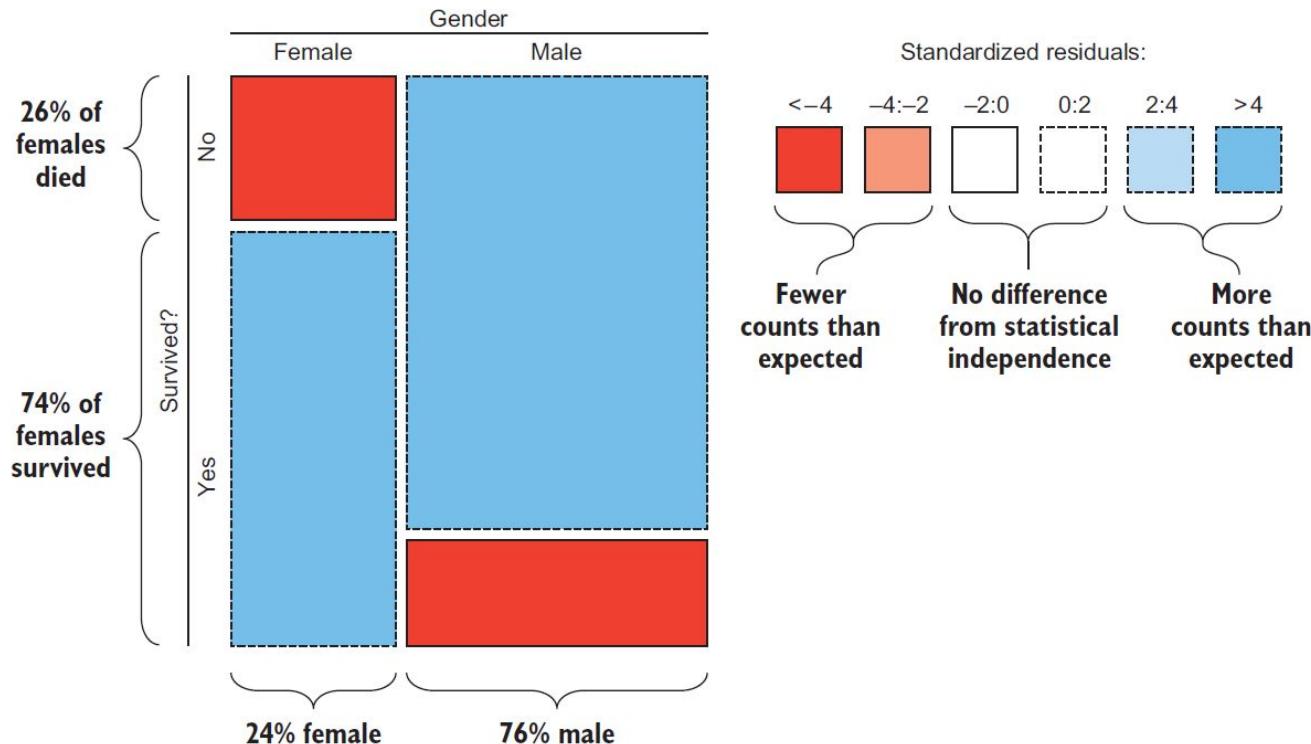
In [21]: %run "C:\Users\thimo\Dropbox\corsi\machine_l...
In [22]: mosaic(df, ['Sex', 'Survived'])
Out[22]:<matplotlib.figure.Figure at 0x18592e80>,
OrderedDict([(('male', '0'), (0.0, 0.0, 0.6443651551
0.6443651551445914, 0.18828054376802911)), (('female'
('female', '1'), (0.6493402795227009, 0.260427026683
x= y=

Visualisation in R

- <https://github.com/thimotyb/rwml-R>
- `setwd(curdir)`
- Chapter 2



Reading a Mosaic Plot



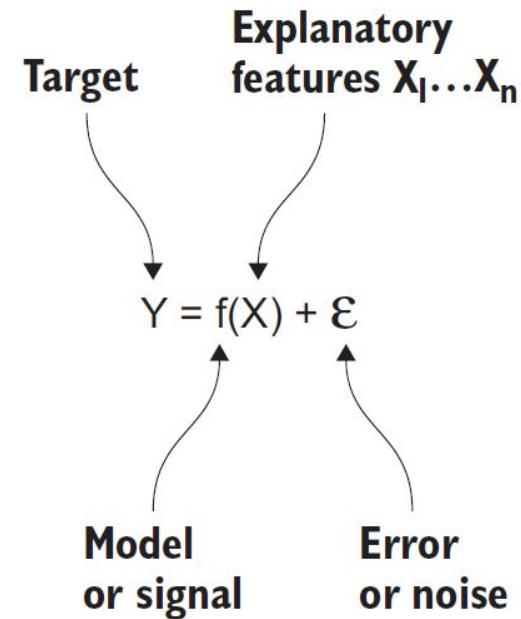
Modeling and Prediction

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...)

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, X_{new} , by plugging those new features into the model. f_{est} denotes the machine-learning estimate of f (the true relationship between the features and the target): $Y_{\text{pred}} = f_{\text{est}}(X_{\text{new}})$

“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.

$$f(\mathbf{X}) = \beta_0 + \mathbf{X}_1 \times \beta_1 + \mathbf{X}_2 \times \beta_2 + \dots$$

In this equation, the unknown parameters, $0, 1, \dots$ 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

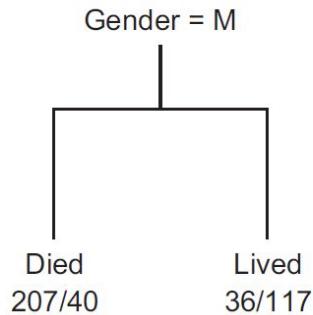
In nonparametric models, f doesn't take a simple, fixed function. Instead, the form and complexity of f adapts to the complexity of the data. A simple example of a nonparametric model is a classification tree. A classification tree is a series of recursive binary decisions on the input features.

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.

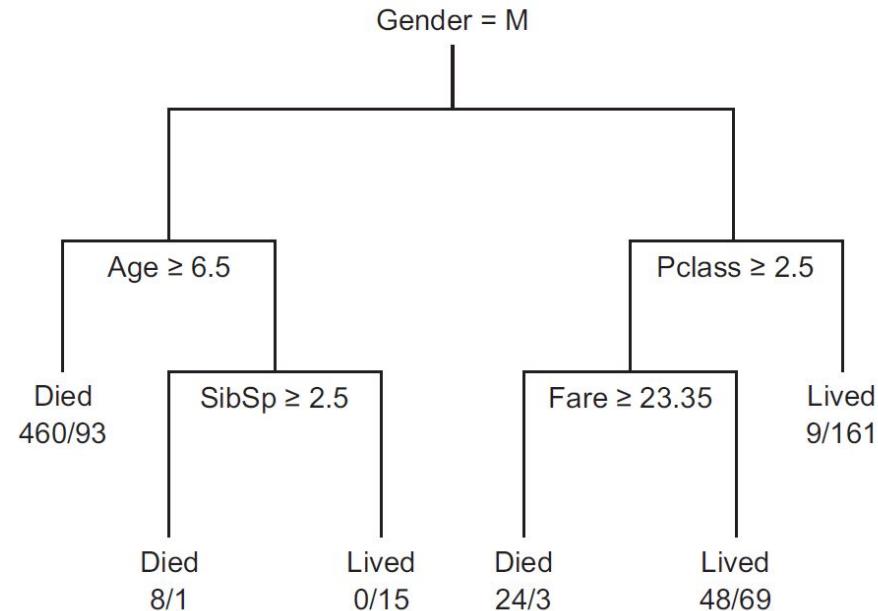
Classification trees are nonparametric because the depth and complexity of the tree isn't fixed in advance, but rather is learned from the data. If the relationship between the target variable and the input features is complex and there's a sufficient amount of data, then the tree will grow deeper, uncovering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

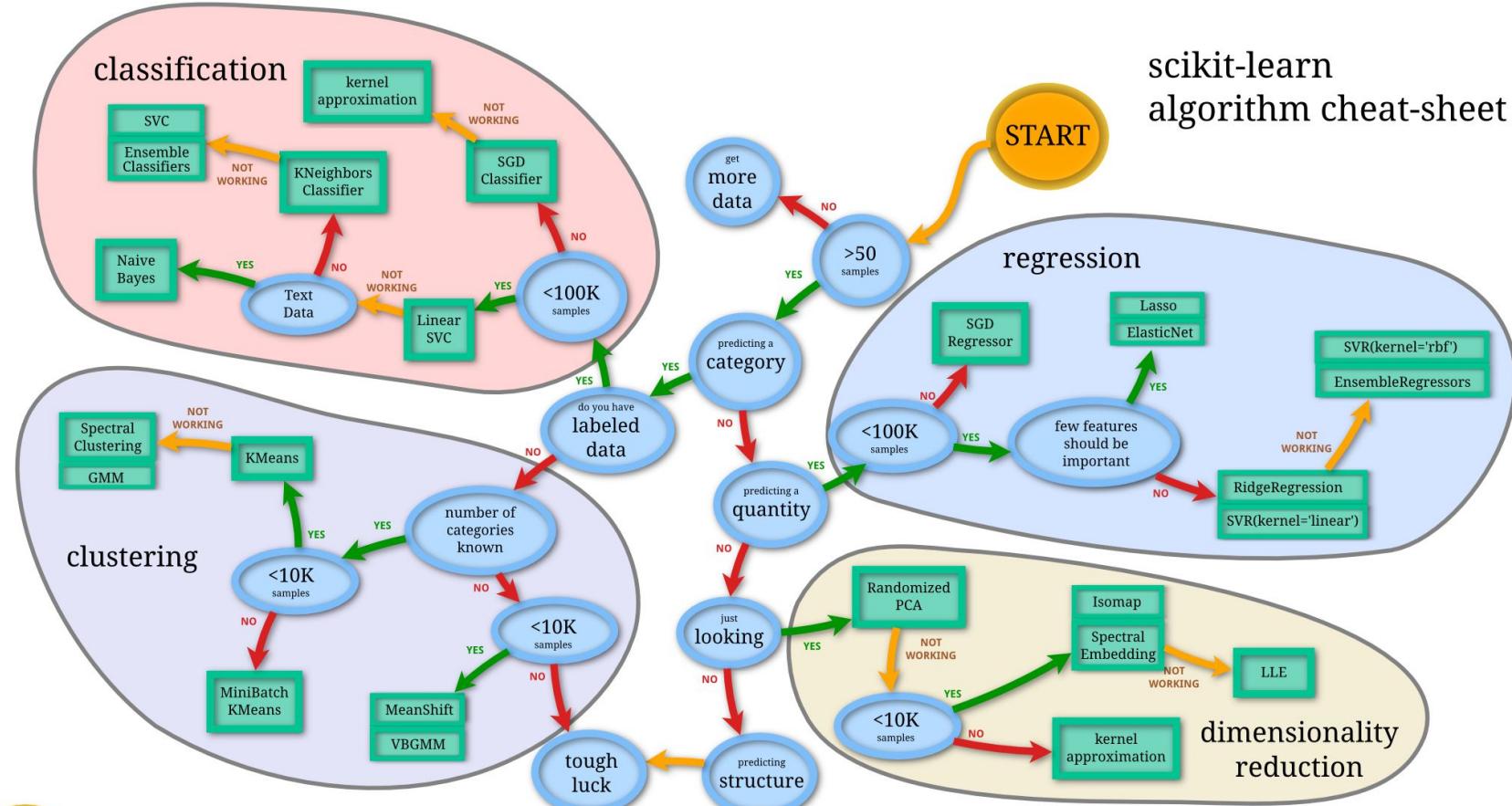
Name	Type	Use	Linear/ nonlinear	Requires normalization
Linear regression	Regression	<p>Model a scalar target with one or more quantitative features. Although regression computes a linear combination, features can be transformed by nonlinear functions if relationships are known or can be guessed.</p>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

SVM	Classification/regression	<p>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.</p> <p>R: https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf Python: http://scikit-learn.org/stable/modules/svm.html</p>	Linear	Yes
SVM with kernel	Classification/regression	<p>SVM with support for a variety of nonlinear models.</p> <p>R: https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf Python: http://scikit-learn.org/stable/modules/svm.html</p>	Nonlinear	Yes

K-nearest neighbors	Classification/regression	<p>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.</p> <p>R: https://cran.r-project.org/web/packages/class/class.pdf Python: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html</p>	Nonlinear	Yes
Decision trees	Classification/regression	<p>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.</p> <p>R: www.statmethods.net/advstats/cart.html Python: http://scikit-learn.org/stable/modules/tree.html#tree</p>	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



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 $\hat{f}(x_0)$,
- the squared bias of $\hat{f}(x_0)$ and
- the variance of the error

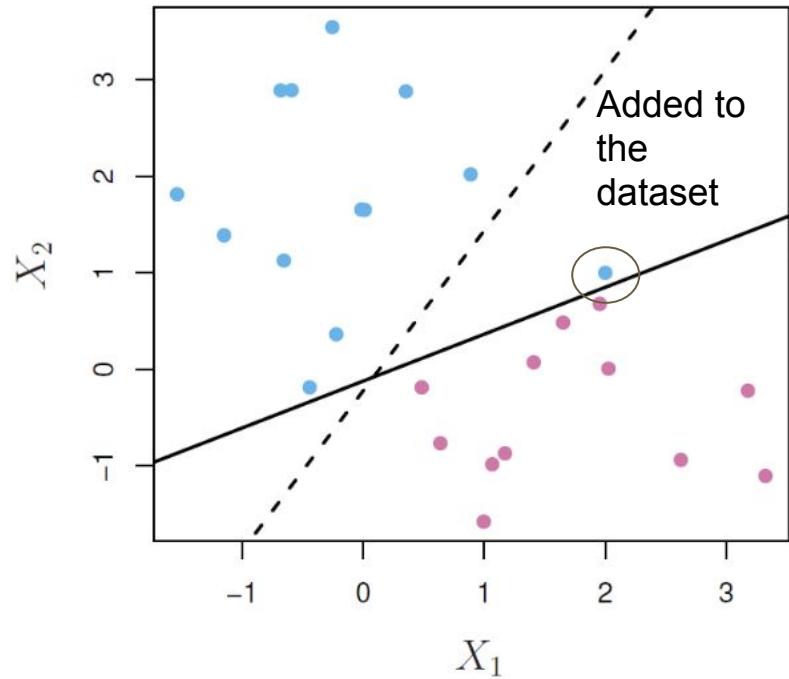
To minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias.

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

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.

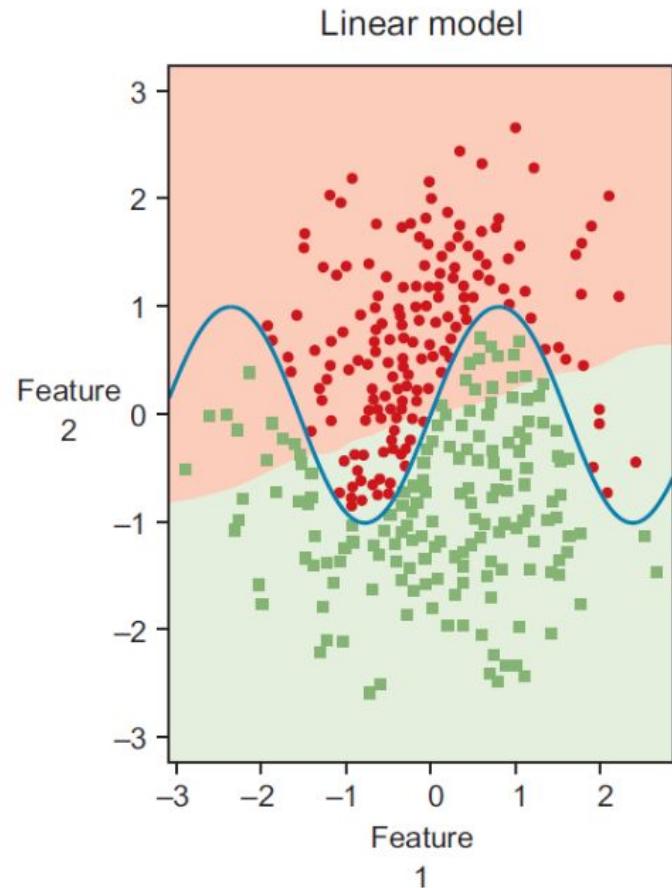


Bias

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 X_1, X_2, \dots, X_p .

