Datacamp Python

# Statistical Thinking in Python Part 1

## Exploratory Data Analysis

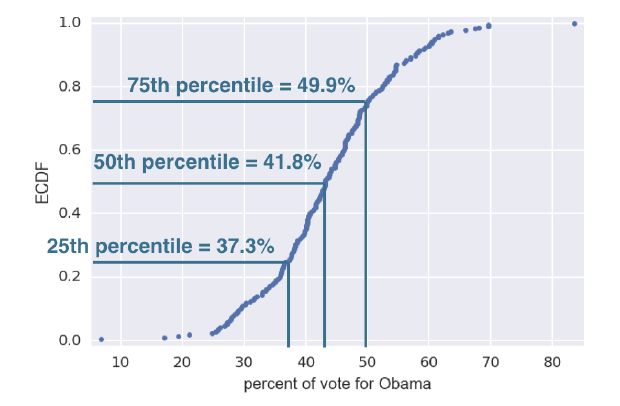
Histogram, bins and binning bias.

Bee warm plot and ECDFs

### ECDF

Empirical Cumulative Distribution Function is the distribution function associated with the empirical measure of a sample. The x axis are the observations, sorted from the smallest to the largest. The y axis is the percentage of observation smaller than or equal to the specified value.

For example



50% percentage of counties has the value less than or equal to 41.8%.

If a country has 41.8% votes for Obama, means it supports Mc Cain.

75th percentile of country in this state support Mc Cain.

## Quantitative exploratory data analysis

### Mean and median

### Percentiles, outliers and box plots

### Variance and standard deviation

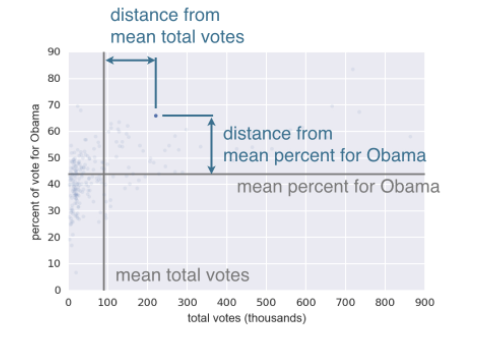
* Variance, the mean squared distance of the data from their mean
* Informally, a measure of the spread of the data

**Population variance** is given by σ2 (delta)

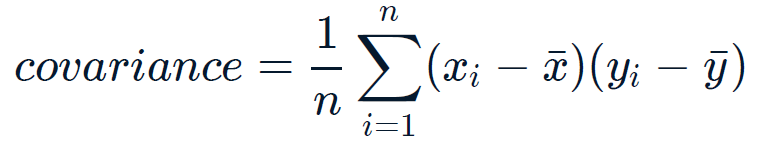
Population standard deviation is the square root of variance.

### Covariance and the Pearson correlation coefficient

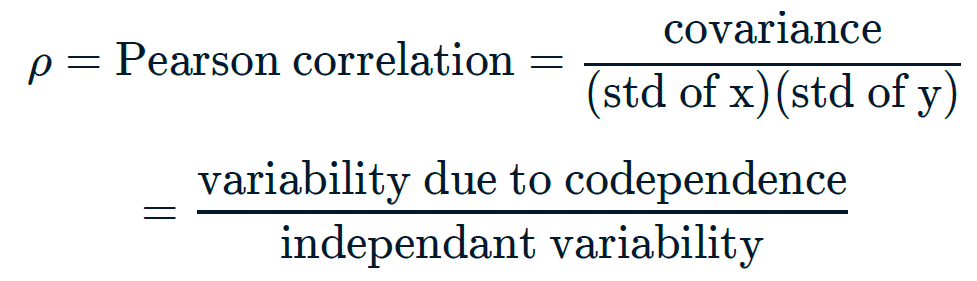
Covariance is when the two variables vary together. For example, when x is distant from x̄, is y distant from ȳ?



The formula for covariance is:



Pearson correlation discounts the units of x and y and is dimensionless.



## Thinking probabilistically – Discrete variables

**Probability** allows us to describe **uncertainty**.

When we measure 50 flowers for their pedal length, we get a mean value. But if I measure another 50 flowers of the same species, what would the mean pedal length be?

This is the heart of **statistical inference**. It is the process by which we go from measured data to probabilistic conclusion about what we might expect if we collected the same data again.

### Binomial Random Variable

In order for a variable to be a binomial random variable,

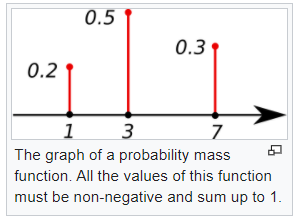
* Each trail must be independent
* Each trail can be called a “success” or a “failure”
* There are a fixed number of trials
* The probability of success on each trial is constant

### Bernoulli trial

An experiment that has two options, “success” and “failure”.

### Probability Mass Function (PMF)

A probability mass function is a function that gives the probability that a discrete random variable is exactly equal to some value.



P(X = 1) = 0.2

P(X = 3) = 0.5

P(X = 7) = 0.3

The discrete random variable X has 0.5 probability to have the value 3.

### Binomial Distribution

The number r of success in n Bernoulli trials with probability of p of success, is Binomially distributed.

The number of r heads in 4 coin flips with probability of 0.5 of heads, in binomially distributed.

np.random.binomial(n, p, size)

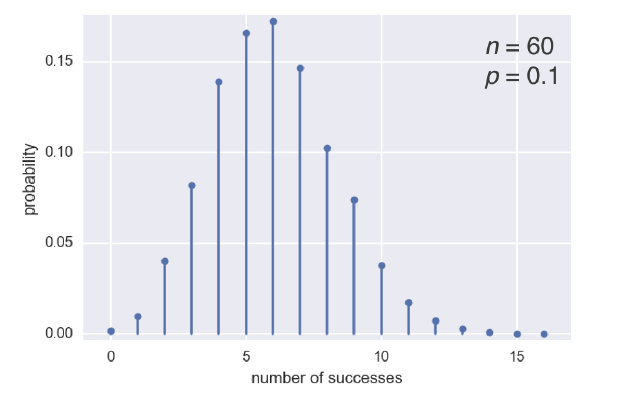
n: n Bernoulli trails

p: probability of success

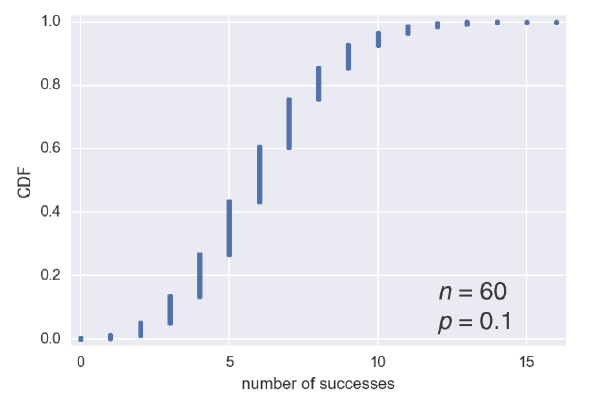
r: return the number of success of N, p

size: how many simulations to run / how many samples to draw from a binomial distribution.

##### The following is the Binomial PMF



##### And the Binomial CDF



### The Poisson Process

A Poison process describes the number of times an event occurs in a period of time, or in a particular area, or over some distance or within any other kind of measurement.

* The experiment counts the number of occurrences of an event over some other measurement.
* The mean is the same for each interval
* The count of events in each interval is independent of the other intervals
* The intervals don’t overlap

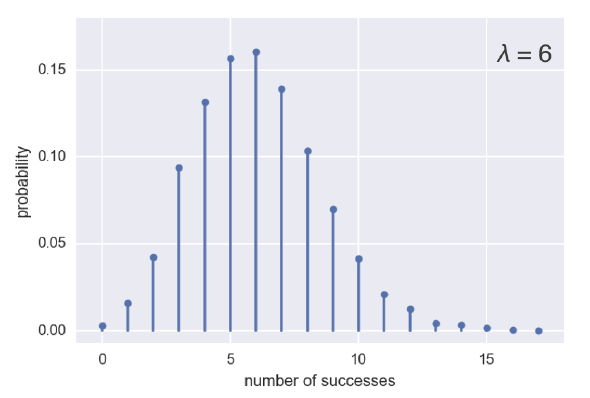
### Poisson distribution

“In his great book on information theory, statistical inference, and machine learning, David MacKay described a town called Poissonville”

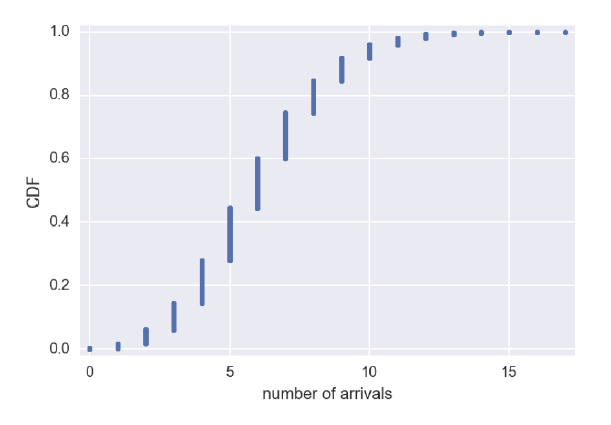
The number **r** of arrivals of a Poisson process in a given time interval with average rate of x arrivals per interval is Poisson distributed.

The number **r** of hits on a website in one hour with an average hit rate of 6 hits per hour is Poisson distributed.