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

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
3	26	0	0	2.815138	1	0
1	35	1	0	7.286975	1	0
3	35	0	0	2.837252	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)
```

Returns
predictions
(0 or 1)

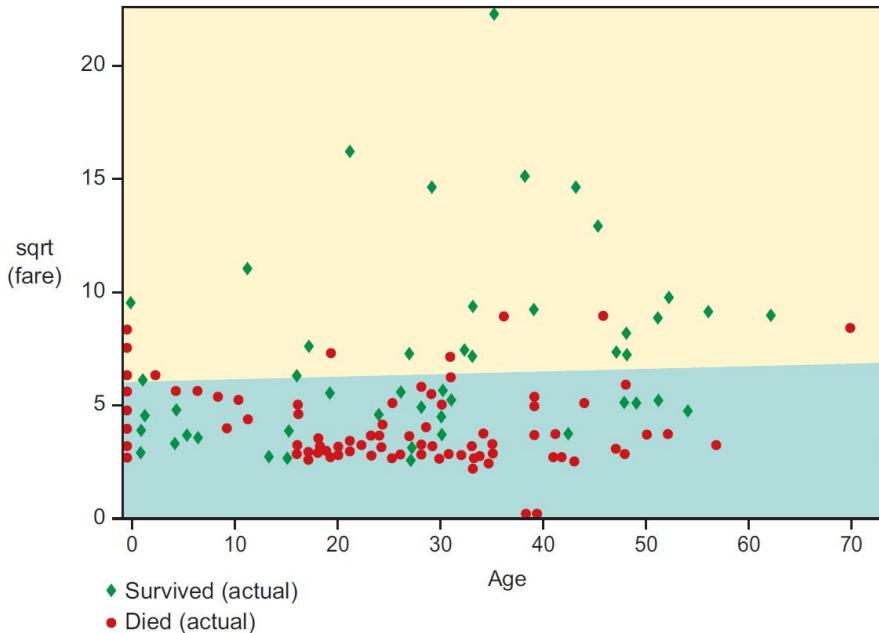
Fits the logistic regression algorithm using features and target data

Makes predictions on a new set of features using the model

Returns the model built by the algorithm

Imports the logistic regression algorithm

Prediction Results



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

```
Out[15]: array([0, 0, 0, 0, 1, 1, 0, 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, 1, 0, 0, 0,  
0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,  
1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,  
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,  
0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,  
0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,  
0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0])
```

```
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"])
```

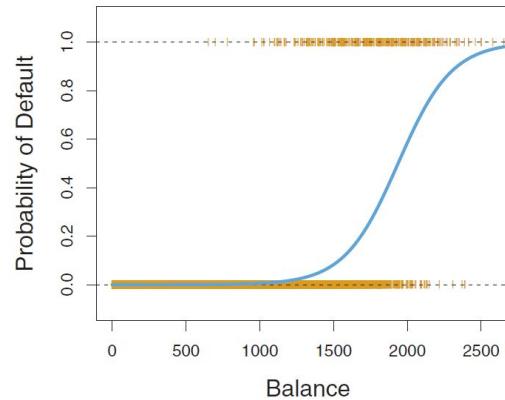
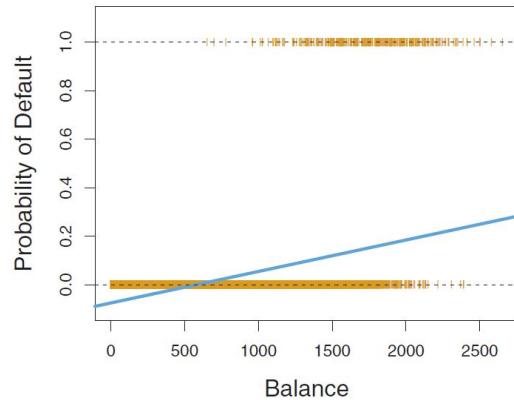
```
Out[16]: 0.82681564245810057
```

Classifier
predicts: Survived

Decision
boundary

Classifier
predicts: Died

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 β_1 , or equivalently it multiplies the odds by e^{β_1} . Because the relationship between $p(X)$ and X is not linear, β_1 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 β_0 and β_1 in are unknown, and must be estimated based on the available training data, using *maximum likelihood* to fit the model.

We seek estimates for β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ of survival for each individual, corresponds as closely as possible to the individual's observed survival status. In other words, we try to find $\hat{\beta}_0$ and $\hat{\beta}_1$ 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.

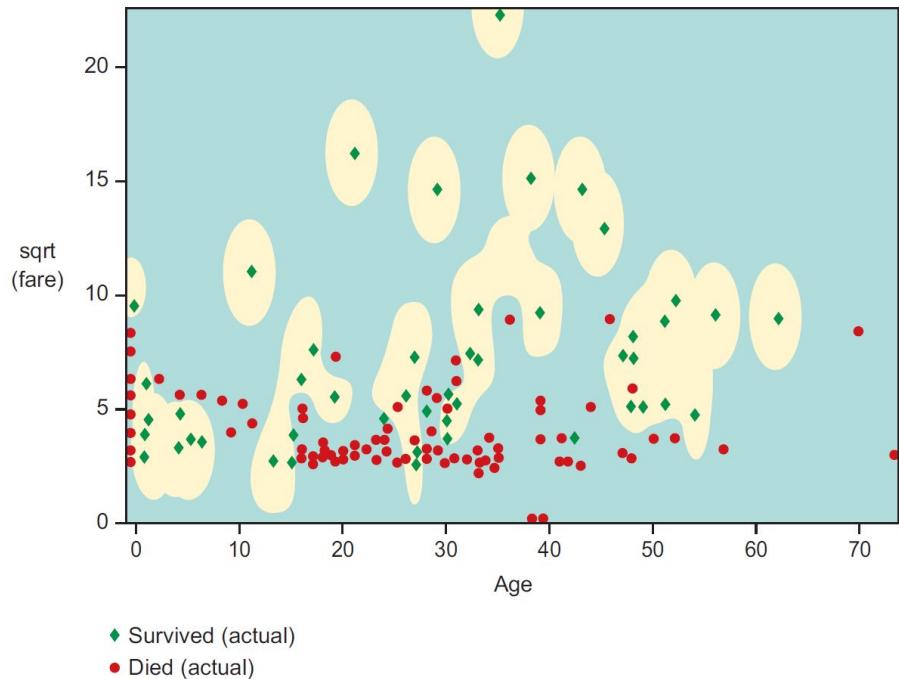
A typical non-linear classifier is SVM (Support Vector Machines)

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

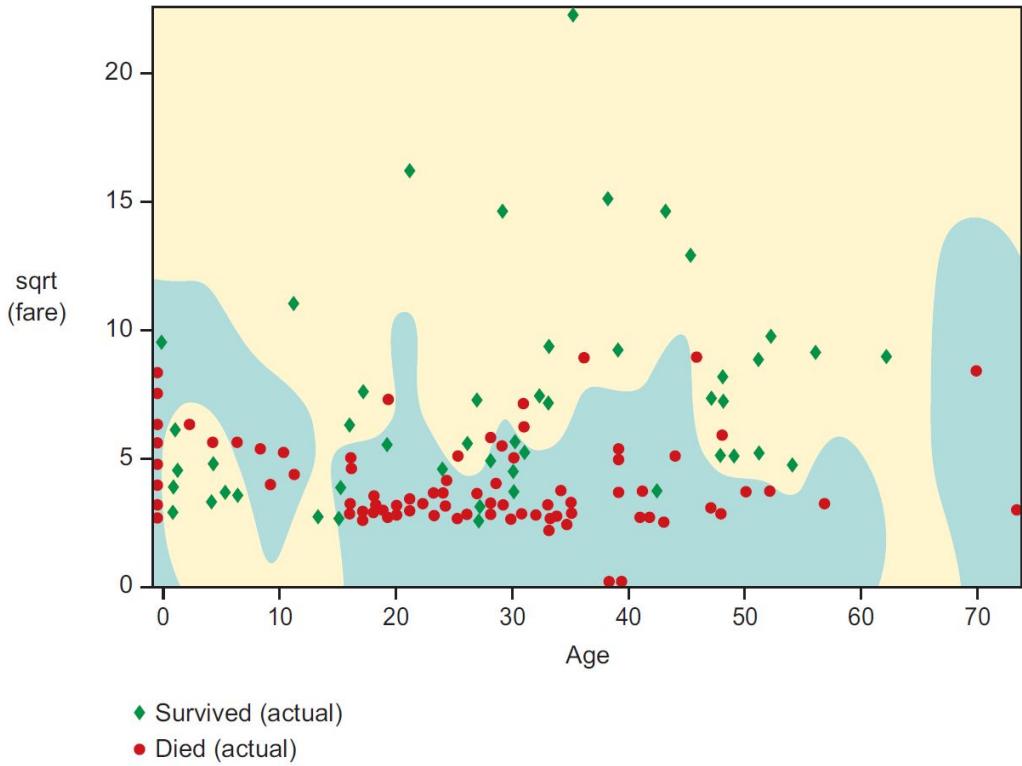
```
In [62]: from sklearn.svm import SVC  
model = SVC()  
model.fit(features, data_train["Survived"])  
  
Out[62]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,  
            kernels='rbf', max_iter=-1, probability=False, random_state=None,  
            shrinking=True, 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:

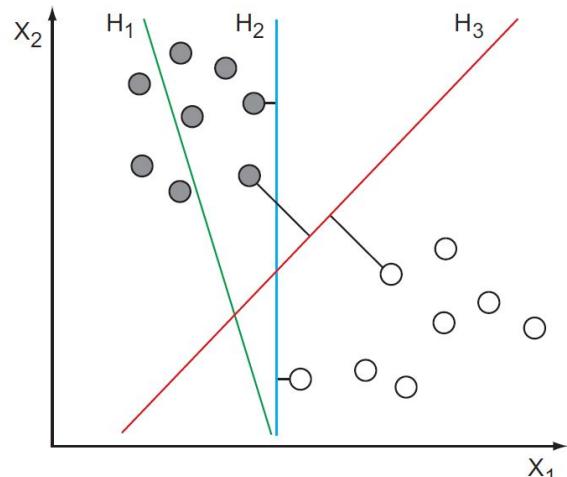
- We soften what we mean by “separates”, and
- We enrich and enlarge the feature space so that separation is possible.

Hyperplanes

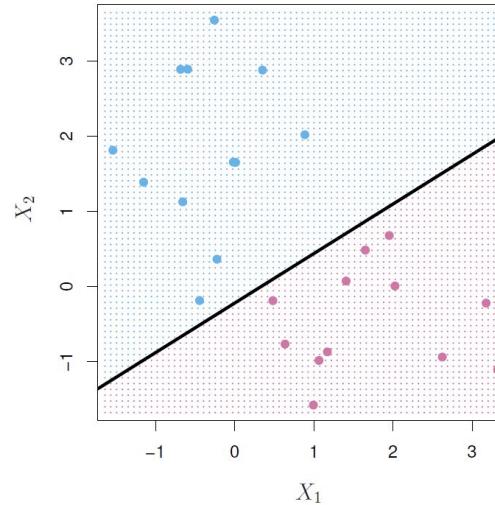
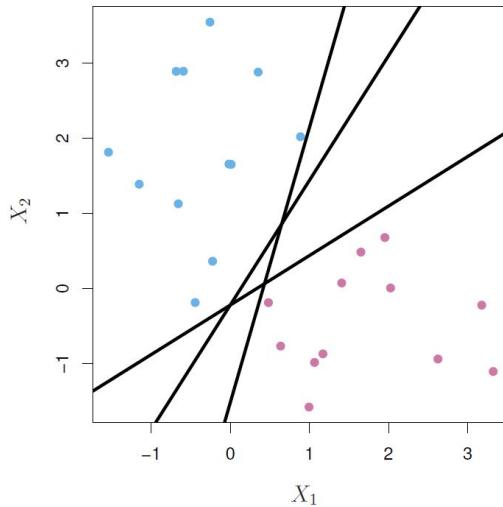
- A hyperplane in p dimensions is a flat affine subspace of dimension $p - 1$.
- In general the equation for a hyperplane has the form

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

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



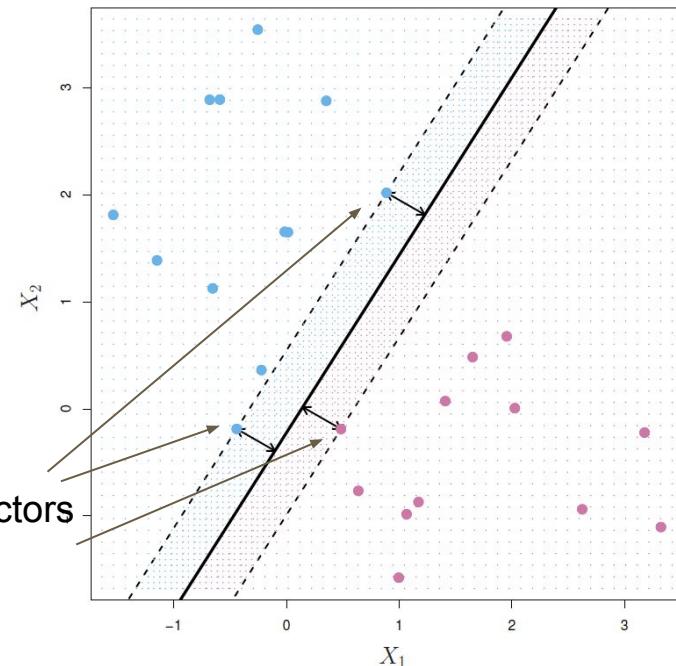
Separating Hyperplanes



- 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.



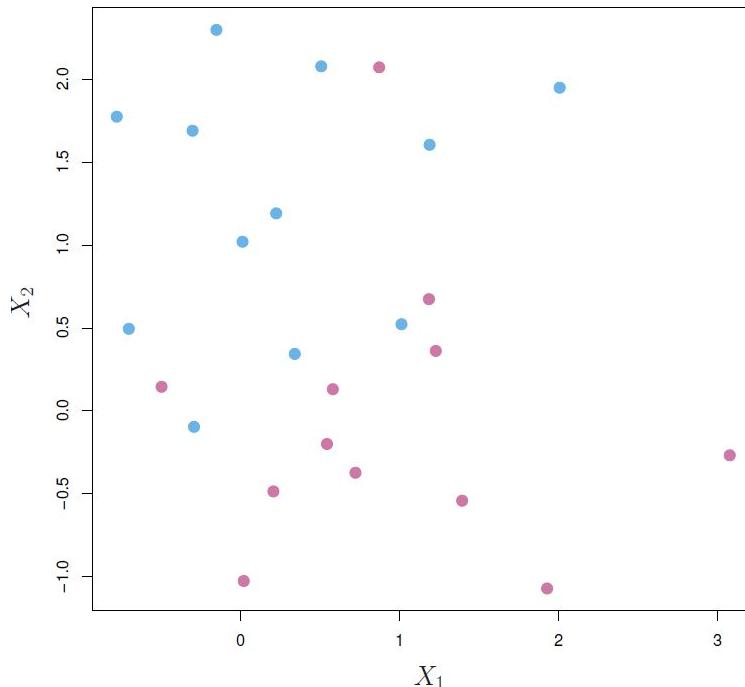
Constrained optimization problem

$$\begin{aligned} & \text{maximize } M \\ & \beta_0, \beta_1, \dots, \beta_p \end{aligned}$$

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

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M \quad \text{for all } i = 1, \dots, N.$$

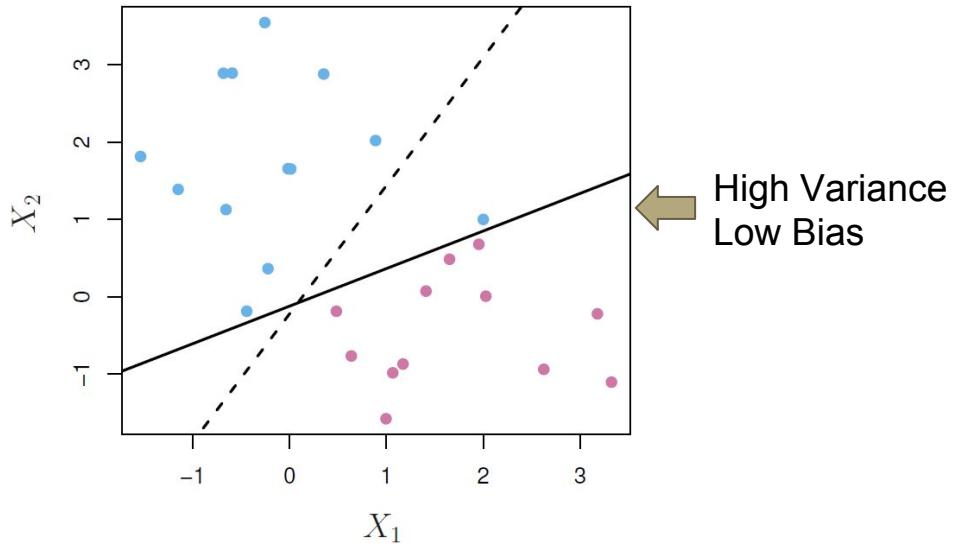
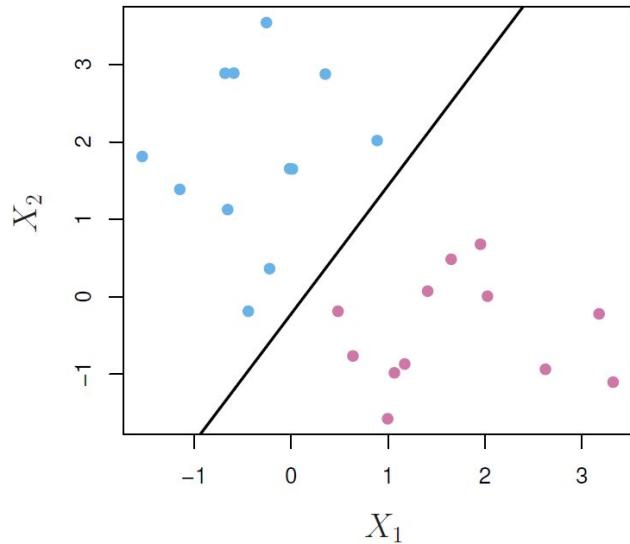
Non-separable Data



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

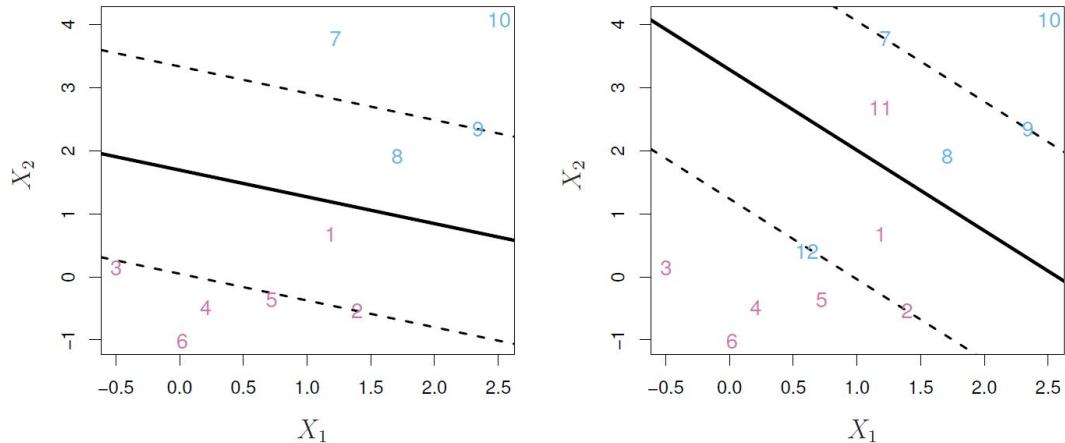
The support vector classifier, sometimes called a soft margin classifier, does exactly this.

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 observations are on the wrong side of the hyperplane and the wrong side of the margin.



$$\begin{aligned}
 & \underset{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n}{\text{maximize}} \quad M \quad \text{subject to} \quad \sum_{j=1}^p \beta_j^2 = 1, \\
 & y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \\
 & \epsilon_i \geq 0, \quad \sum_{i=1}^n \epsilon_i \leq C,
 \end{aligned}$$

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 ϵ s 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 i th observation is on the correct side of the margin

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

If $\epsilon_i > 1$ then the i th observation is on the wrong side of the hyperplane

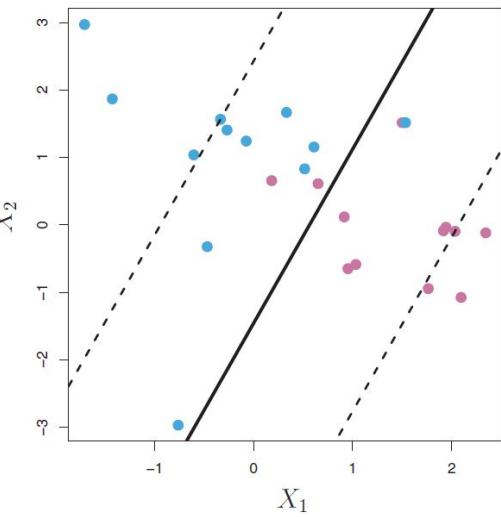
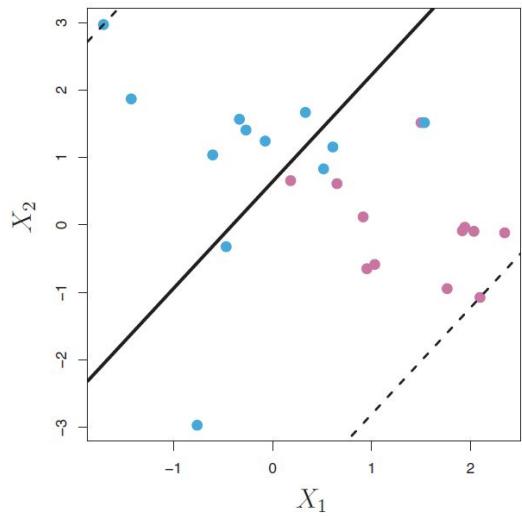
“C” parameter as slack budget

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

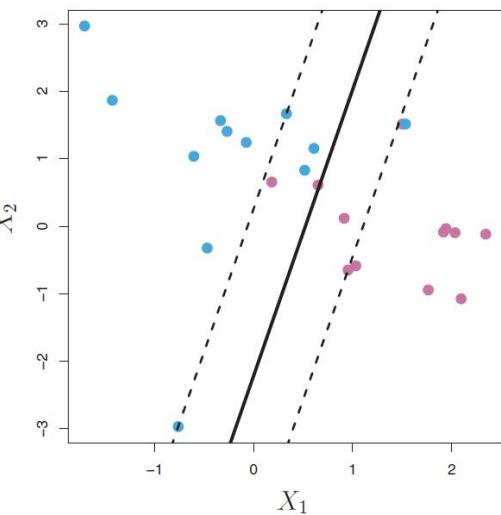
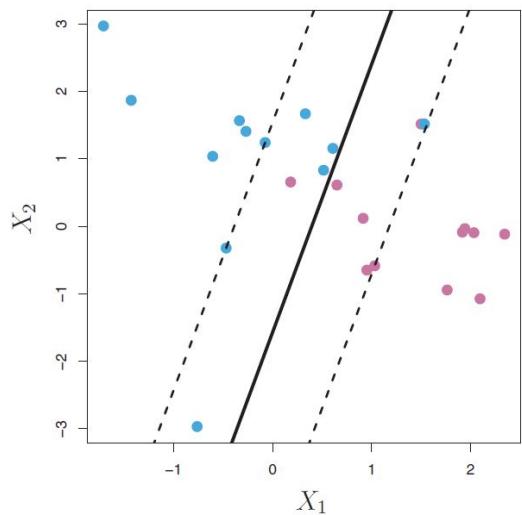
For $C > 0$ no more than C observations can be on the wrong side of the hyperplane (each slack = 1)

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.

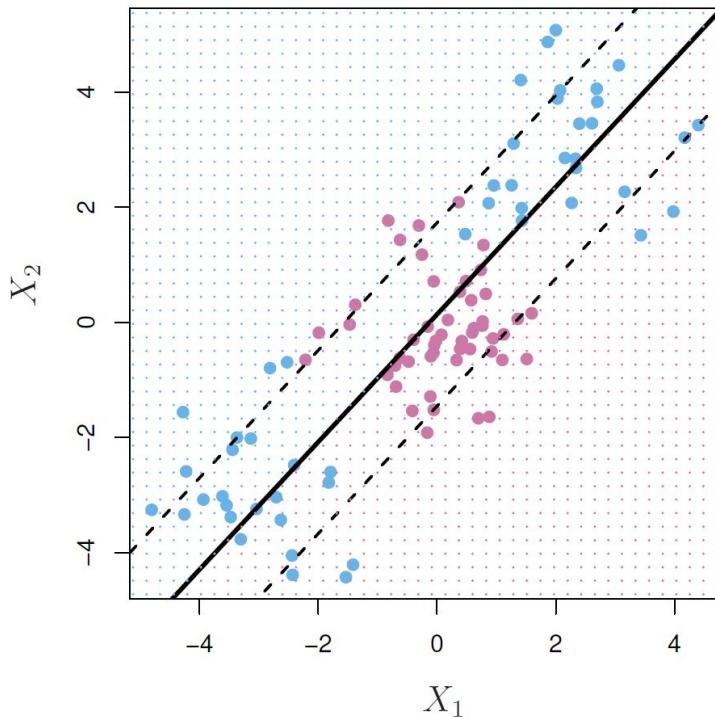


Top left: high C

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. $X_1^2, X_1^3, X_1X_2, X_1X_2^2, \dots$. 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

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

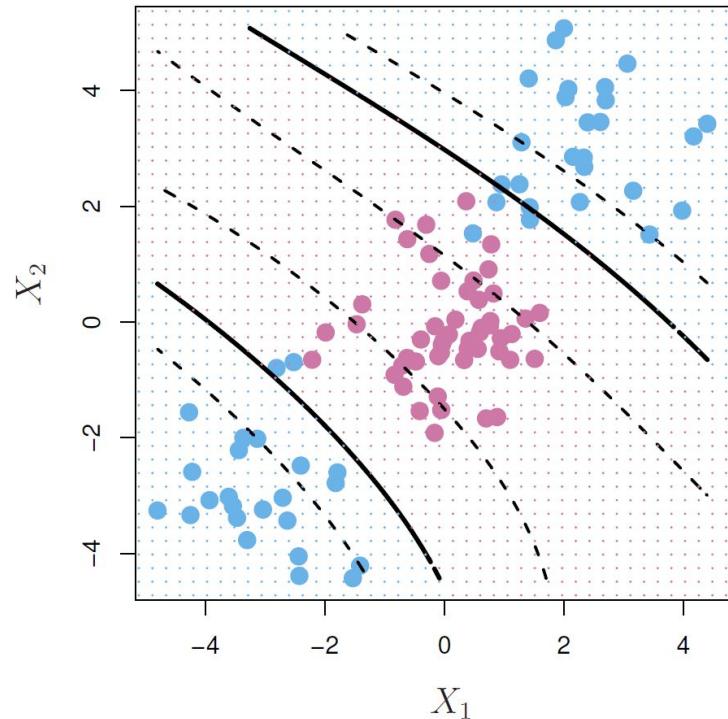
This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

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



$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 + \beta_6 X_1^3 + \beta_7 X_2^3 + \beta_8 X_1 X_2^2 + \beta_9 X_1^2 X_2 = 0$$

Inner

Product

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

- The linear support vector classifier can be represented as

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

- To estimate the parameters $\alpha_1, \dots, \alpha_n$ 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.

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

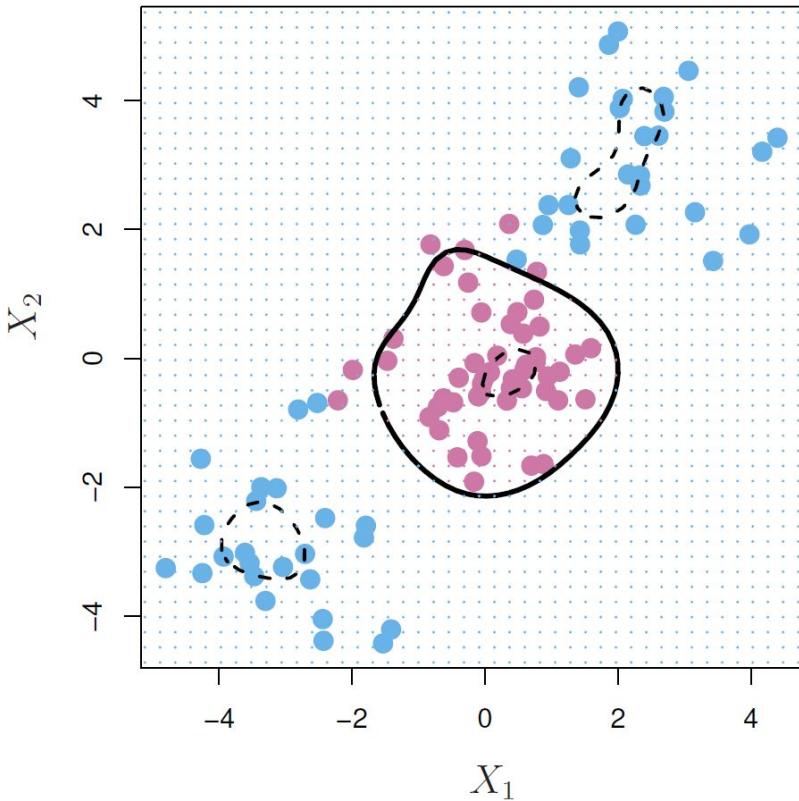
computes the inner-products needed for d dimensional polynomials — $\binom{p+d}{d}$ basis functions!

- The solution has the form

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

Radial Kernel (RBF)

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



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

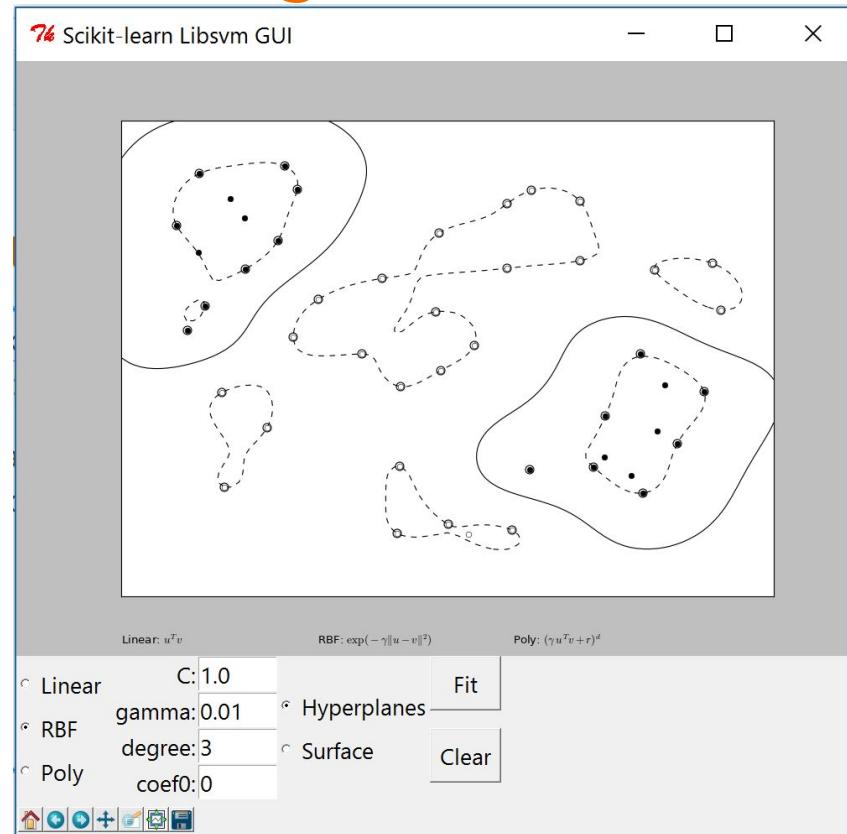
Implicit feature space;
very high dimensional.

Controls variance by
squashing down most
dimensions severely

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.



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.



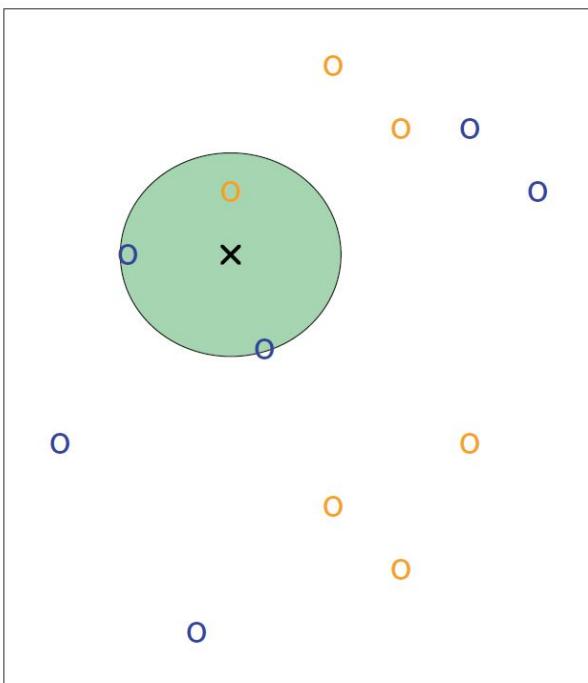
KNN Classification (K-Nearest Neighbors)

Given a positive integer K and a test observation x_0 , the KNN classifier first identifies the neighbors K points in the training data that are closest to x_0 , represented by N_0 .

It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:

$$\Pr(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j).$$

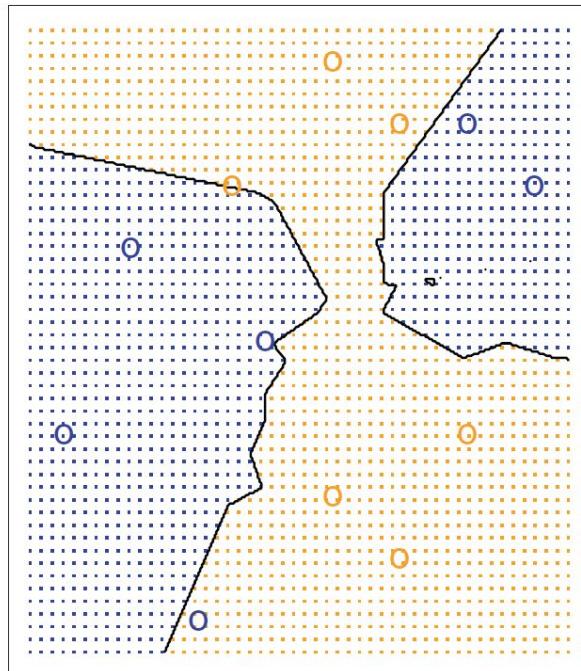
KNN Classification - Working principle



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

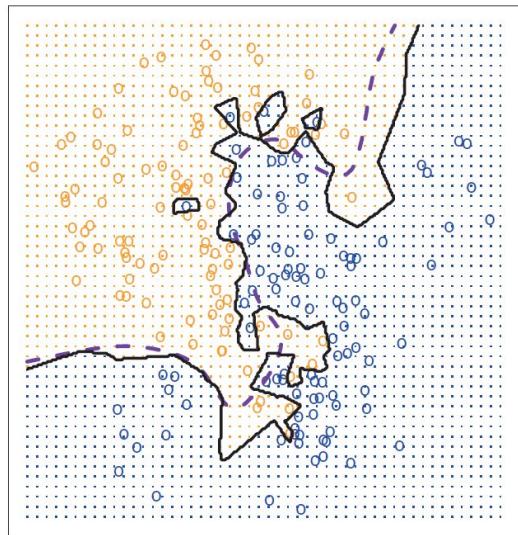


We have applied the KNN approach with $K = 3$ at all of the possible values for X_1 and X_2 , and have drawn in the corresponding KNN decision boundary.

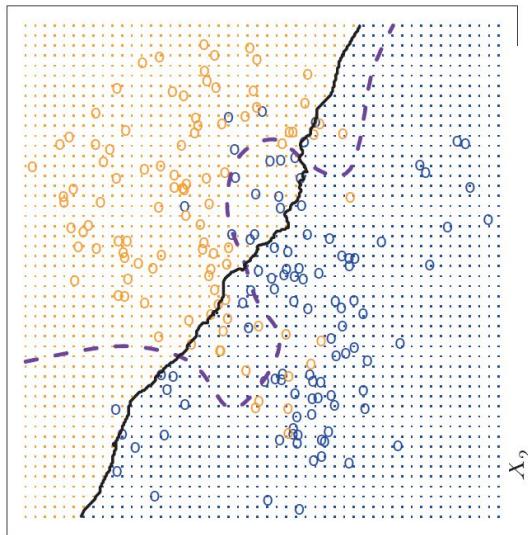
With a larger dataset, the value of K is important to find the correct balance between overfitting and predictive inflexibility.