#### Poisson PMF



#### Poisson CDF



Poisson distribution is an approximation of the binomial distribution when

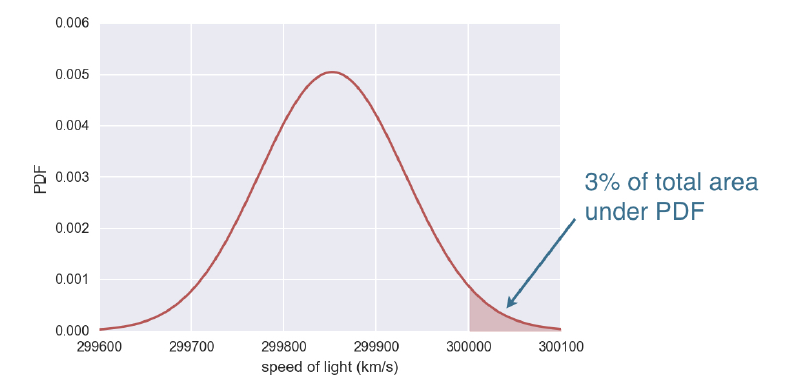
* the number of trials is at least 20
* the probability of success is less than 0.05

i.e., for rare events

## Thinking probabilistically – Continuous variables

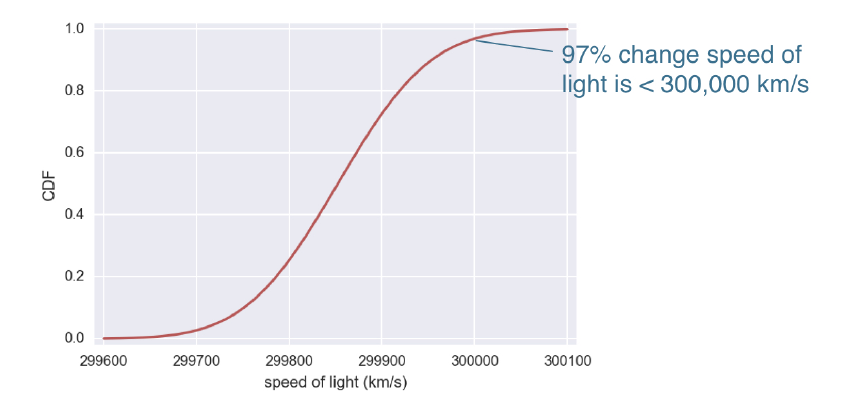
### Probability Density Function (PDF)

Continuous analog to the PMF. It describes the probability of observing a value of a continuous variable.



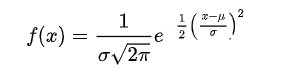
The probability of observing the light whose speed is greater than 300,000 km/s is 3%, the total area under the PDF

On CDF



### The Normal Distribution

The normal (or Gaussian) distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is

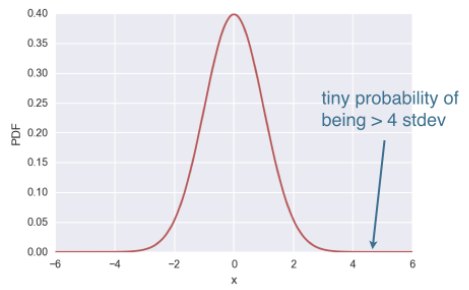


Where mu is the mean and sigma is the standard deviation

### The Normal Distribution: Properties and Warnings

Not all data is normally distributed.

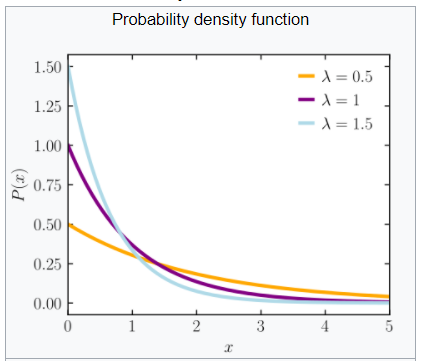
A normal distribution has light tails, that is, values outside 3 stds are extremely unlikely. However, in reality, some data sets have many outliers and therefore, normal distribution might not be a good description of the data.

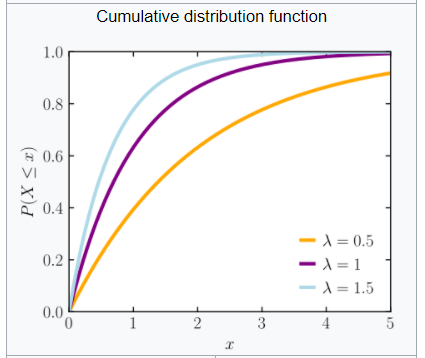


### The Exponential Distribution

The exponential distribution is the probability of the time between events in a Poisson point process.

The waiting time between arrivals of a Poison process is exponentially distributed.





# Statistical Thinking in Python Part 2

## Parameter estimation by optimization

Optimal parameters bring the model closest agreement with the data, given that the model we choose is the correct model.

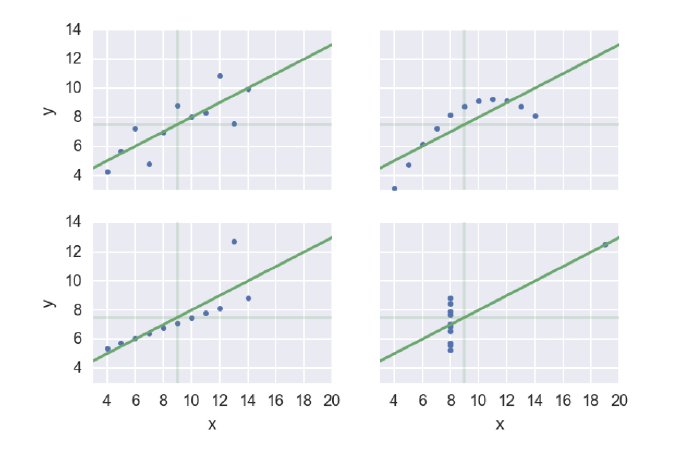
### Linear Regression

Linear regression is the process of using a linear model with point-intercept form to approximate the data.

Least-squares is the process of finding the parameters for which the sum of the squares of the residuals is minimal.

numpy.polyfit(x, y, degree = 1) can be used for linear regression

### Anscombe’s quarter



The four data sets have the same mean value of x, mean value of y, residual and line-slope formulas.

Therefore, perform EDA first.

## Bootstrap confidence intervals

**Resampling** is the process of random selecting observations from the data set (with replacements) as if we are re-conducting the experiment.

**Bootstrapping** is the use of resampled data to perform statistical inference.

A resampled array of data is a **bootstrap sample**.

A statistic computed from a resampled array is called **bootstrap replicate**.

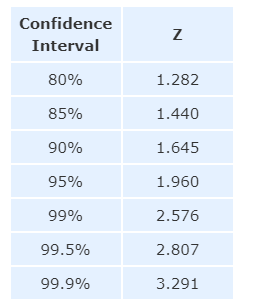
### Confidence interval of a statistic

If we repeat the measurements over and over again, p% of the observed values would lie within the p% confidence level.

**Example in theory**

The average height of men is 176cm, with a standard deviation of 20cm. What is the 95% confidence interval?

We find the z score of 2.5% and 97.5%



Plug in the z-value into the formula



And we have 175 +/- 6.2

Therefore, the interval [168.8, 181.2] includes 95% of the observations.

**Example in simulation**

****

We draw bootstrap replicas from the sample data

Take the percentiles to get the confidence interval.

### Pair bootstrap

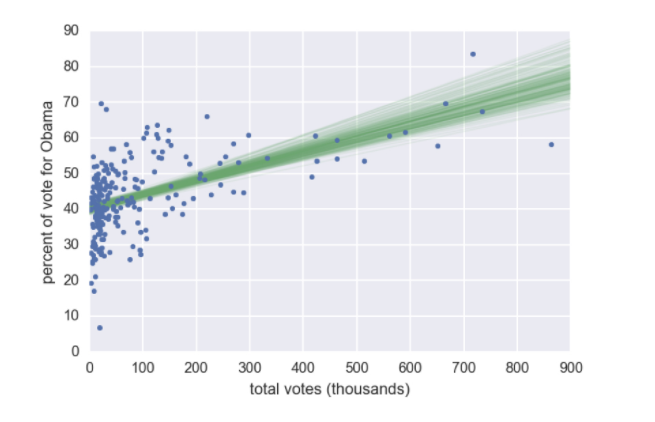
So far, we have been resampling with no assumption about the model or probability distribution of the underlying data, a.k.a, the resampling have been make on the data along.

If we want to perform statistical inference of a linear regression model, we have to consider two parameters, the slope and the intercept.

### Pairs bootstrap for linear regression

* Resample data in pairs (using indices)
* Computer slope and intercept from resampled data
* Each slope and intercept is a bootstrap replicate
* Compute confidence intervals from percentiles of bootstrap replicates.

The results look like this

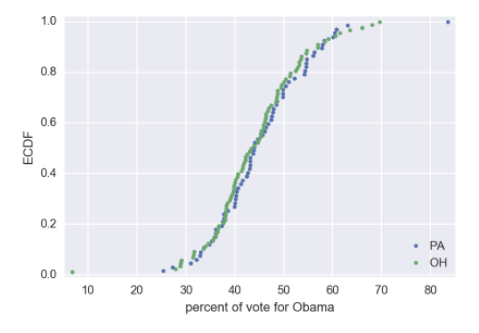


## Introduction to Hypothesis Testing (Datacamp)

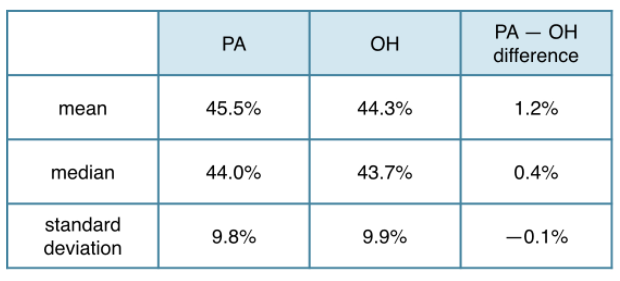
Hypothesis testing is the assessment of how reasonable the observed data are assuming a hypothesis is true.

Ohio and Pennsylvania are similar states. We can hypothesize that the county-level voting in these two states have identical probability distributions. We call this the null hypothesis.

From the ECDFs, the graphs “seem” to be identical.



The summary statistics are also close to each other.

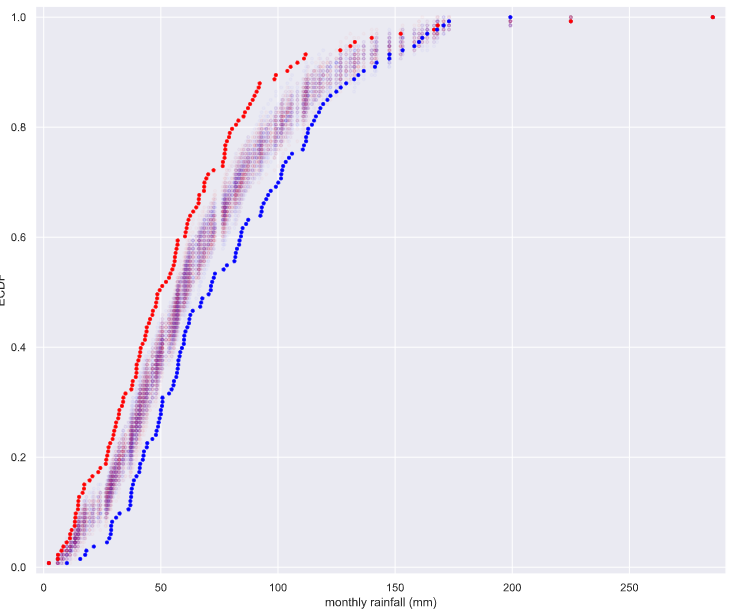


But do we have a scientific way to solid statistical inference about the null hypothesis?