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

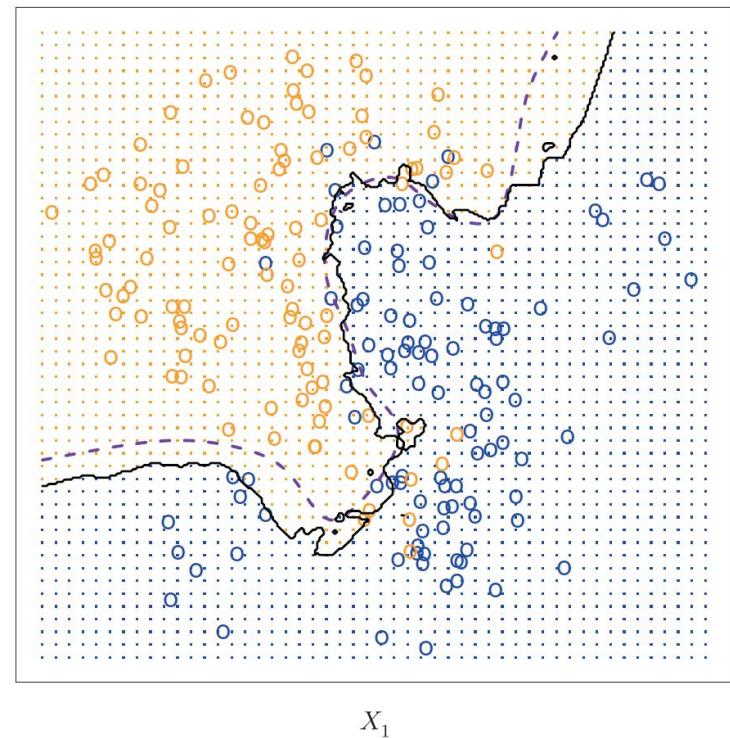
KNN: K=1



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	0.0	1.0	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	0.0	0.0
Digit 3	0.3	0.0	0.0	0.0	0.0	0.6	0.0	0.1	0.0	0.0
Digit 4	0.0	0.0	0.1	0.0	0.0	0.0	0.5	0.0	0.4	0.0
Digit 5	0.0	0.0	0.0	0.0	0.7	0.0	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.

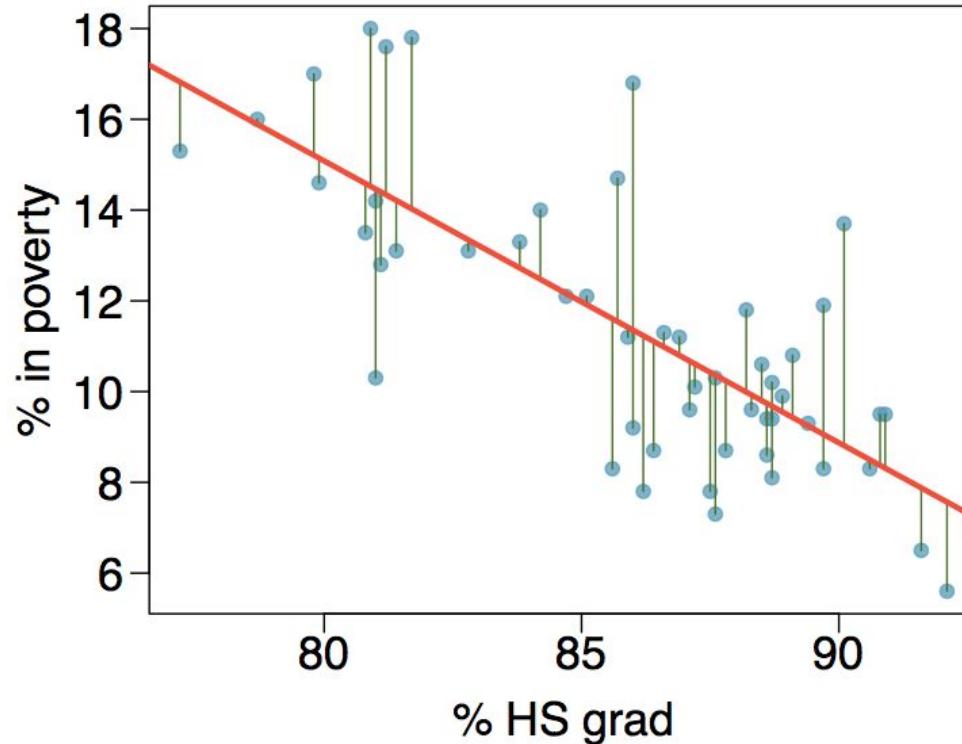
We call the act of predicting numerical values regression, and the model itself a regressor.

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.

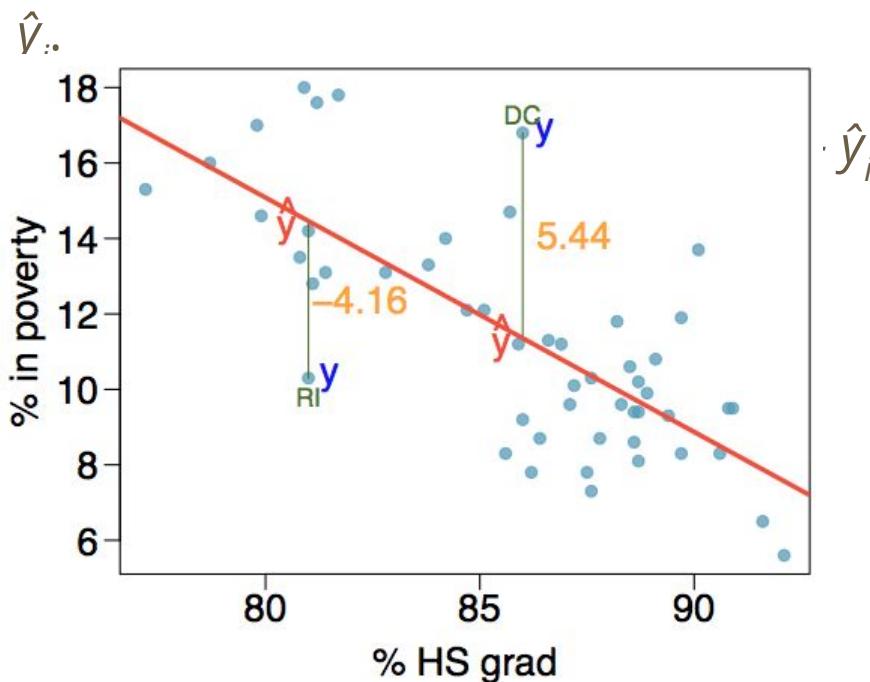
Residuals

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



Residuals (cont.)

Residual is the difference between the observed (y_i) and predicted



- % 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

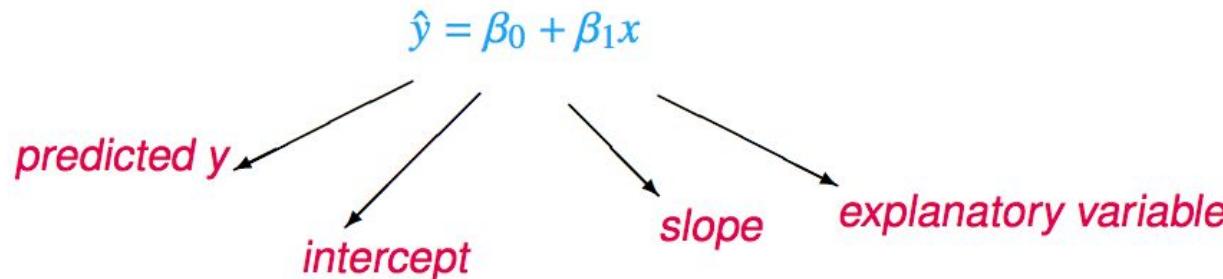
$$|e_1| + |e_2| + \dots + |e_n|$$

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

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

- 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

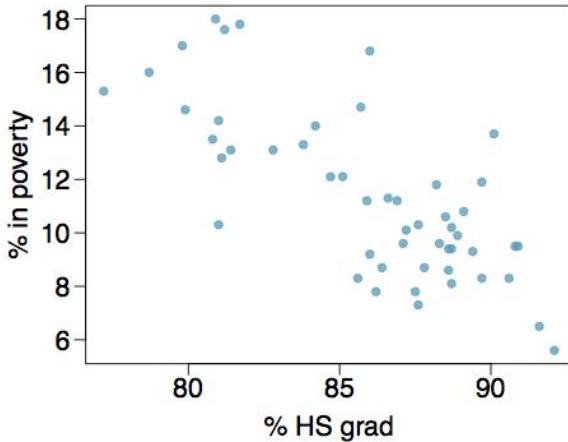
The least squares line



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...

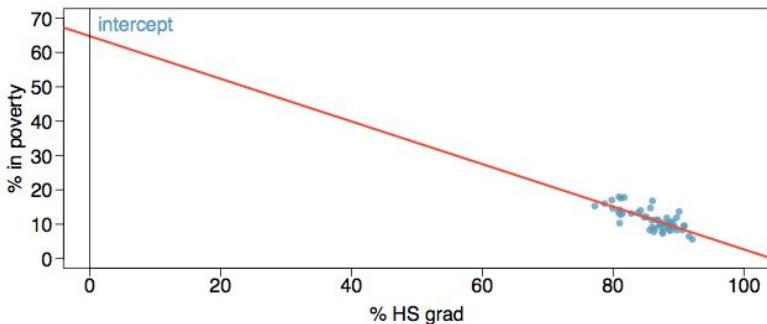
$$b_1 = \frac{3.1}{3.73} \times -0.75 = -0.62$$

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}) .

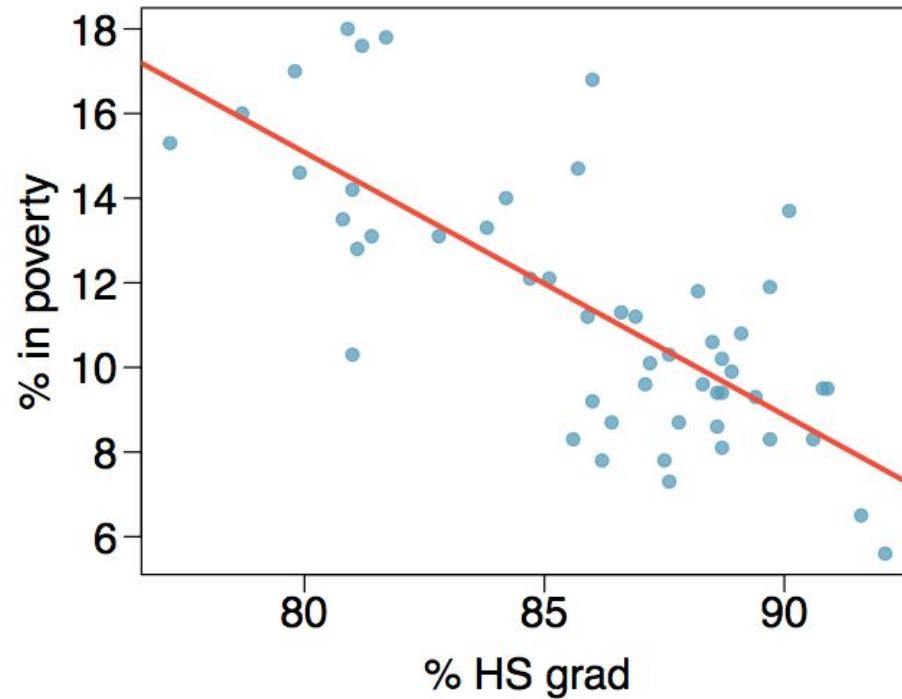


$$- b_1 \bar{x}$$

$$\begin{aligned}b_0 &= 11.35 - (-0.62) \times 86.01 \\&= 64.68\end{aligned}$$

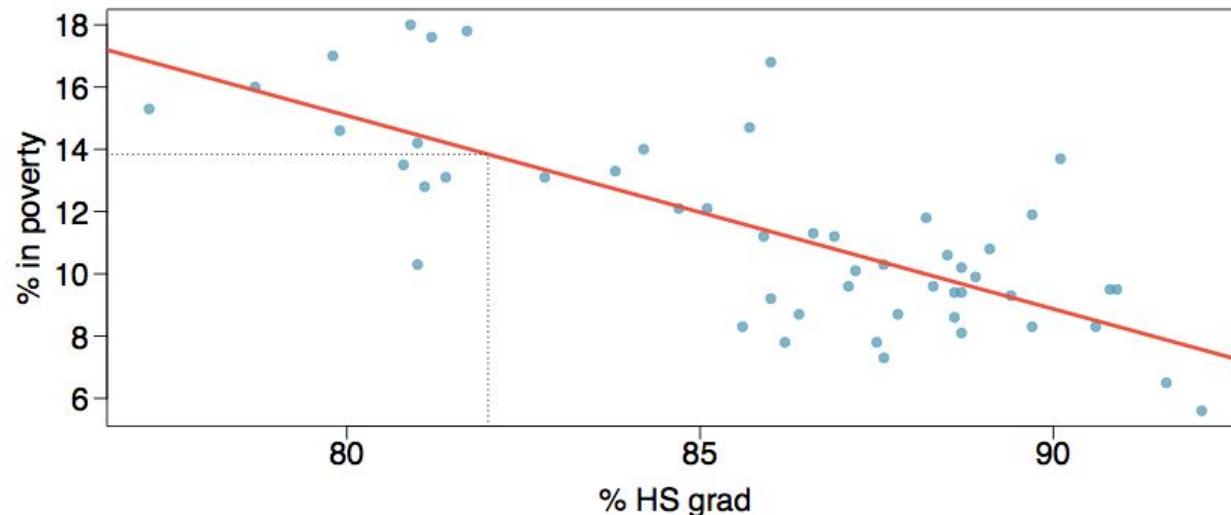
Regression line

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



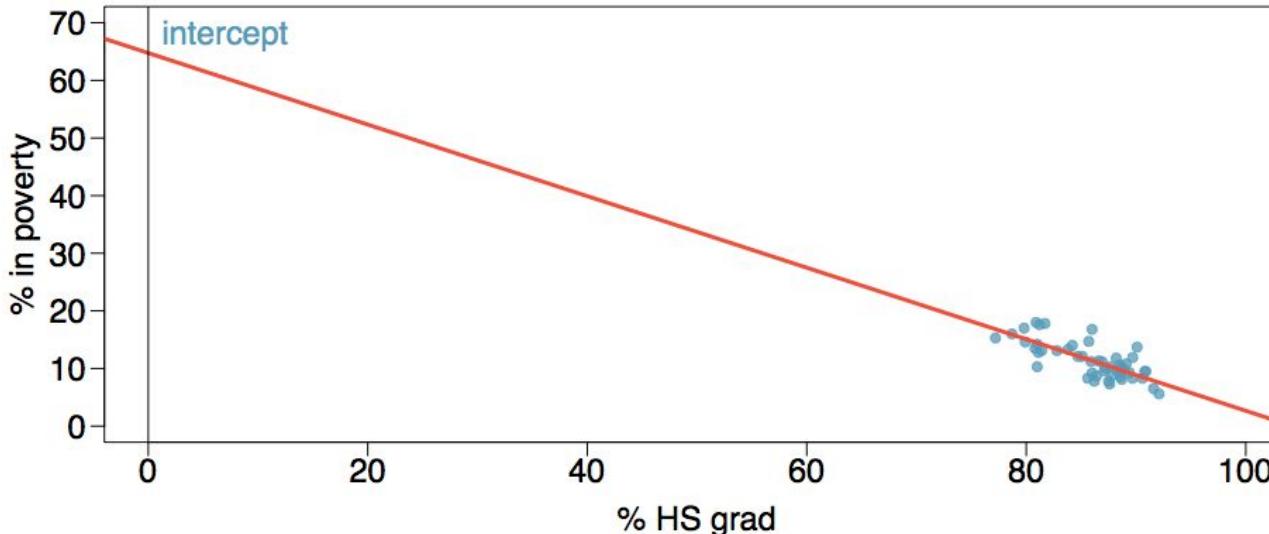
Prediction

- 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.