### Permutation of samples

In the case where we have two sample groups and we want to find out if the two groups are identical or not. We merge the two groups together, scramble the orders and randomly label them as group A and group B, as if they are identical. Then we compute test statistics on the permutation samples and compare it to the test statistics on the observed sample.

For example, we hypothesis that the rain fall in the month of June (a wet month) are identical to the rainfall of November (a dry month)



We permutate the samples and plots the ECDFs, it turns out that the ECDF’s of the permutated samples lay between the ECDF of June and November. Is there a **quantities measurement** to help us to make a judgement here?

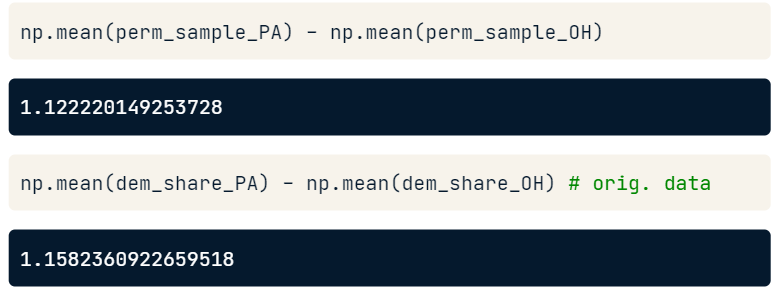
### Test statistics and p-values

Test statistics are

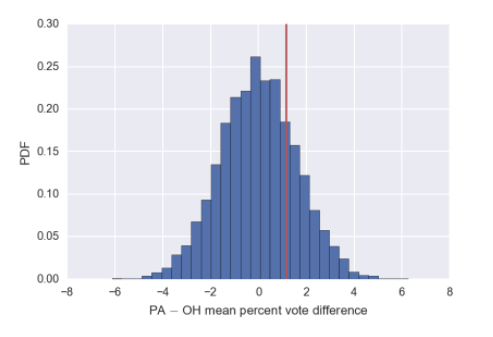
* A single number that you can compute form observed data and from data you simulate under the null hypothesis.
* It serves as a basis of comparison between the two
* The test statistics should be chosen so it is pertinent to the question that we are trying to answer.

The hypothesis is that the votings are identical in two states. Therefore, the difference of means should be zero (or very low). Hence, we choose difference of means as test statistics.

We compute permutation replicate 1.122 and compare it to the actual difference of means 1.158



We draw many permutation replicates and plot the distribution



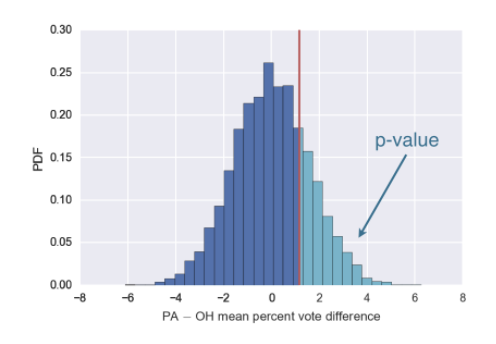
The red line is the observed test statistics, it lay with in 2 standard deviations in the distribution. We are inclined to say that it is reasonable to observed the test statistics if the hypothesis is true. However, the distribution of difference of means is the PDF, it is difficult to measure the probability of the single point, therefore, we re-formulate the statement to, “Is it reasonable to obtain the test statistics that is equal to or greater than the observed value?”

#### p-values

We could also tally up the area to the right of the red line We get that about 23% of the simulated elections had at least 1.16% difference of greater.

We call this 23% the p value.

It is the probability of obtaining a value of your test statistic that is at least as extreme as what was observed, under the assumption the null hypothesis is true.



#### Statistical significance

If the p value is small, we say that the data are statistically significantly different than what we would observe under the null hypothesis.

For this reason, the hypothesis testing are sometimes called the null hypothesis significance testing, or **NHST**.

### Pipeline for hypothesis testing

* Clearly state the null hypothesis
* Define your test statistics
* Generate many sets of simulated data assuming the null hypothesis is true
* Compute the test statistic for each simulated data set
* The p-value is the fraction of your simulated data sets for which the test statistic is at least as extreme as for the real data.

## Introduction to hypothesis testing (Udemy)

<https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/>

Example 1:

Percentage of vote to democratic party in counties among Ohio and Pennsylvania are similar.

* Use permutation to randomly reorder two arrays as if they are the same.

### P value

P value is used in the hypothesis testing to help you support or reject the null hypothesis: The p value is the evidence against a null hypothesis. The smaller the p-value, the stronger the evidence that you should reject the null hypothesis.

P-value is the probability of observing a test statistic equally or more extreme than the one you observed, given that the null hypothesis is true.

### The Alpha value

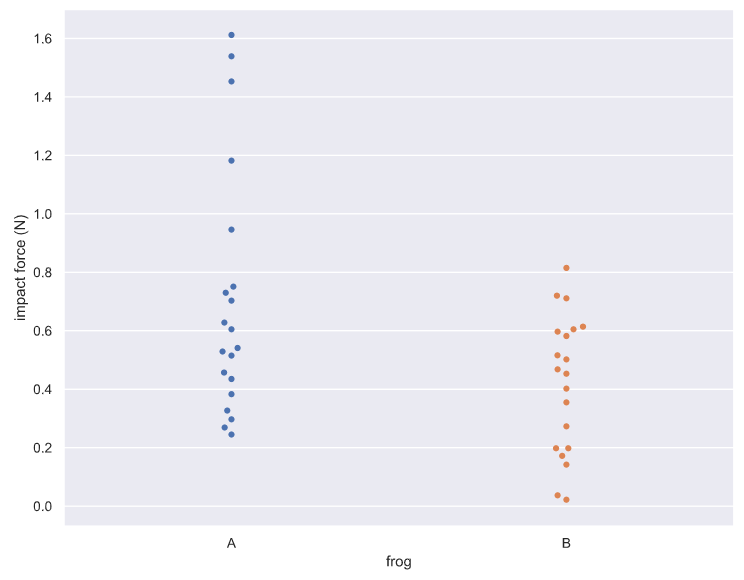
The alpha value is controller by the research and it relates to the confidence level. Depends on if you are running a one-tail test or a two-tail test, a = 1 – 10%, a = 1 – 10%/2

If the p value is extremely small or large and it falls into the area of reject, then we can reject the null hypothesis and approve the alternative hypothesis.

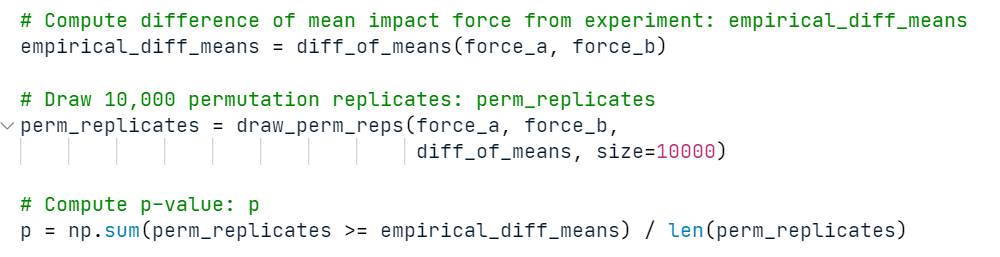
Example 1

Two frogs, one adult and one juvenile hit a wall with their tongue with a force. We are testing the hypothesis that the two frogs the same distribution of impact force.

First, use visual EDA



Then, draw permutations and calculate permutation replicates



Compute the p value, the percentage of permutation replicates that the difference of impact force is greater than the empirical difference

P = 0.0063.

The p value tells us that there is about a 0.6% chance that we would get the difference of means observed in the experiment if frogs were exactly the same.

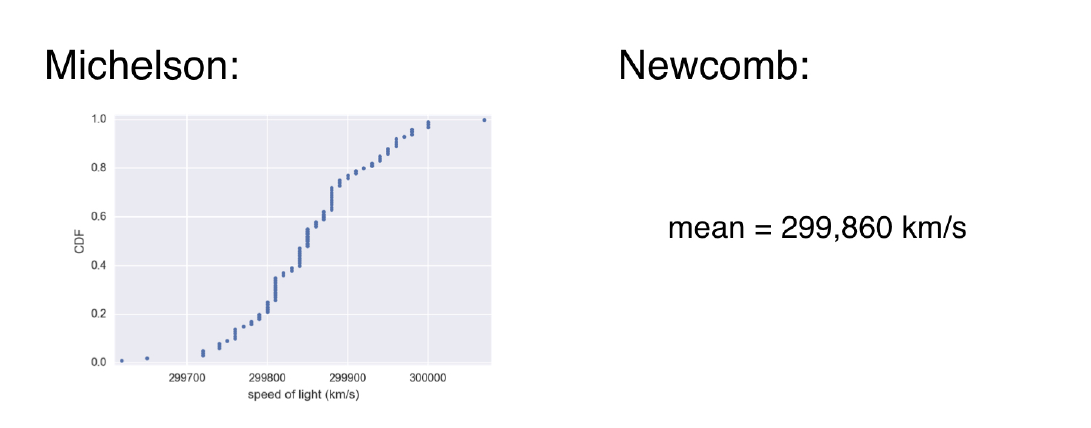
### Pipeline for hypothesis testing

* Clearly state the null hypothesis
* Define your test statistics (mean, property, difference etc / equal to, larger than)
* Generate many sets of simulated data assuming the null hypothesis is true
* Compute the test statistic for each simulated data set
* The p-value is the fraction of your simulated data sets for which the test statistic is at least as extreme as for the real data.

### Michelson and Newcomb

Michelson measure the speed of lights 100 times, Newcomb only have the mean.

Question: Could Michelson got the dataset he had from the experiments, if the true mean of the light is of Newcomb’s



**Null hypothesis**

The true mean speed of light in Michelson’s experiments was actually Newcomb’s reported value.