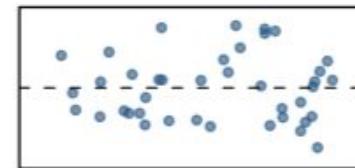
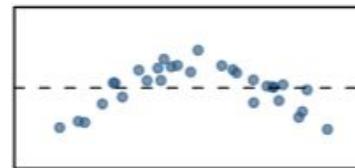
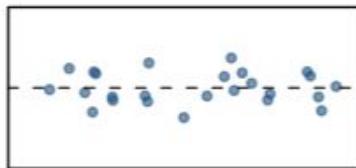
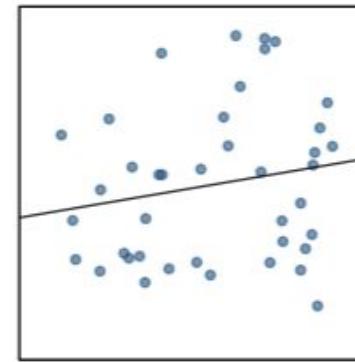
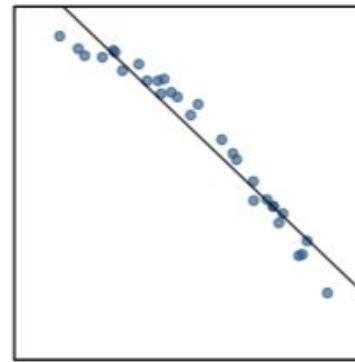
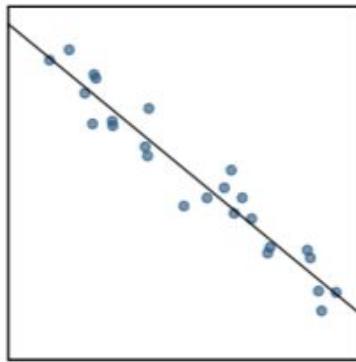


Conditions for the least squares line

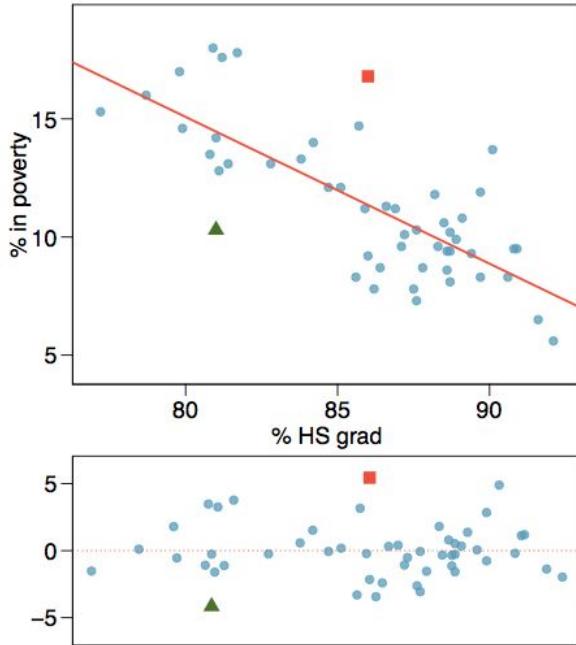
1. Linearity
2. Nearly normal residuals
3. Constant variability

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:

$$\% \text{ HS grad} = 81 \quad \% \text{ in poverty} = 10.3$$

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

$$\begin{aligned} e &= \% \text{ in poverty} - \widehat{\% \text{ in poverty}} \\ &= 10.3 - 14.46 = \textcolor{green}{-4.16} \end{aligned}$$

■ DC:

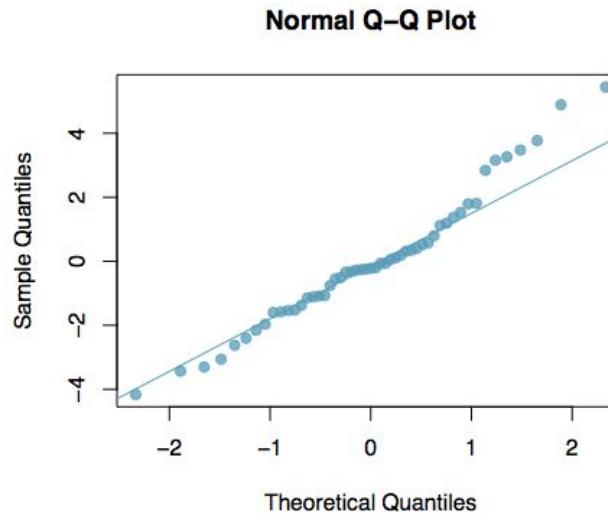
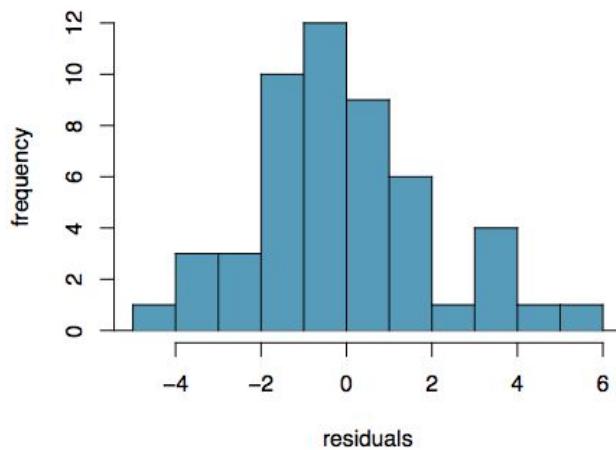
$$\% \text{ HS grad} = 86 \quad \% \text{ in poverty} = 16.8$$

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

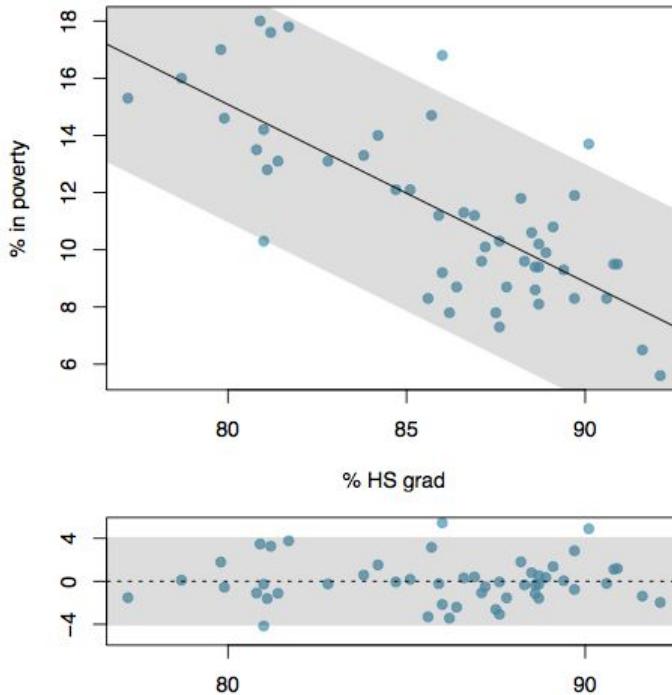
$$\begin{aligned} e &= \% \text{ in poverty} - \widehat{\% \text{ in poverty}} \\ &= 16.8 - 11.36 = \textcolor{red}{5.44} \end{aligned}$$

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.

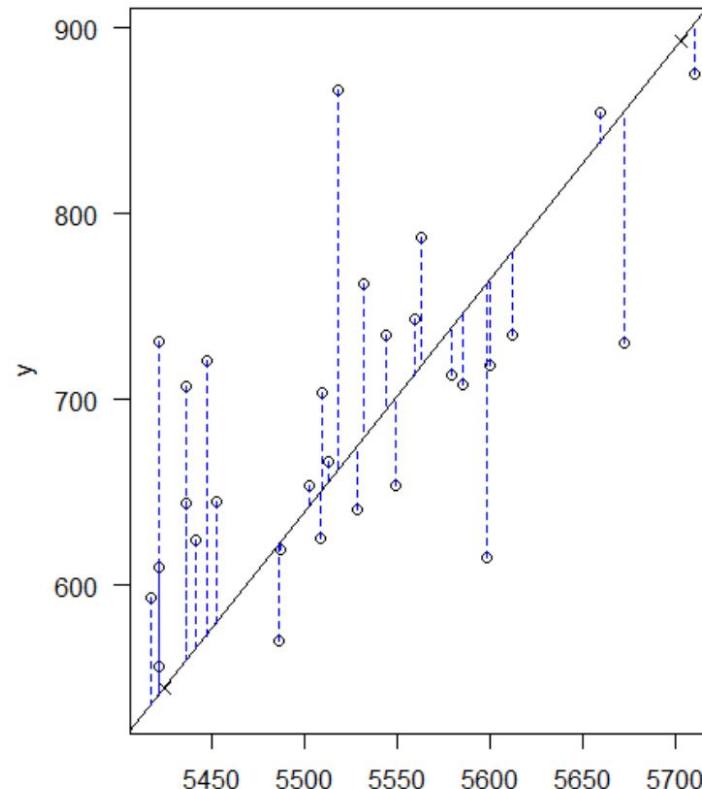
R^2

- 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: Understanding Linear Regression

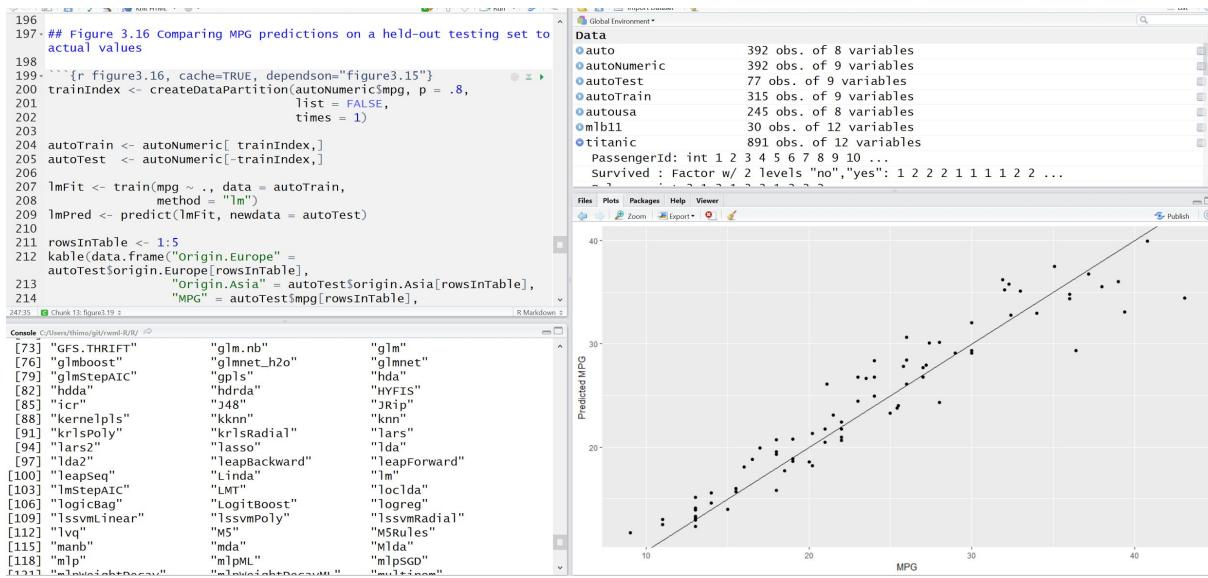
We will tour the “Simple Regression” lab of Open Intro Statistics

- Collinearity coefficient
- Sum of Squared Residuals
- Linear model fitting
- Reading the LM summary
- Model diagnostics:
 - linearity, nearly normal residuals, homoscedasticity



Lab: Using Linear Regression on the MPG set

We can use R and caret (<http://topepo.github.io/caret/index.html>) to visualise and assess visually the performance of a Linear Regression Model (lm):



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

Predicting numerical values with a regression model

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

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

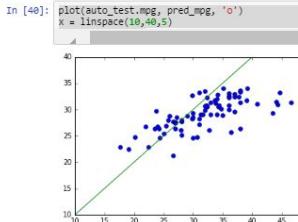
```
# Convert origin to categorical variable  
auto = auto.join(cat_to_num[auto['origin']])  
auto = auto.drop('origin', axis=1)  
  
# Split in train/test set  
auto_train = auto[int(0.8*len(auto)):]  
auto_test = auto[int(0.8*len(auto)):]
```

	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()  
regf = RandomForestRegressor()  
reg.fit(auto_train.drop('mpg', axis=1), auto_train['mpg'])
```

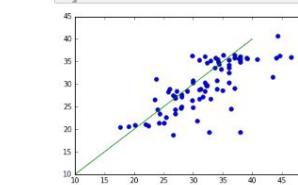
```
Out[38]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,  
                                max_features=None, max_leaf_nodes=None,  
                                min_impurity_split=1e-07, min_samples_leaf=1,  
                                min_samples_split=2, min_weight_fraction_leaf=0.0,  
                                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')
```

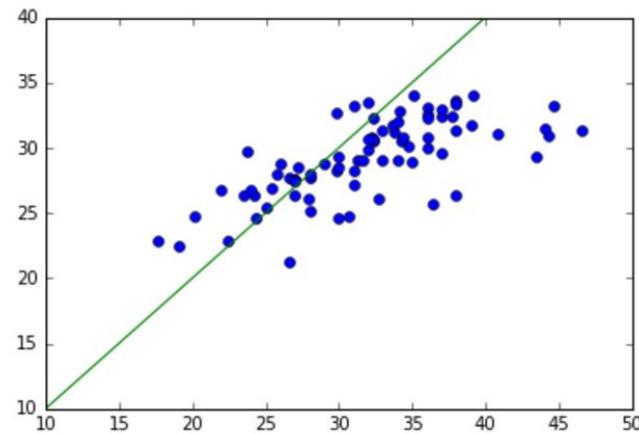
```
In [41]: pred_mpg_rf = regf.predict(auto_test.drop('mpg', axis=1))
```



Comparison of R-squared (model score)

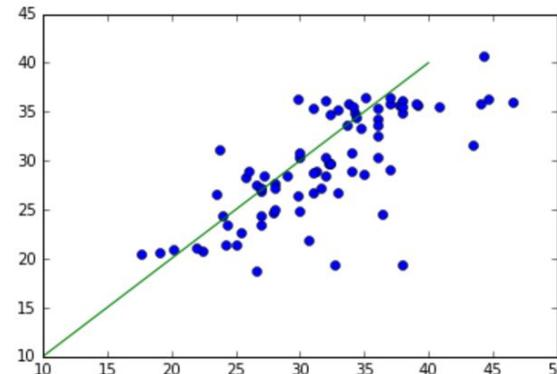
```
In [39]: pred_mpg = reg.predict(auto_test.drop('mpg',axis=1))
```