## Hypothesis test examples

### A/B Testing

## Putting it all together

# Introduction to Linear Modeling in Python

## Exploring Linear Trends

A linear **model** can be used to descript the distance travelled over time for a vehicle. The model can be used to predict values in between the sample data points (**interpolation**) or values outside the domain / range of the data points (**extrapolation**)

### Visualizing linear relationship

In matplotlib.pyplot use plot to plots scatter plots or a linear function

plt.plot(xs, ys, \*\*options)

### Quantifying Linear Relationships

#### Review of Single Variable Statistics

Mean, deviation, variance and standard deviation describes the central tendency and the spread the data.

**Deviation** measures the spread of the data, however, each spread has a sign and when summed up all together, they cancel one and other out to zero.

**Variance** is the square of the deviation, to overcome the problem of summing to zero.

However, variance is at a different unit of the original data, therefore, we take the square root of it and call it **standard deviation**.

std = sqrt(var)

#### Covariance and correlation

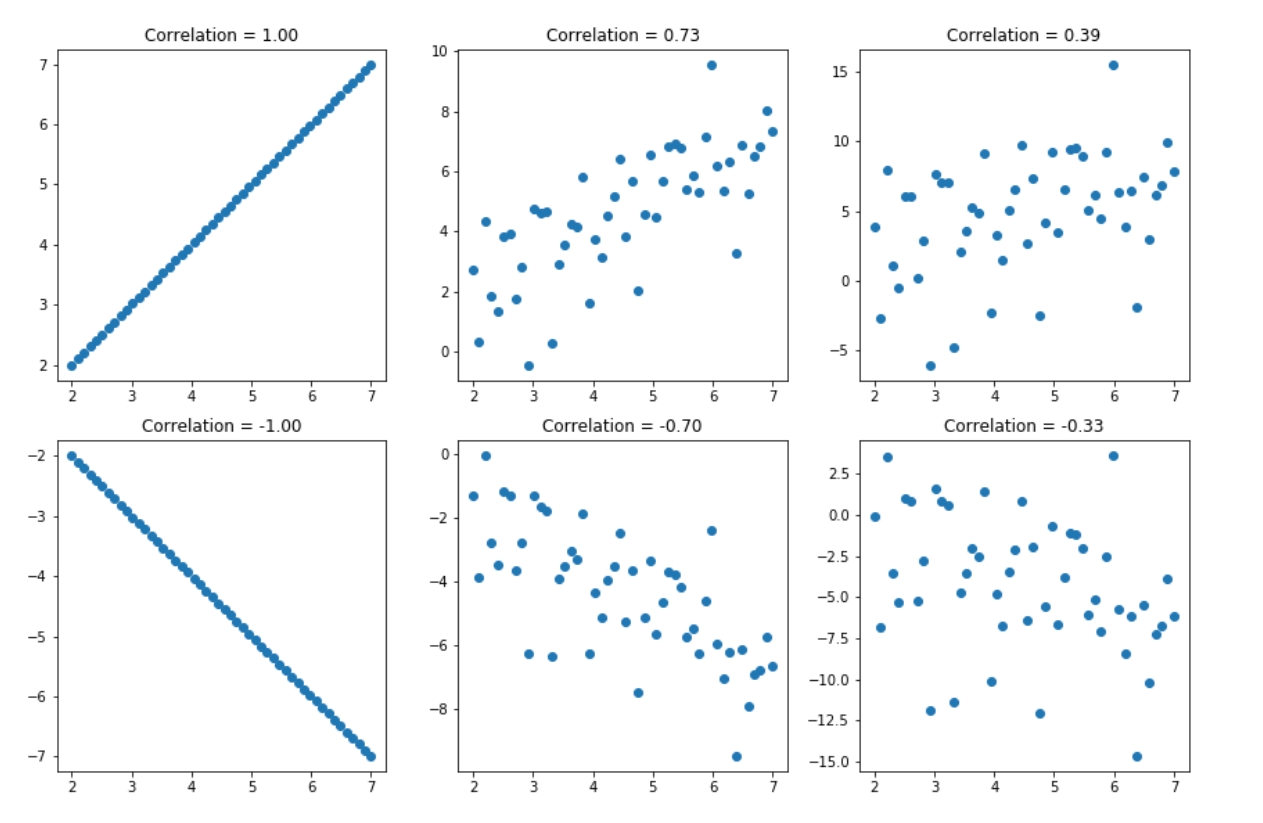
**Covariance** measure how two variables vary together. Covariance can have signs. If x and y varies together in the same direction, we will get larger product values.

**Correlation** is covariance normalized by the standard deviation of x and y. The normalization removes the weight bias of certain variable.

**Normalization**: rescales the values into range of [0, 1]

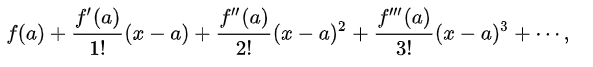
**Standardization** rescales the data to have a mean of 0 and standard deviation of 1 (unit variance)

Correlation’s magnitude vs direction

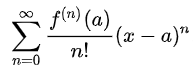


## Build Linear Models

A talyor series



In a more compact sigma notation, this can be written as



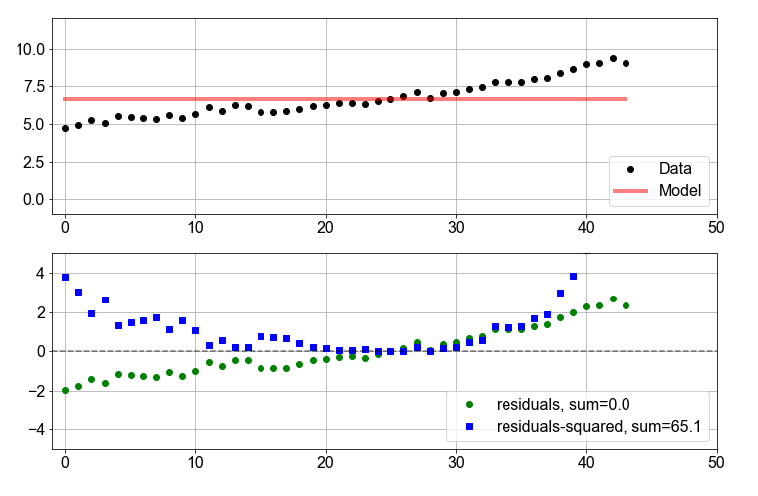
A linear model is a Taylor series with two degrees,

y = a0 + a1 \* x

where as a0 is the intercept and a1 is the slope.

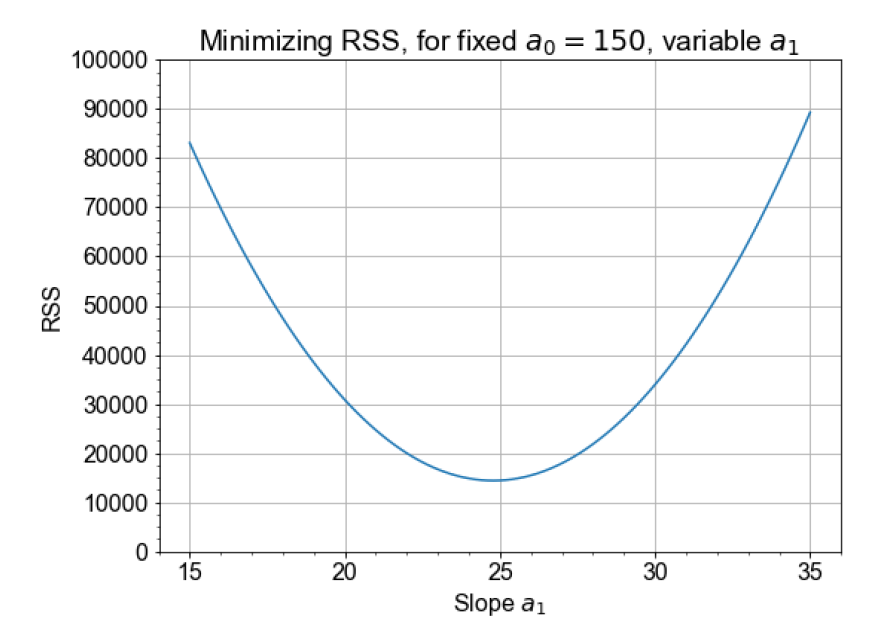
### Residuals

In the graph, the green dots are the residuals. There are both negative and positive values and they could cancel each other out. The blue dots are the squared residuals, they penalize values that are far from the mean.



**RSS**, **R**esidual **S**um of **S**quares is therefore calculated.

For a linear model, we can set a0 to a fixed value and plot possible values of RSS for a1 in a certain range



And we pick the minimum which gives us the least RSS.

### Least-Squares Optimization

Setting RSS slope = 0 and some calculus, yields:

a1 = covariance (x, y) / variance(x)

a0 = mean(y) – a1 \* mean(x)

## Making Model Predictions

### Modeling Real Data

ScikitLearn as a LinearRegression class with yields optimal parameters of a linear model. It taks two numpy arrays.

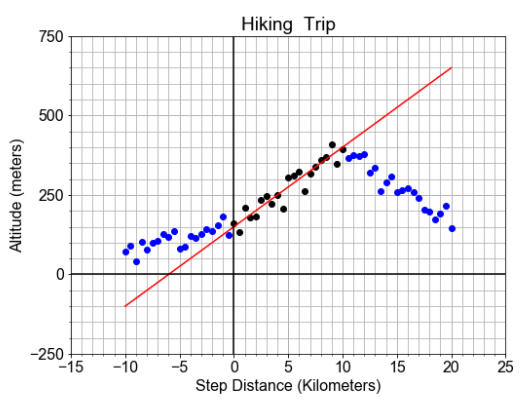
StatsModel has an OLS (ordinary least square algorithm) for linear model prediction, it accepts a pandas array. It gives error on the estimated parameters.

### The Limits of Prediction

Limits in interpolation: use SPX500 monthly price to build a linear model, but use the model to predict daily price.

Limits in extrapolating. Understand the problem and apply the model only to a reasonable range.

#### Domain of Validity



In the above picture, the domain of validity is in the range of [0, 10]. The linear model best captures the trend of the black dots. Outside the domain of validity, the linear model can no longer portrait the blue dots.

### Goodness-of-Fit

#### Three different R’s

**RSS** is used to help you find the optimal values for model parameters, and thus, the best model. But even the best model will still have non-zero residuals, so how “good” is the best model?

There are two common ways to quantify the goodness-of-fit for a linear model: **RMSE** and **R-squared**.

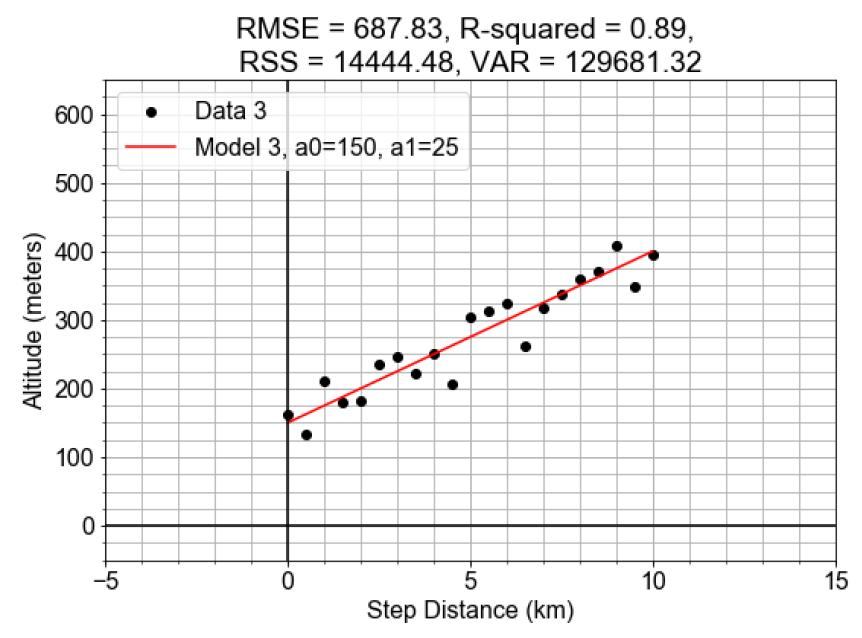
#### RMSE

RMSE is the root mean squared of the residuals. It tells us how much the model deviates from the data.

#### R-squared

R-squared (or the coefficient of determination) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable.

measures how much variation in the data is due to the linear trend.

The two models have the same value of RSS and RMSE. As they are calculated by

residuals = y\_model – y\_data

RSS = np.sum( np.square( residuals ))

RMSE = np.sqrt ( np.mean ( np.square (residuals) ) ) )

As best parameters are found to minimize the residuals for the two linear models, the RSS are the same.

deviations = np.mean(y\_data) – y\_data

var = np.sum(np.square(deviations))

r\_squared = 1 – (RSS/VAR)

deviations on a flat line will be much smaller than the deviation of a steep line. For a steep line (a model with a strong linear trend), the var gets larger, and the r\_squared gets smaller.

### Standard Error

You have computed quantitative measures of variation and "goodness" of the model \*predictions\*, but what about the variation or errors in the model \*parameters\*? How accurate are the model PARAMETERS, are there variations in those parameters, and how much of the variation is due to deterministic trends versus inherent randomness? In this lesson, instead of using a single value like RMSE that summarizes the entire model prediction, we will compute the standard error of each of the model parameters separately.

## Estimating Model Parameters

### Inferential Statistics Concepts

Previously, we found the single best value of each model parameter and used them to build a model. In this Chapter, we’ll treat a model parameter, like slope, not a single value but as a “distribution” of values, whose mean gives our “best” value.

### Model Estimation and Likelihood

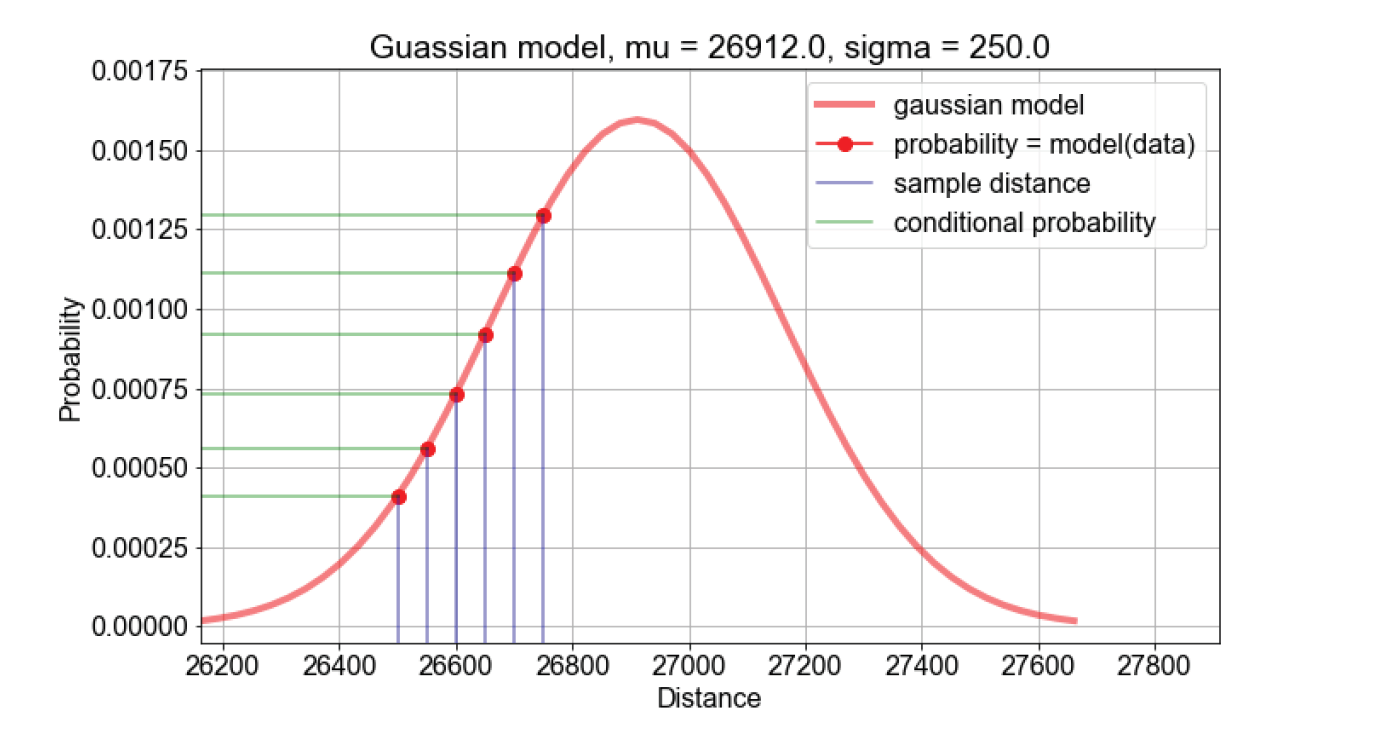
We use estimation to build models of population distribution from sample statistics.

A condition probability is stated as a question: what is the probability that A occurs, “given the condition” that B has already occurred. In the context of data and models, there is a naming convention for conditionals.

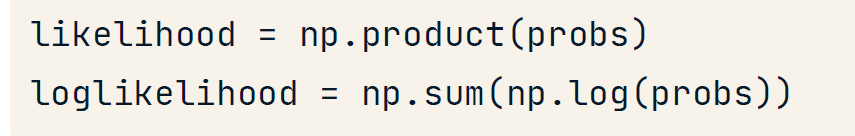
If the **model is given**, what is the **probability** that it output any particular data point.

If the **data is given**, what is the **likelihood** that a candidate model could output the particular data we have.

If we have two candidate models, we would like to choose the one that has a greater likelihood to output a given data.



For a certain value of mu and sigma, calculate the probability of all the sample data points. Take the products of all the probabilities, and take the log to compute the loglikelihood.



Compute log likelihoods for a range of estimations, find the best guess.

### Model Uncertainty and Sample Distributions

Previously, to estimate a model parameter, we assumed a shaped of the parameter distribution. Least-squares assumes a gaussian; maximum likelihood estimation requires us to chose a shape, so we chose gaussian.

But there are situations where the distribution shape is unknown.

Recall how the shape of the sampled temperature data resembled the population shape? What if we used the sample as the model of the population? If we compute the mean of the single sample, it gives us a guess, but no knowledge of the uncertainty in this guess. If we resample the samples, it gives a distribution sample statistic. We can use this to make predictions.

### Model Errors and Randomness

We’ve seen linear model parameters as distributions, spread about some central peak. Now we’ll relate the parameter distribution to the “standard error” of linear model parameters, and check whether our parameter estimates are effected by randomness.

#### Type of Errors

* Measurement error
* Sampling Bias
* Randomness

#### Null Hypothesis

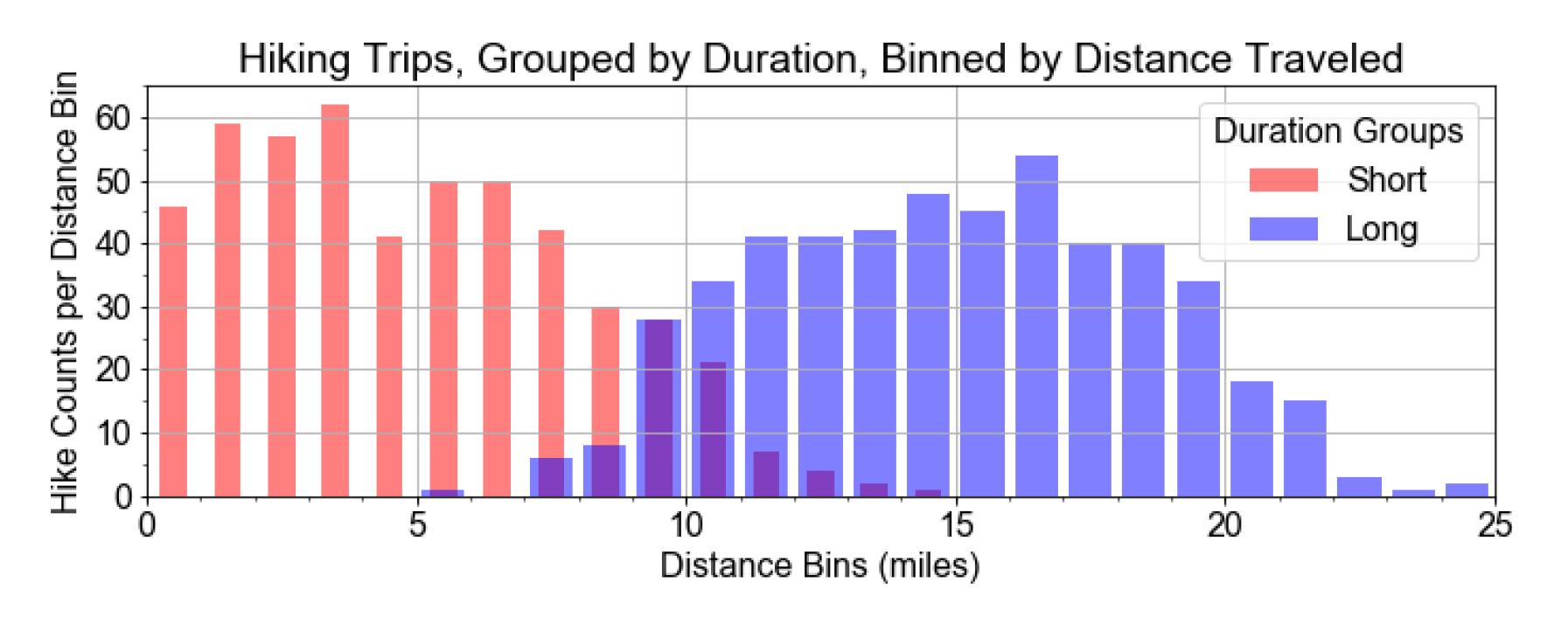
We restate the question:

“Is our effect due to a relationship or due to random chance?”