```
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
```

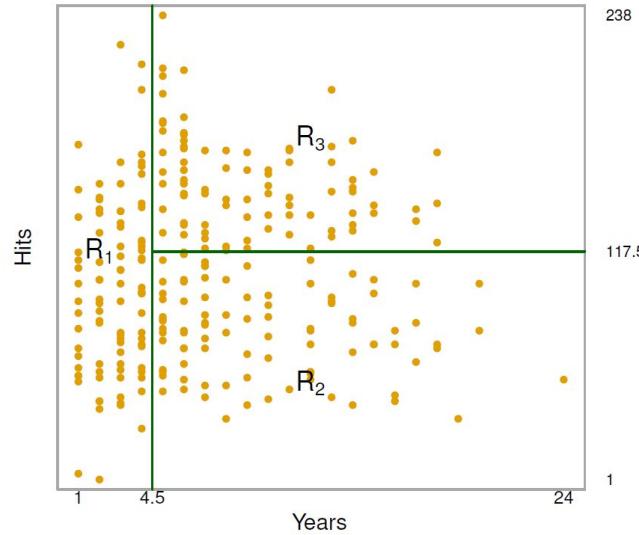
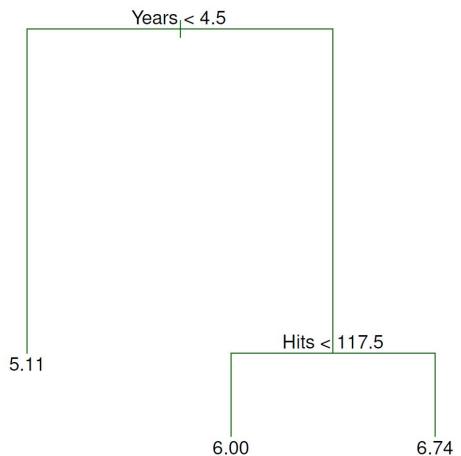
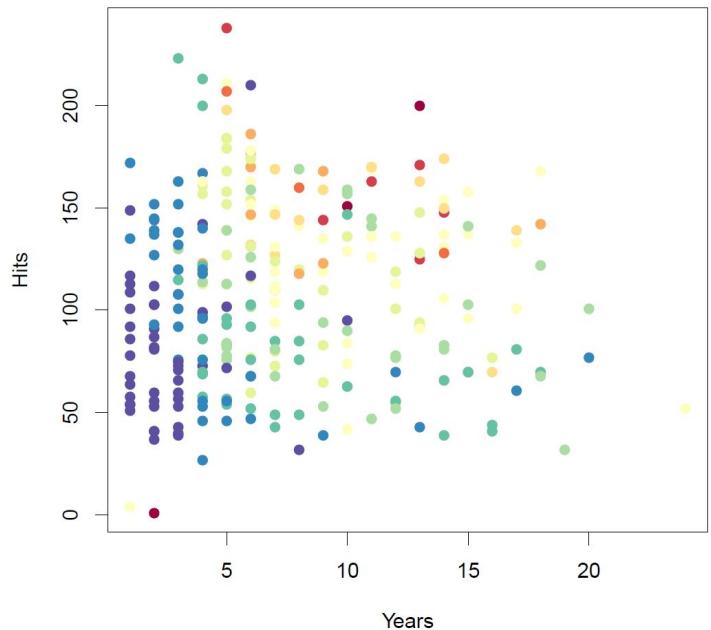
Tree-based methods

- 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, \dots, X_p — into J distinct and non-overlapping regions, R_1, R_2, \dots, 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, \dots, 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 j th box.

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 m th region that are from the k th class.

- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini index

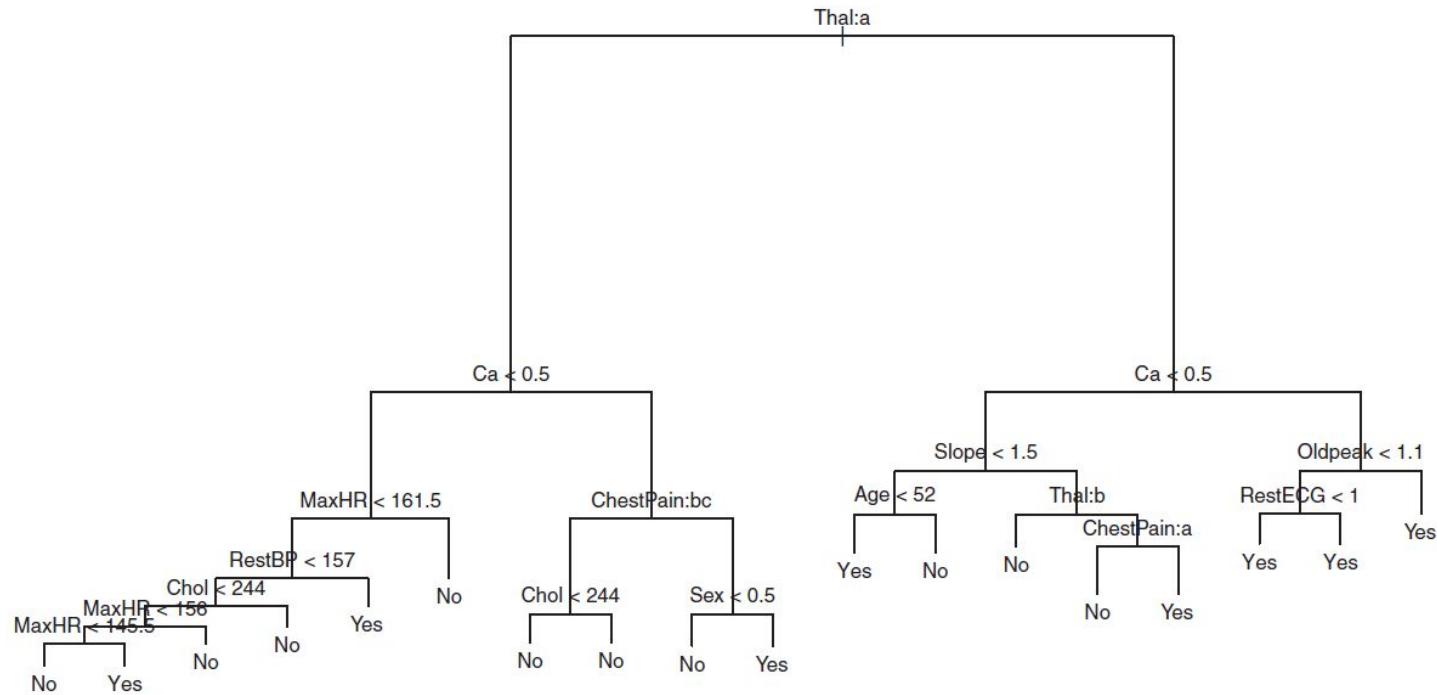
- The *Gini index* is defined by

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

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 seen in this book.

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, \dots, 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 b th 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

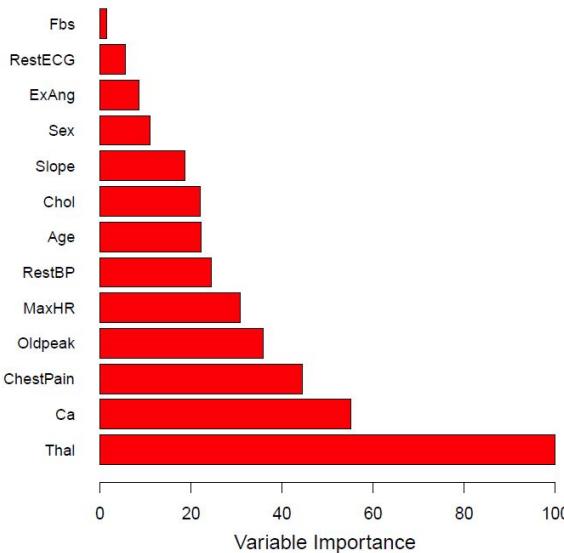
- The above prescription applied to regression trees
- For classification trees: for each test observation, we record the class predicted by each of the B trees, and take a *majority vote*: the overall prediction is the most commonly occurring class among the B predictions.

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

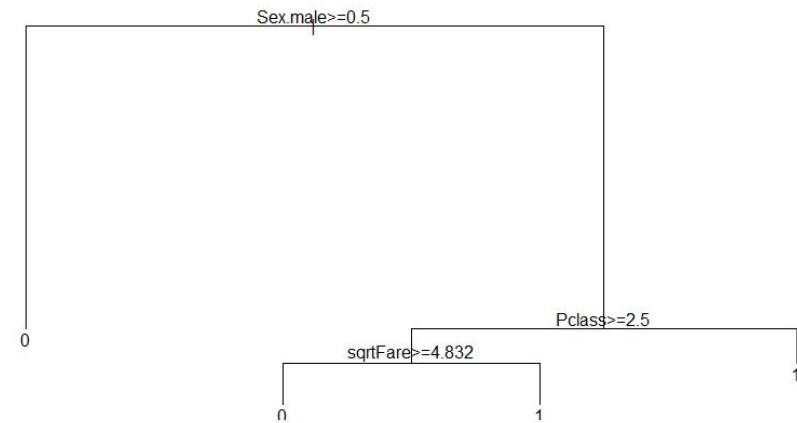
```
67 features = prepare_data(data_train)
68 features[:5]
69 model = DecisionTreeClassifier()
70 model.fit(features, data_train["Survived"])
71
72 #print model.decision_path(features)
73
74 tree_to_code(model, features.columns)
75 print model.score(prepare_data(data_test), data_test["Survived"])
76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

```
titanicTrain <- titanicTidyNumeric[ trainIndex,]
titanicTest  <- titanicTidyNumeric[-trainIndex,]

formula <- as.formula(Survived.yes ~.)
t <- train(formula,titanicTrain,method = "rpart",cp=0.002,maxdepth=8)
plot(t$finalModel)
text(t$finalModel)
summary(t$finalModel)
|
# make predictions
x_test <- titanicTest[,2:9]
y_test <- titanicTest[,1]
predictions <- predict(t, x_test)
# summarize results
confusionMatrix(predictions, y_test)
```



Printing the classification tree in R

```
variable importance
  Sex.male    sqrtFare     Pclass      Parch       Age       SibSp Embarked.Q
      53          16         13          7          5          3          3

Node number 1: 889 observations,      complexity param=0.4411765
predicted class=0  expected loss=0.3824522  P(node) =1
  class counts:   549   340
  probabilities: 0.618  0.382
left son=2 (577 obs) right son=3 (312 obs)
Primary splits:
  Sex.male < 0.5      to the right, improve=123.172200, (0 missing)
  Pclass    < 2.5      to the right, improve= 43.046950, (0 missing)
  sqrtFare < 3.237475 to the left,  improve= 37.430830, (0 missing)
  Embarked.S < 0.5     to the right, improve=  9.673680, (0 missing)
  Parch     < 0.5      to the left,   improve=  9.363975, (0 missing)
Surrogate splits:
  Parch     < 0.5      to the left,  agree=0.679, adj=0.087, (0 split)
  sqrtFare < 8.810365 to the left,  agree=0.678, adj=0.083, (0 split)

Node number 2: 577 observations
predicted class=0  expected loss=0.1889081  P(node) =0.6490439
  class counts:   468   109
  probabilities: 0.811  0.189

Node number 3: 312 observations,      complexity param=0.03088235
predicted class=1  expected loss=0.2596154  P(node) =0.3509561
  class counts:    81   231
```

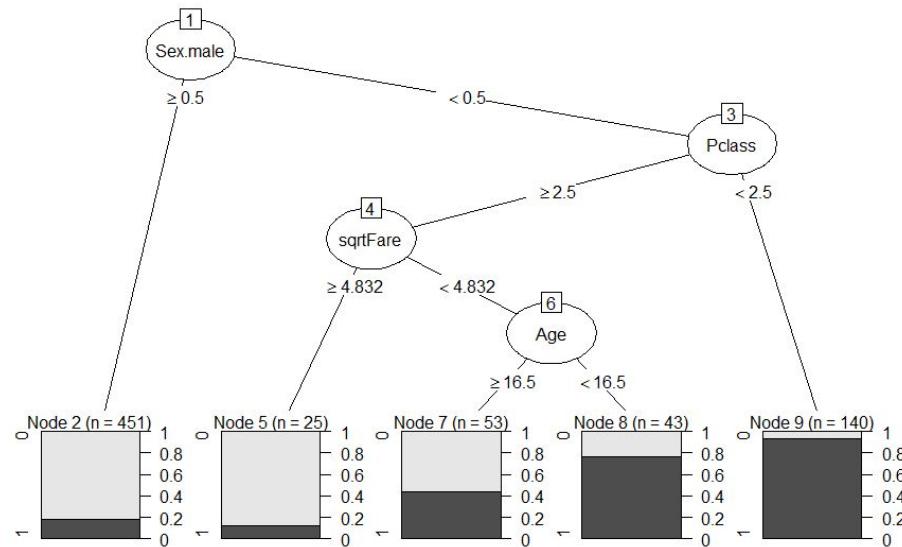
Enhanced tree plotting and visualisation

```
# Plot with partykit and using specific classifier method
rpart1 <- rpart(formula,titanicTrain)
library(partykit)
tparty <- as.party(rpart1)
plot(tparty)

> rpart1
n= 712

node), split, n, loss, yval, (yprob)
 * denotes terminal node

1) root 712 272 0 (0.61797753 0.38202247)
  2) Sex.male>=0.5 451 82 0 (0.81818182 0.18181818) *
  3) Sex.male< 0.5 261 71 1 (0.27203065 0.72796935)
    6) Pclass>=2.5 121 59 0 (0.51239669 0.48760331)
      12) sqrtFare>=4.832173 25 3 0 (0.88000000 0.12000000) *
      13) sqrtFare< 4.832173 96 40 1 (0.41666667 0.58333333)
        26) Age>=16.5 53 23 0 (0.56603774 0.43396226) *
        27) Age< 16.5 43 10 1 (0.23255814 0.76744186) *
    7) Pclass< 2.5 140 9 1 (0.06428571 0.93571429) *
```



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

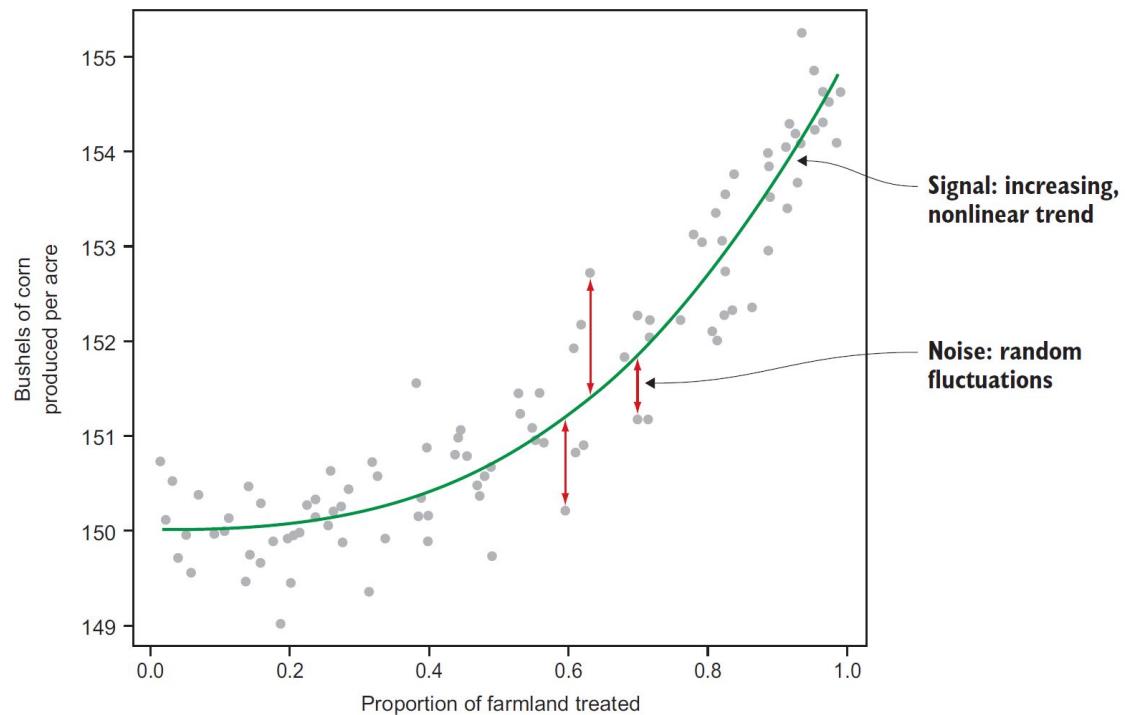
The general technique is called Cross-Validation, and it involves separating the data in training and test data.

Overfitting and Model Optimism are typical problems.

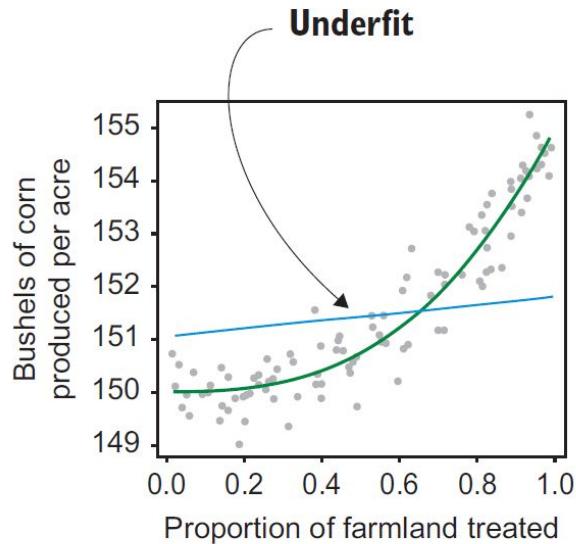
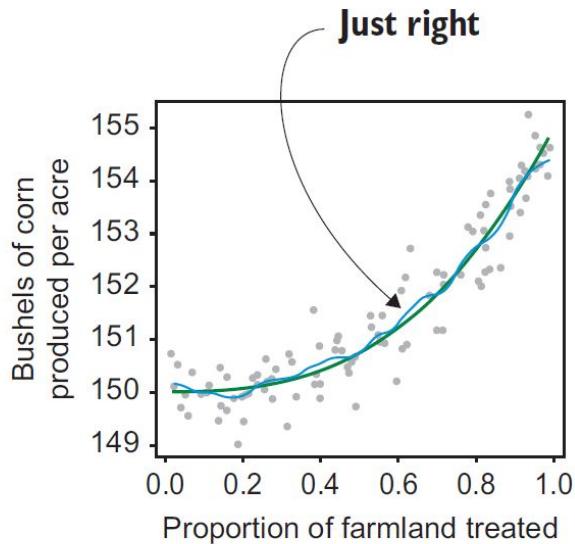
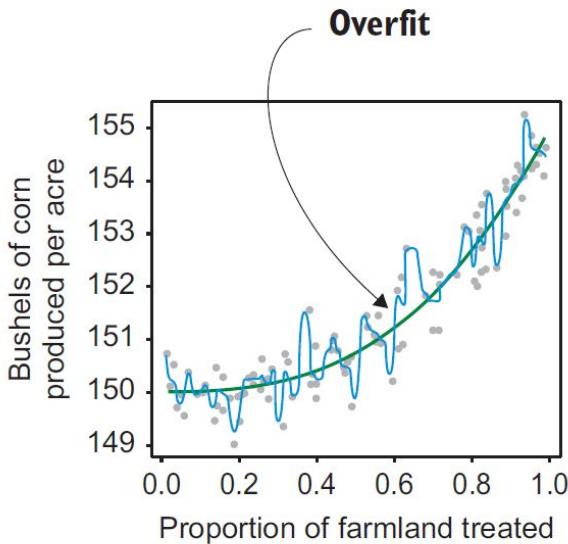
Let's look at an example

Performing a non-linear regression

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



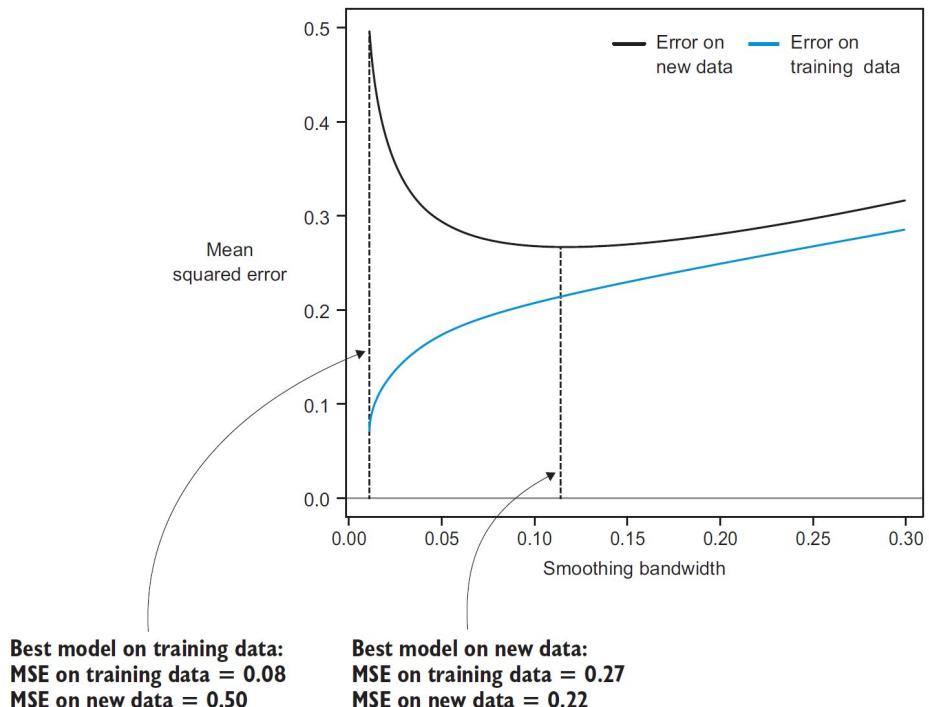
Tuning bandwidth in Kernel Smoothing



Increasing bandwidth parameter

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.