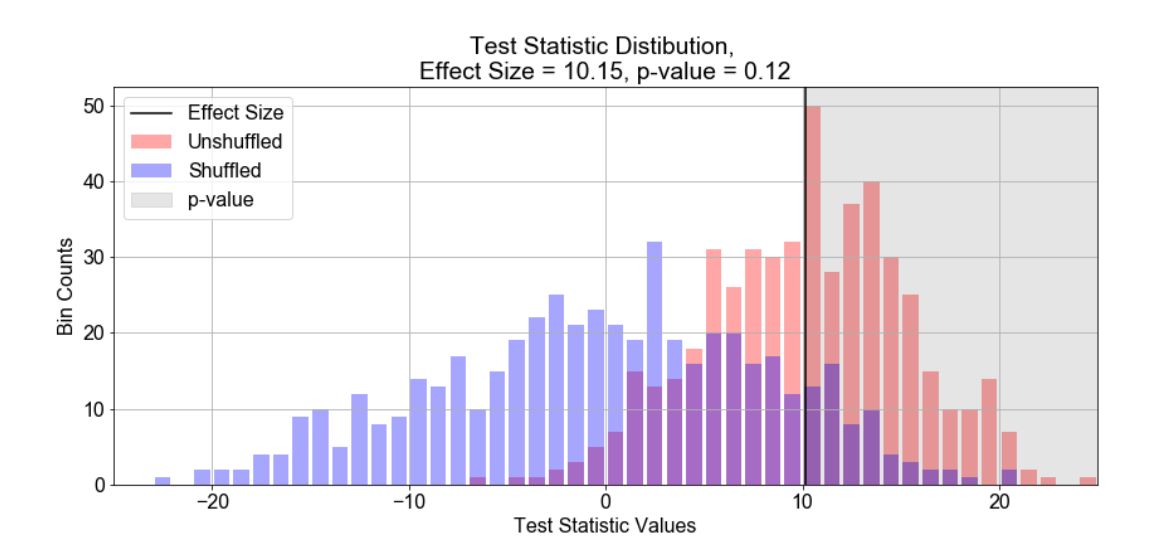
“Does the ordering or grouping of the data cause an effect larger than what could be produced by randomly shuffled data?”

#### Testing null hypothesis

We break the samples into two groups, short and long distance. The difference between those groups, are 10. We call it the **effect size** = 10



We shuffled the data and see the percentage of observations in the shuffled data that exceed the effect size. We call the percentage the p-value.



From the graph, we can see that it is 12 percent chance to get a speed of 10 or more just from random chance. If it is 0.01% chance to get a speed of 10 or more just from random chance, then we know that it is quite certain that the average speed is 10. Here, 12% is kind of in the middle.

# Statistical Simulation in Python

## Basics of randomness & simulation

Simulation Steps

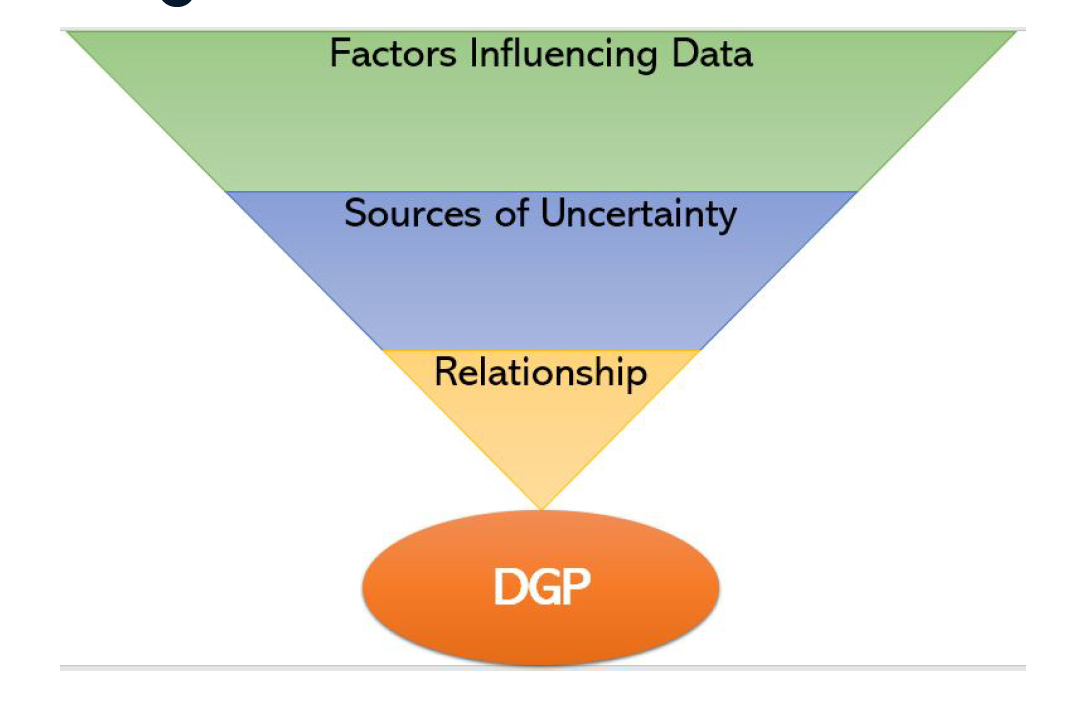
1. Define possible outcomes for random variables
2. Assign probabilities
3. Define relationships between random variables
4. Get multiple outcomes by repeated random sampling
5. Analyze sample outcomes

## Probability & data generation process

### Steps for Estimating Probability:

1. Construct sample space or population
2. Determine how to simulate one outcome
3. Determine rule for success
4. Sample repeatedly and count successes.
5. Calculate frequency of successes as an estimate of probability.

### Data Generating Process



e.g.

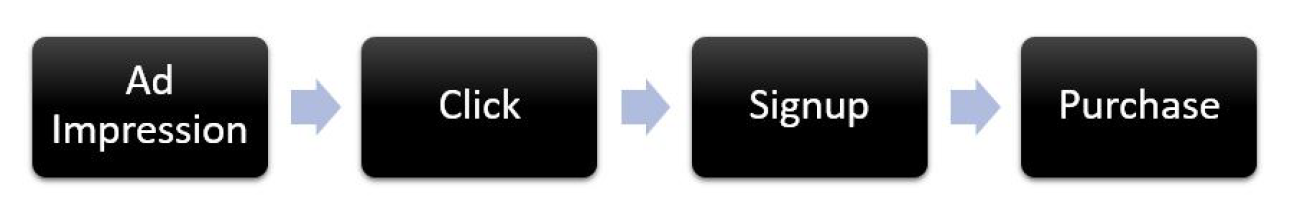
Factors influencing data: How many steps you take on a day

Source of uncertainty: Mood and motivation to go to the gym, 40% of the time

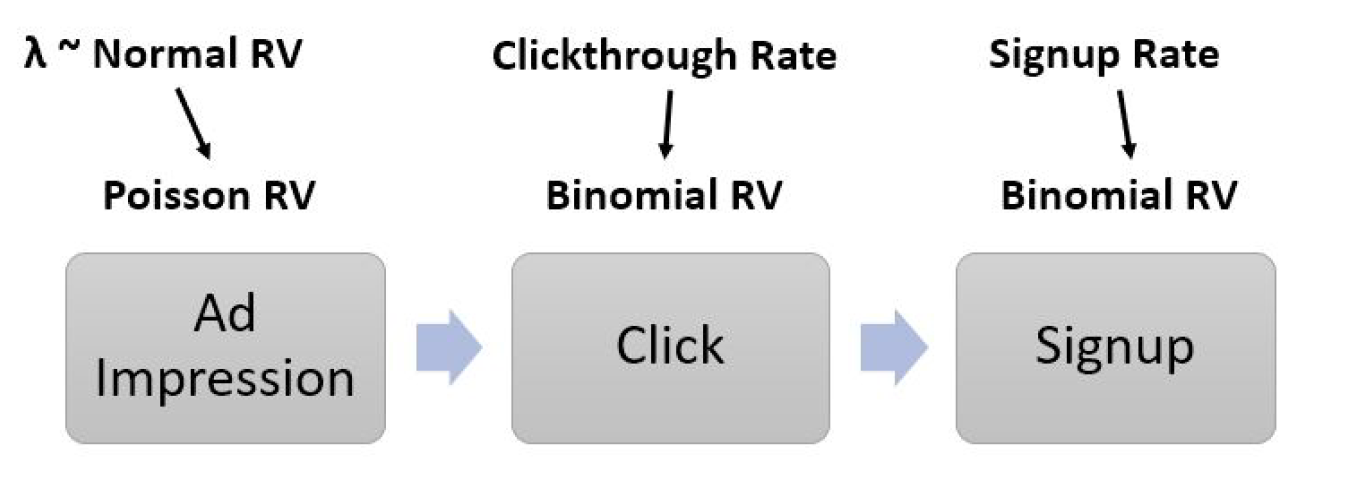
Relationship: If steps > 10000, 80% chance lose one pound, otherwise 20% chance gain one pound.

### eCommerce Ad Simulation

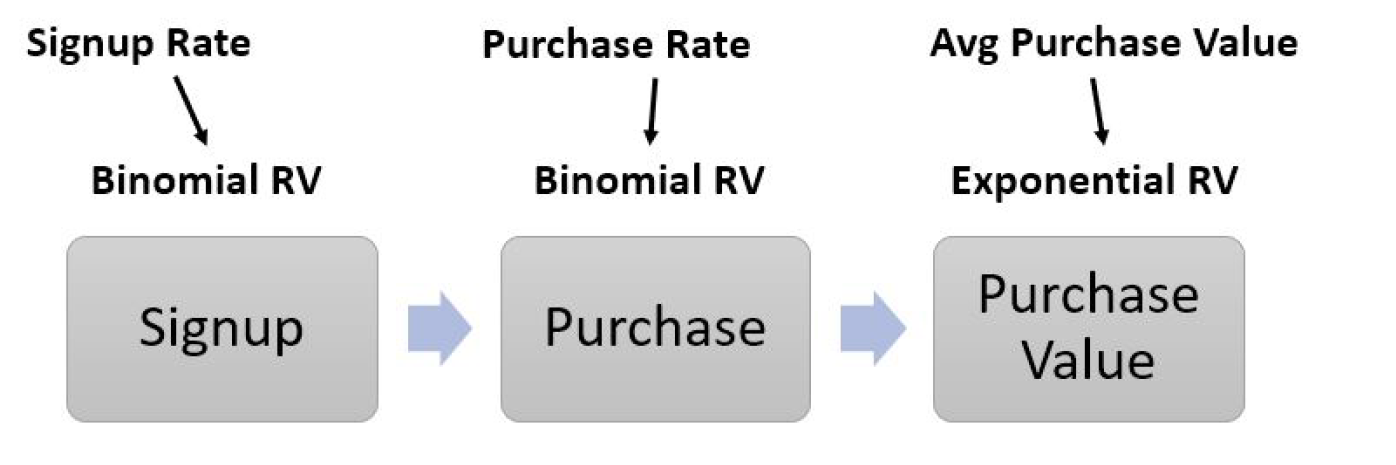
A general eCommerce flow



We model the first three steps using Poisson random variable and binomial random variables.



And we model the purchase flows using binomial and exponential random variables



## Resampling methods

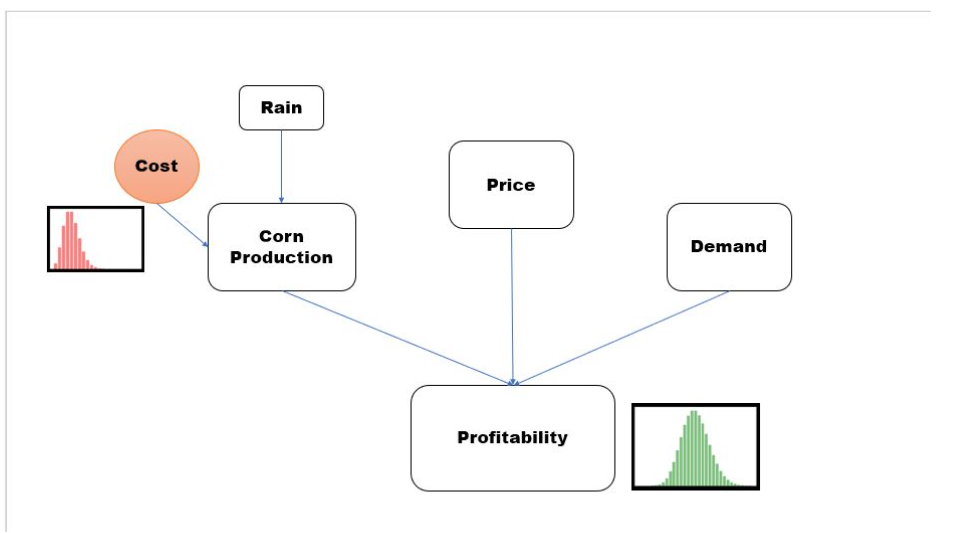
Different resampling methods

* Bootstrap resampling
* Jackknife resampling
* Permutation

## Advanced Applications of Simulation

### Simulation for Business Planning

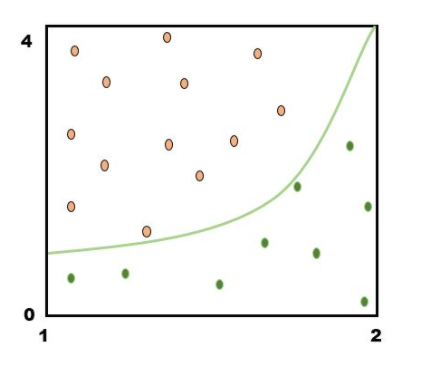
Simulation for Corn production



### Monte Carlo Integration

Steps

* Calculate overall area
* Randomly sample points in the area
* Multiply the fraction of the points below the curve by overall area.



### Simulation for Power Analysis

### Applications in Finance

# Supervised Learning with scikit-learn

## What is Machine Learning

### Supervised learning

Uses labeled data. Train the model with labeled dataset. Then predict whether an email is a spam or not.

### Unsupervised learning:

Unsupervised learning uses unlabeled data. It uncovers patterns from unlabeled data. For example, grouping customers into distinct categories (**clustering**, without having any presumptions of the groups)

### Reinforcement learning

Software agents interacts with an environment.

* Learn how to optimize their behavior
* Given a system of rewards and punishments.

## Classification

In supervised learning, datasets can be divided into features and target variables

The aim is to predict the target variable, given features.

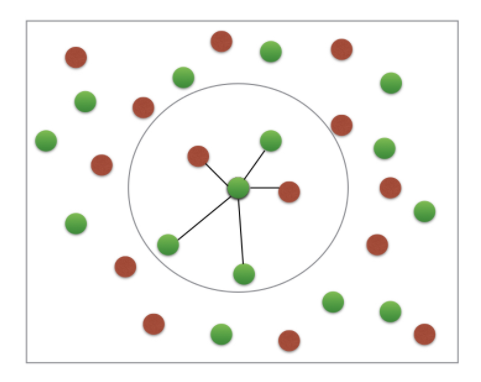
* Classification: target variable consists of categories
* Regression: target variable is continuous

### k-Nearest Neighbors

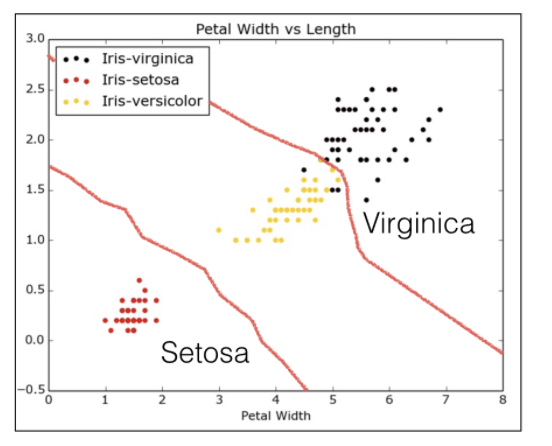
Basic idea: predict the label of a data point by

* Looking at the k closest labeled data points
* Taking a majority vote

K-Nearest neighbors in 2-dimensions, if K = 5, the point in the center is classified as green because 3 out of 5 nodes are green.

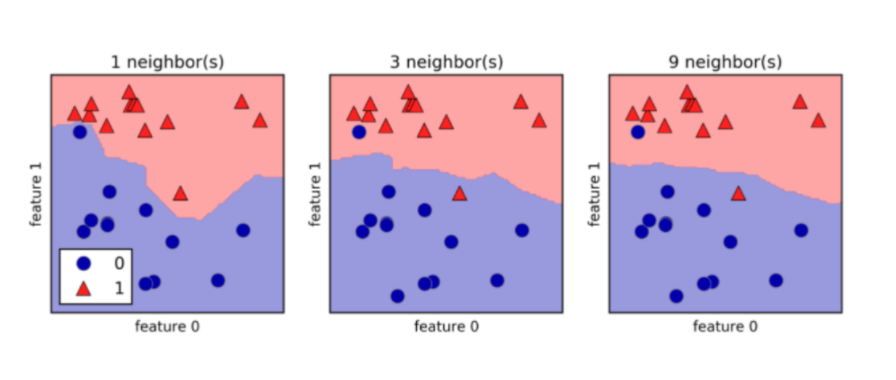


When fitting a KNN model, decision boundaries are created

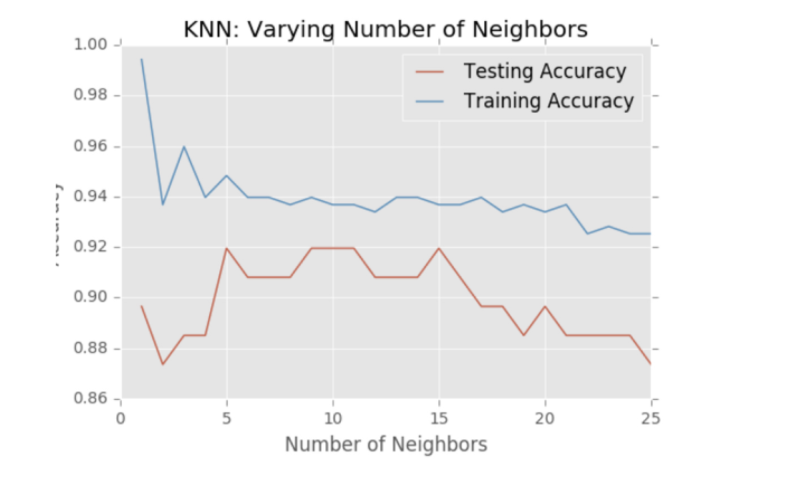


### Measuring performance

For categorization, we measure the performance in the percentage of correctly predicted test data. We split the dataset into training dataset and test dataset.



When k increases, the decision boundary get smoothers and less curvy. It is considered as a less complex model. Complex models run the risk of being sensitive to noise in the specific data you have, rather than reflecting general trends in the data. This is known as **overfitting**.



If we increase K even more, it performs less well in both training and test data set. This is called **underfitting**.