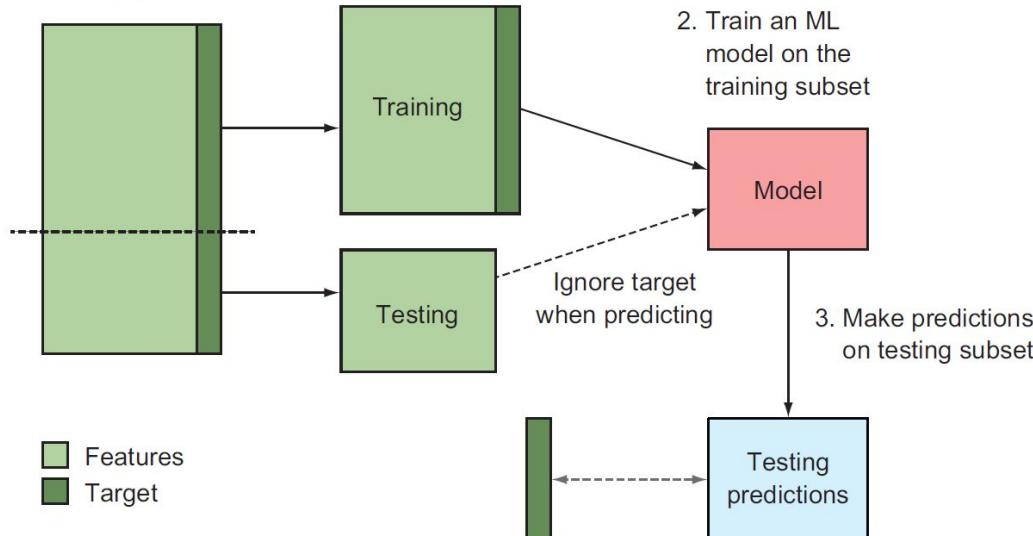
Practitioners typically leave out 20–40% of the data as the testing subset.

```
# 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]             | Splits index  
idx_test = idx[N_train:]  
  
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

1. Randomly split training instances into training and testing subsets



2. Train an ML model on the training subset

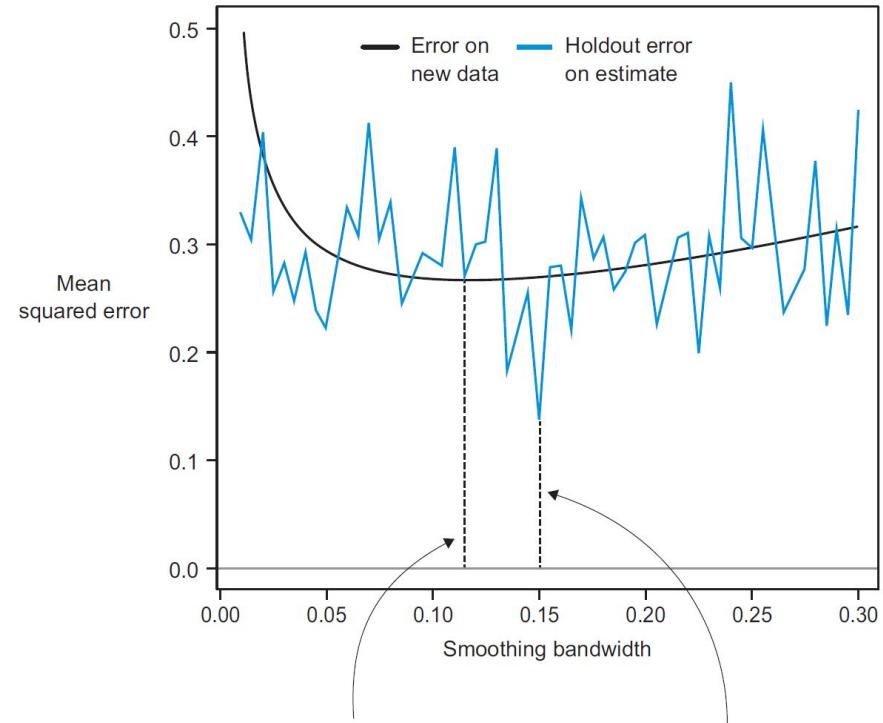
3. Make predictions on testing subset

4. Comparing testing predictions to testing target to assess accuracy

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.



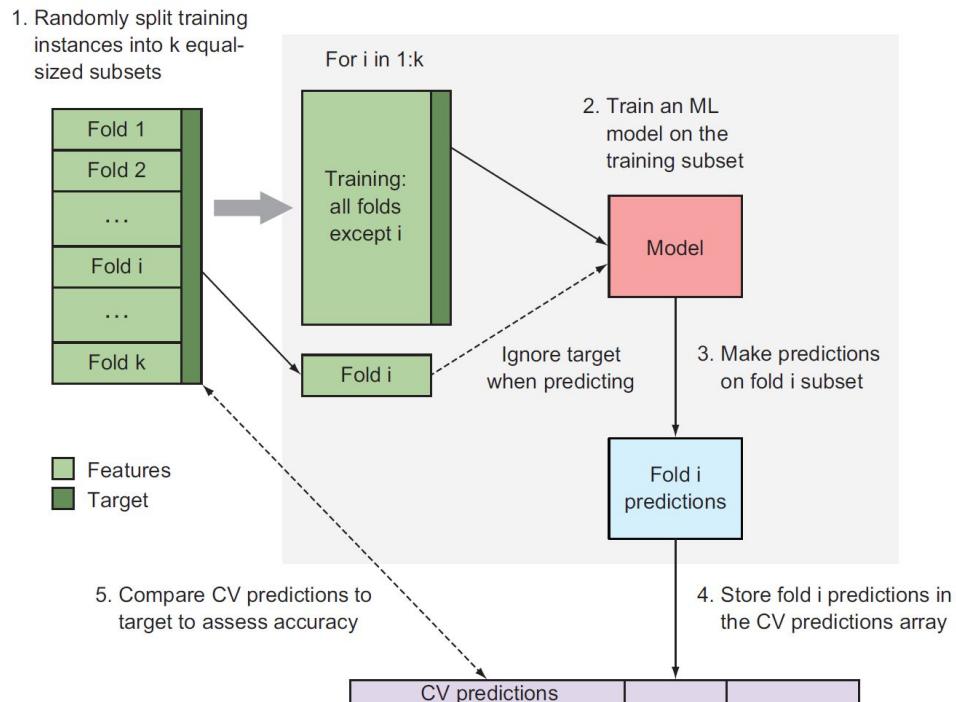
Best model on new data:
Holdout MSE = 0.30
MSE on new data = 0.27

Best holdout model:
Holdout MSE = 0.14
MSE on new data = 0.27

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

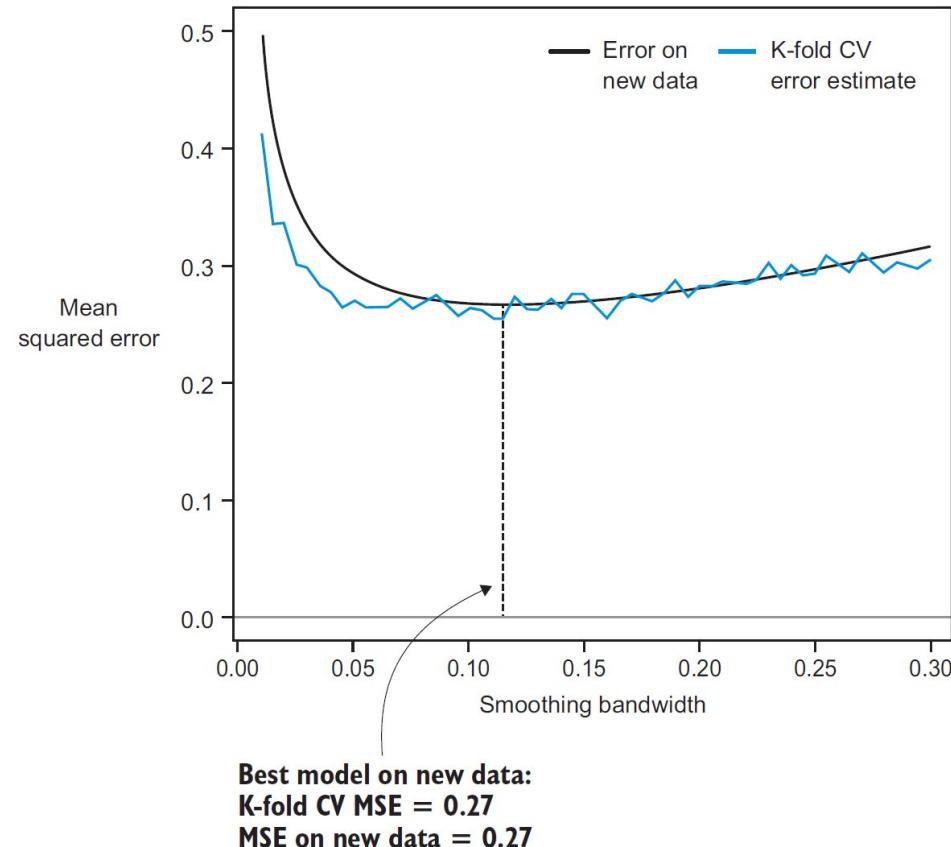
After all k-folds are cycled through, the predictions for each fold are aggregated and compared to the true target variable to assess accuracy



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) : ← Loops over  
the folds

    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)

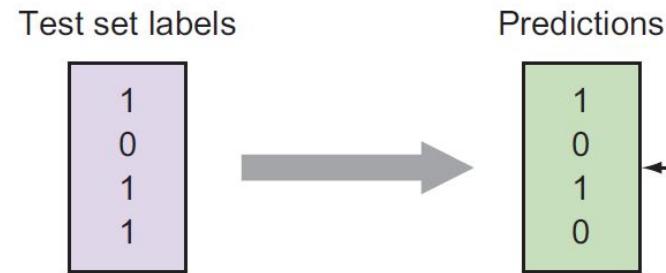
    # accuracy = evaluate_acc(preds_kfold, target)
```

**Breaks your data
into training and
testing subsets**

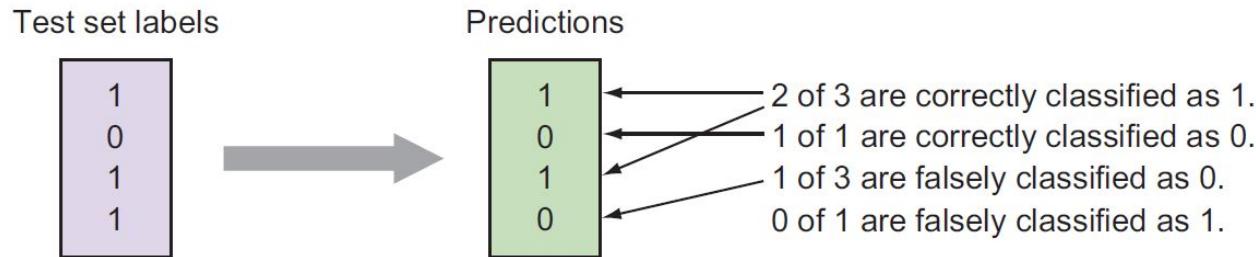
Evaluation of classification models

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



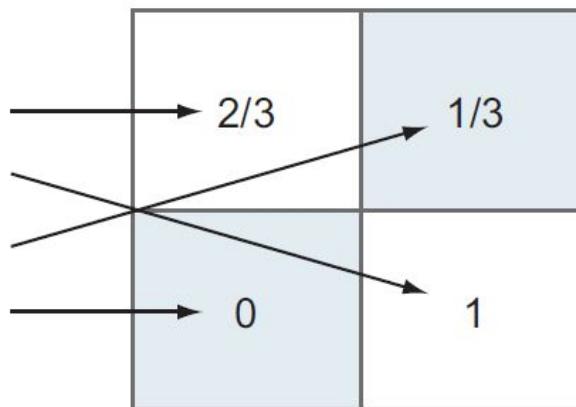
The predictions provide more information than simply being correct or not, and analyze the accuracy per class. For binary classification, you can be wrong in two ways:

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*

- 2 of 3 are correctly classified as 1.
- 1 of 1 are correctly classified as 0.
- 1 of 3 are falsely classified as 0.
- 0 of 1 are falsely classified as 1.



Receiver operating characteristics (ROCs)

True-positive rate
(a.k.a. sensitivity)

		Predicted class	
		1	0
True class	1	2/3	1/3
	0	0	1

False-negative rate
(a.k.a. miss rate)

False-positive rate
(a.k.a. specificity)

Whether we are predicting
the positive or negative class

True/False

positive/negative

rate

Whether prediction is
correct or incorrect

Out of the total
number of positive
or negative records

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



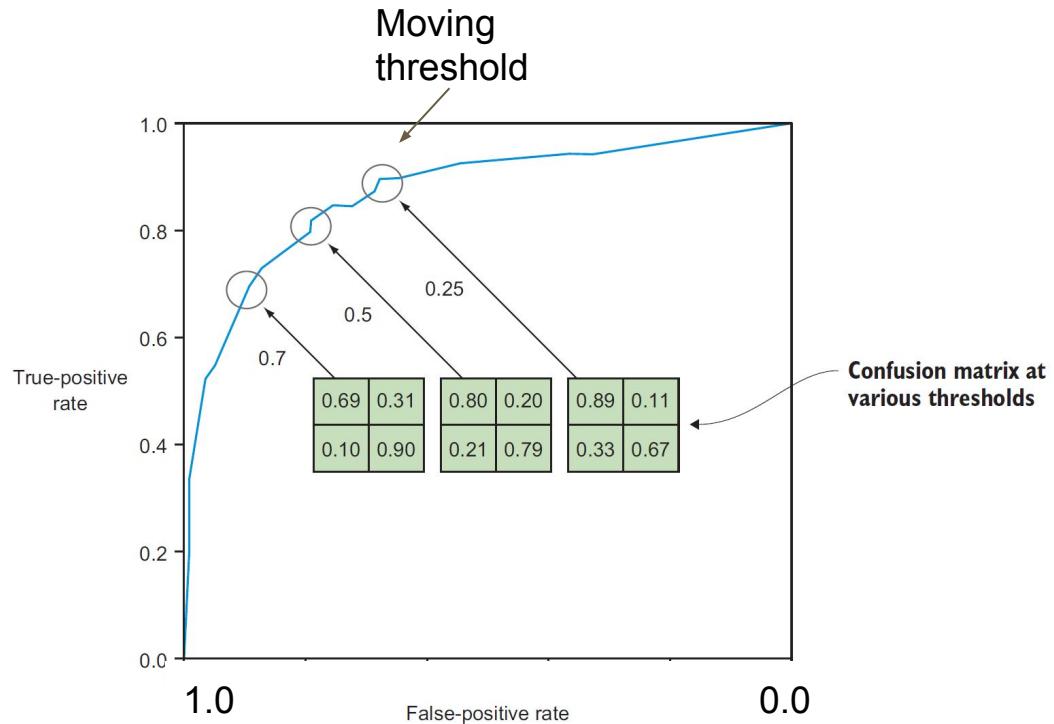
Threshold: “survived”
probabilities > 0.7

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

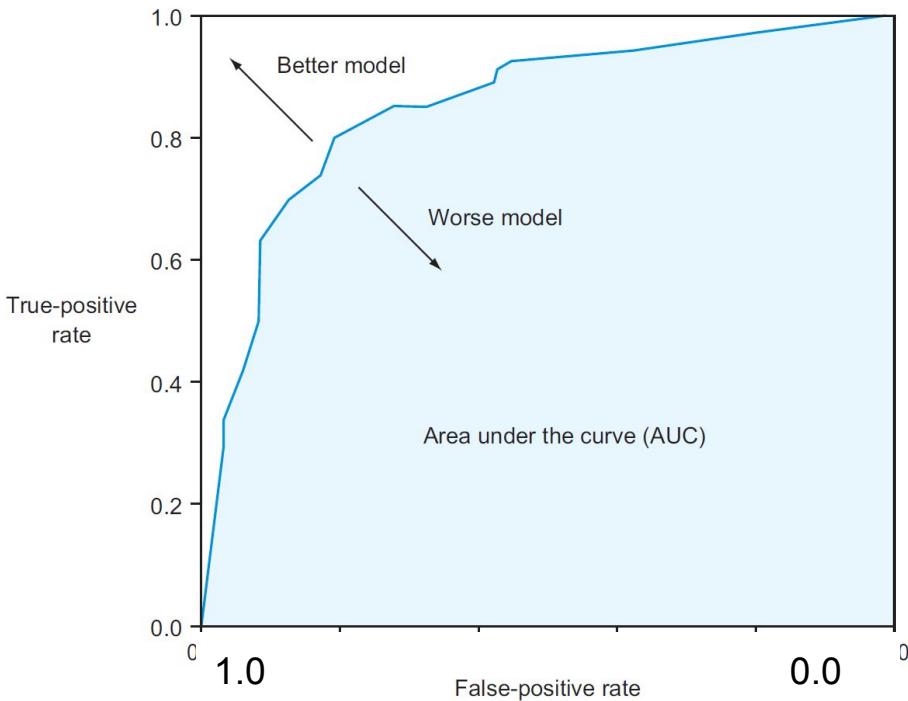
Precalculates values for the positive and negative cases, used in the loop

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.

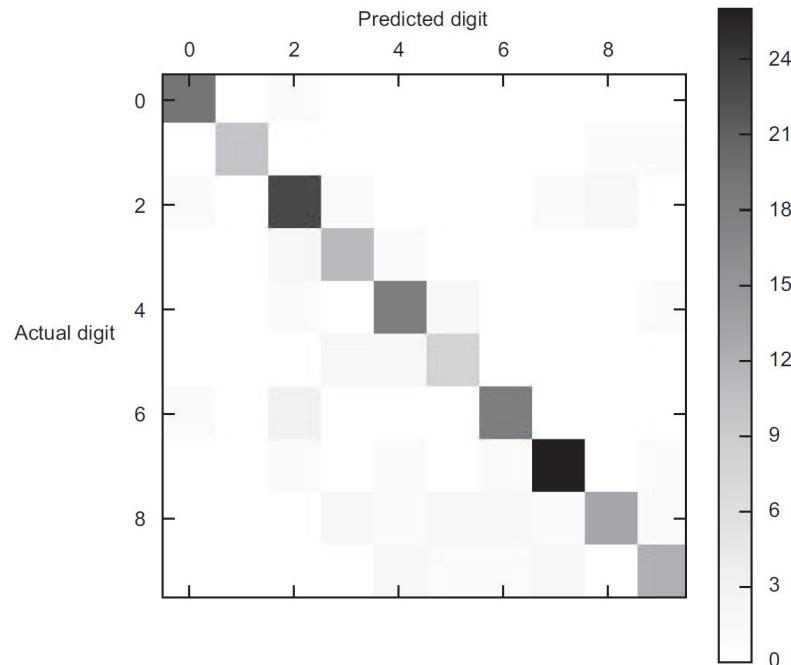


Multiclass Classification

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.

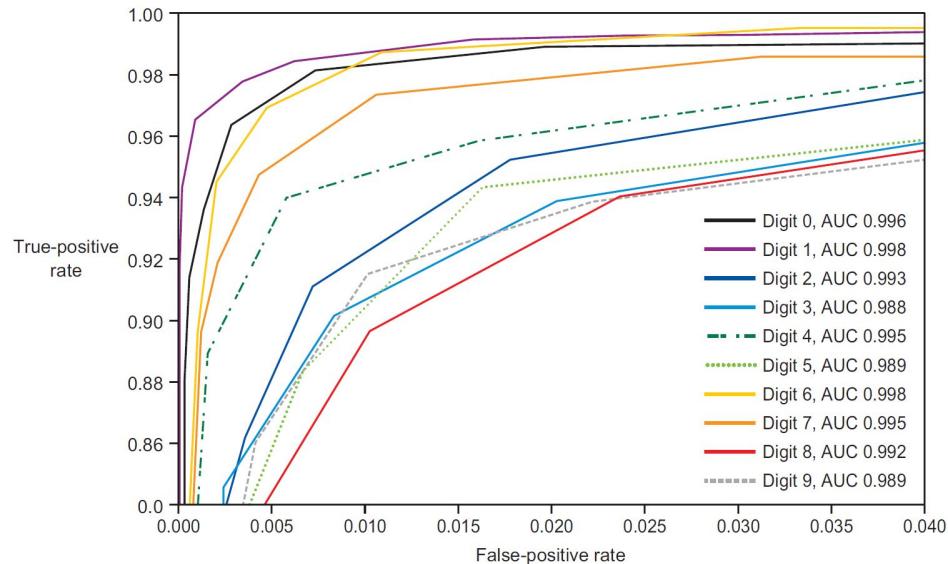
Areas shaded outside the diagonal provide a visual idea of the confusion in prediction



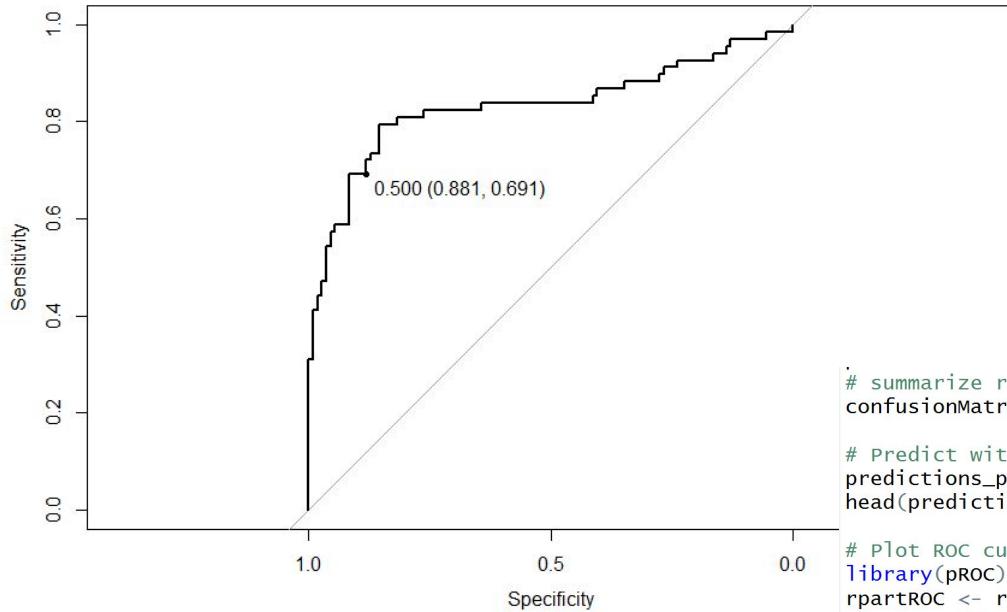
Multiclass ROC

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 Classification Tree in R



```
> confusionMatrix(predictions, y_test)
Confusion Matrix and Statistics

Reference
Prediction   0   1
      0 101  27
      1    8  41

Accuracy : 0.8023
95% CI  : (0.7359, 0.8582)
No Information Rate : 0.6158
P-Value [Acc > NIR] : 7.432e-08

# summarize results
confusionMatrix(predictions, y_test)

# Predict with probabilities
predictions_probs <- predict(t, x_test, type="prob")
head(predictions_probs)

# Plot ROC curve
library(pROC)
rpartROC <- roc(y_test, predictions_probs[, "0"], levels = c("0", "1"))
plot(rpartROC, type = "s", print.thres = .5)
```

Performance Statistics

Confusion Matrix and Statistics

		Reference
Prediction	0	1
0	97	21
1	12	47

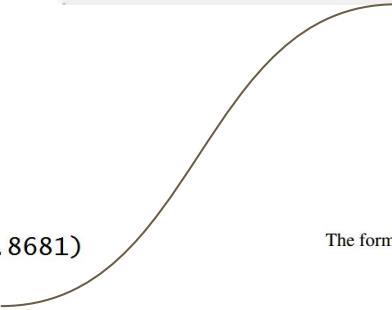
Accuracy : 0.8136
95% CI : (0.7483, 0.8681)
No Information Rate : 0.6158
P-value [Acc > NIR] : 1.058e-08

Kappa : 0.5959
McNemar's Test P-Value : 0.1637

Sensitivity : 0.8899
Specificity : 0.6912
Pos Pred Value : 0.8220
Neg Pred Value : 0.7966
Prevalence : 0.6158
Detection Rate : 0.5480
Detection Prevalence : 0.6667
Balanced Accuracy : 0.7905

'Positive' Class : 0

a one-sided test to see if the accuracy is better than the "no information rate," which is taken to be the largest class percentage in the data.



		Reference
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$$\text{Sensitivity} = A/(A + C)$$

$$\text{Specificity} = D/(B + D)$$

$$\text{Prevalence} = (A + C)/(A + B + C + D)$$

$$\text{PPV} = (\text{sensitivity} * \text{prevalence}) / ((\text{sensitivity} * \text{prevalence}) + ((1 - \text{specificity}) * (1 - \text{prevalence})))$$

$$\text{NPV} = (\text{specificity} * (1 - \text{prevalence})) / (((1 - \text{sensitivity}) * \text{prevalence}) + ((\text{specificity}) * (1 - \text{prevalence})))$$

$$\text{DetectionRate} = A/(A + B + C + D)$$

$$\text{DetectionPrevalence} = (A + B)/(A + B + C + D)$$

$$\text{BalancedAccuracy} = (\text{sensitivity} + \text{specificity})/2$$

$$\text{Precision} = A/(A + B)$$

$$\text{Recall} = A/(A + C)$$

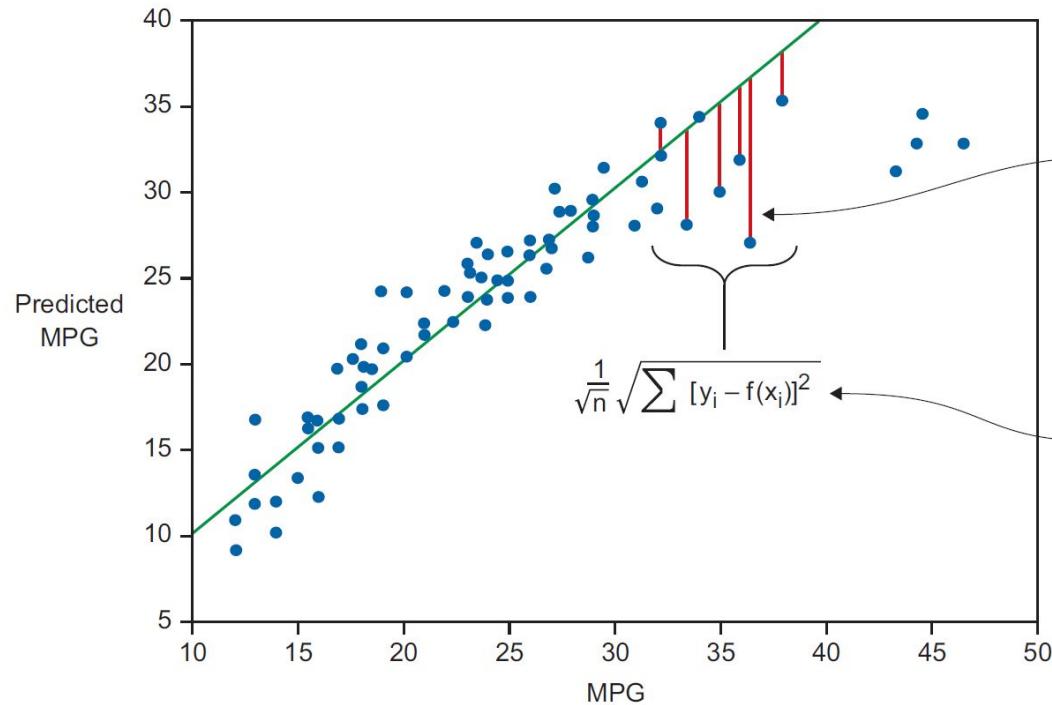
$$F1 = (1 + \beta^2) * \text{precision} * \text{recall} / ((\beta^2 * \text{precision}) + \text{recall})$$

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



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.

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
folds = np.random.randint(0, K, size=N)           ← Number of cross-validation folds

# 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)

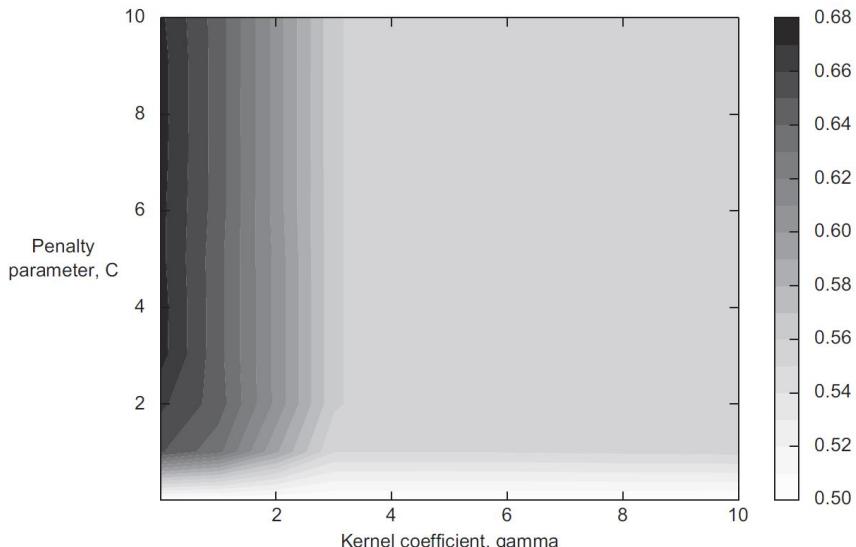
    # evaluate the AUC of the predictions
    AUC_all.append(roc_auc_score(y, y_cv_pred))
```

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 ($\text{gamma} = 0.01$), 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 $\text{gamma} = 0$, 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

Target variable Categorical variable

interested	invited	birthyear	gender	timezone	lat	long
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

Selection data User data Event data

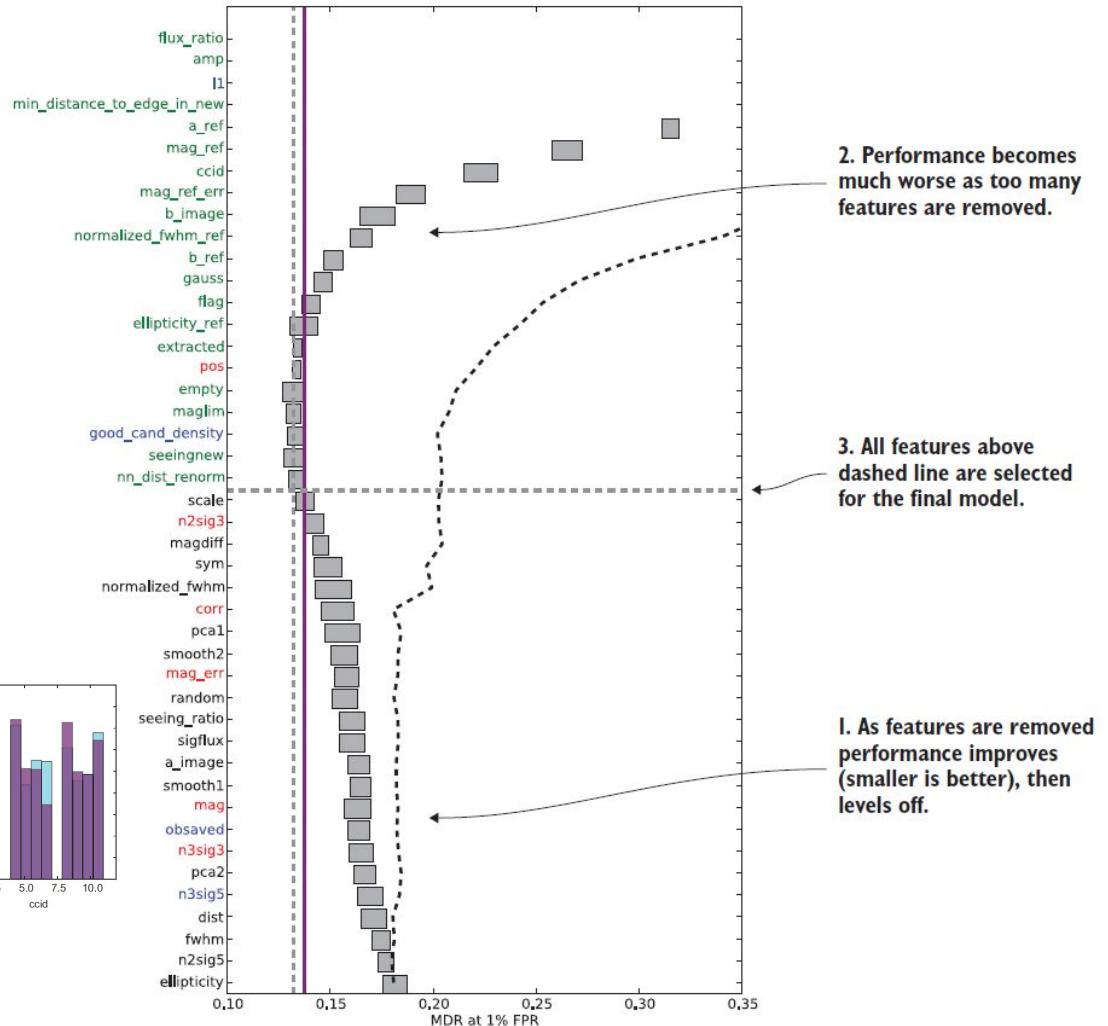
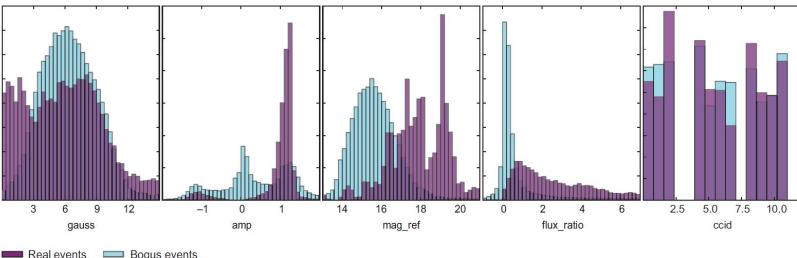
Missing data

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

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

Feature Selection

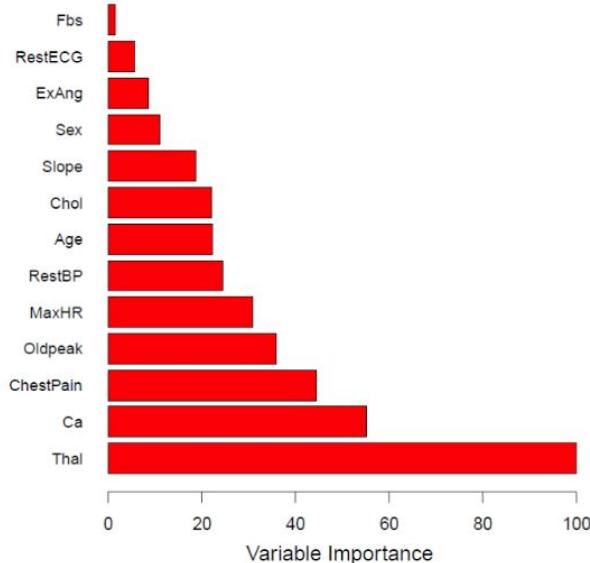
Feature selection can be performed with forward selection or backward elimination. Some models already have built-in processes for feature selection.



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/>