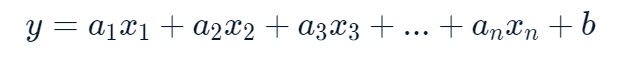
## Regression

### Regression mechanics

* y = a \* x + b
* y is the target
* x is a single feature
* a and b are parameters of the model
* How do we choose a and b?
* Define an error function and minimize the error

Ordinary least squares (**OLS**) minimize the sum of the squares of the residuals.

### Linear regression in higher dimensions

  
Cross validation splits data into n folds, runs regression n times, each time, split fold n as test data and the rest of the fold as training data.

### Ridge regression

Loss function = OLS loss function +

It penalizes models with large coefficients (both large positive and large negative)

Alpha is the parameter we need to choose. It is a hyper parameter, similar to picking k in kNN

Alpha controls model complexity

* Alpha = 0, we get back to OLS, (can lead to overfitting)
* Alpha is high, can lead to underfitting

### Lasso regression

Loss function = OLS loss function +

It can be used to select important features of a dataset.

Shrinks the coefficient of less important features to exactly 0,

Features that are not shrunk to zero are selected.

## Fine-tuning your model

### How good is your model?

Confusion matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Predicted | |
| Spam | Real |
| Actual | Spam | True Positive | False Negative |
| Real | False Positive | True Negative |

Accuracy = (tp + tn) / (tp + tn + fp + fn)

**Precision** = tp / (tp + fp)

Precision is also called positive predictive value or PPV. High precision means that our classifier had a low false positive rate, that is, not many real emails were predicted as being spam.

**Recall** = tp / (tp + fn)

Recall is also called sensitivity, hit rate or true positive rate. High recall means that our classifier predicted most positive or spam email correctly.

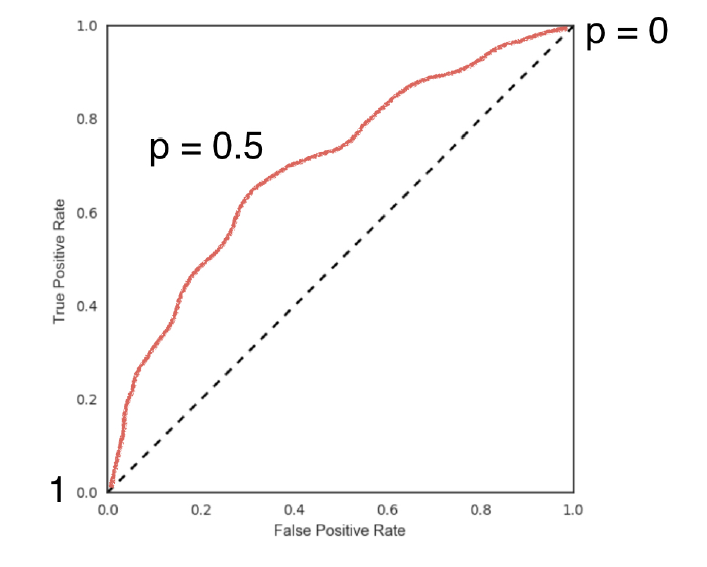
F1-score = 2 \* (precision \* recall ) / (precision + recall)

### Logistic Regression and the ROC curve

Logistic regression is a classification algorithm. Given one feature variable, it will output a probability p,

If P> 0.5, label as 1, If P < 0.5, label as 0.

The probability p is a threshold set to 0.5 by default, we when set p to all possible values, we can plot out the receiver characteristics curve (**ROC**) curve.



#### Area under the ROC curve, AUC

The large the area under the ROC curve, the better the model is.

### Hyperparameter Tuning

Linear regression, choosing coefficients

Ridge/Lasso regression: choosing alpha

k-Nearest Neighbors: choosing n\_neighbors

Parameters like alpha and k are hyperparameters. Hyperparameters cannot be learned by fitting the model.

To choose the correct hyperparameter, we try a bunch of different hyperparameters values and choose the best performing one.

## Preprocessing and pipelines

Dealing with categorical values

Handling missing data, drop null values or impute with mean or mode

Centering and scaling

# Unsupervised Learning in Python

## Clustering for Dataset Exploration

Unsupervised learning finds patterns in data. Clustering and dimension reductions are two major applications in unsupervised learning.

### k-means

K-means finds a specified number of clusters in the samples.

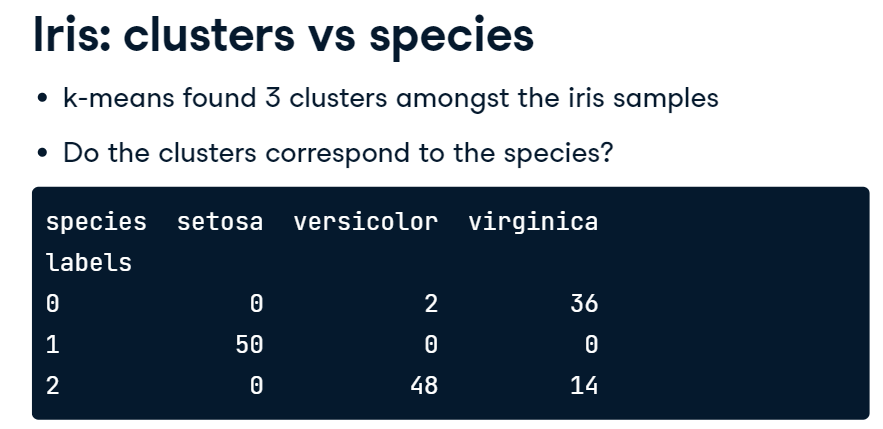
model.fit() method fits the model to the data, by locating and remembering the regions where the different clusters occur.

Model.predict() predicts the label of the data. It can predict either the training data set or the test data set.

### Evaluating a clustering

Method 1: Use cross tabulation to compare clustering with known labeled samples

Pandas.crosstab()

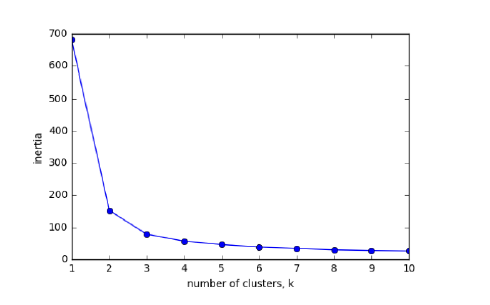


Method2: Calculate inertia

We need a good way to measure the quality of clustering that uses only the clusters and the samples themselves.

A good clustering has tight clusters, they are not too spread out.

Inertia measures how far samples are from the centroids.



More clustering means lower inertia, but we don’t want too many clusters. We choose an elbow point where the inertia begins to decrease more slowly.

### Transforming Features for better clusterings

The wine dataset has 17 features and some features have very different variances. Features have high variances will have a large impact on the k-mean algorithms. To give every feature an equal chance, they need to be transformed so that they have equal variance.

**StandardScaler** transforms each feature to have mean 0 and variance 1.

<https://www.youtube.com/watch?v=vd7duB8hHeM>

StandScaler has fit() / transform() methods

KMeans has fit() / predict() methods

MaxAbsScaler and Normalizer are another two scalers to be used in preprocessing steps.

Normalizer rescales each companies stock price, independently of the other.

## Visualization with Hierarchical Clustering and t-SNE

### dendrogram

Agglomerative clustering merges smaller clusters into bigger ones. Divisive clustering splits data sets into smaller groups.

In a dendrograms, you can choose a subgroup of clusters by specifying a height. Height is the distance between merging clusters. Greece and Cyprus has a height/distance of 6. Don’t merge clusters further apart than 15.

There are also different linkage types, single, average, complete and ward.



### t-SNE for 2-dimensional maps

t-SNE stands for t-distributed stochastic neighbor embedding. It maps samples from a higher dimensional space to 2D or 3D space.

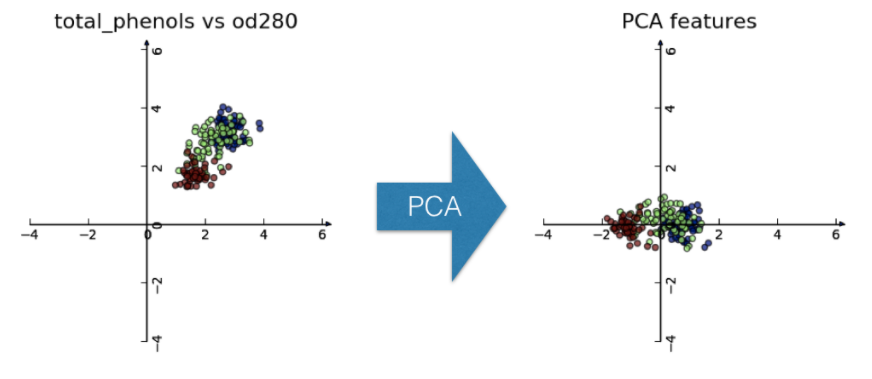
* t-SNE only has fit\_transform() method. It does not have separate fit(), transform() method.
* You need to choose a learning rate for t-SNE, try a couple of values between 50 and 200.
* t-SNE features are different every time.

## Decorrelating your Data and Dimension Reduction

Dimension reduction finds patterns in data, and uses these patterns to re-express it in a compressed form. It reduces a dataset to its “bare bones”, discarding noisy features that cause big problems for supervised learning tasks such as regression and classification.

### Principal Component Analysis (PCA)

PCA performs dimension reduction in two steps. First, it de-correlates the data. Then it reduces dimensions.



It rotates data samples to be aligned with axes and shifts data samples so they have mean of 0.

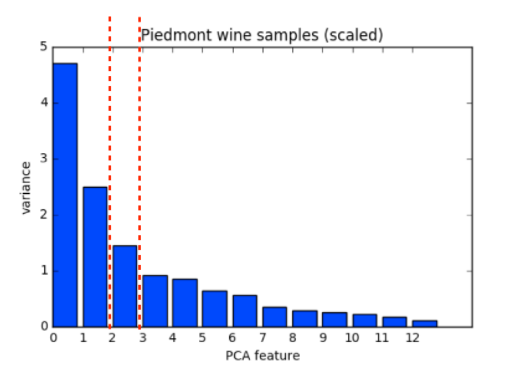
Once the dataset has been fitted and transformed, you can use model.components\_ to retrieve the principal components. Principal components are the directions in which the data varies the most. PCA aligns the principal components with the axes.

### Intrinsic dimension

A data set of flight path have two features: longitude and latitude. It appears to be 2-dimensional. But it can be approximated using one feature: displacement along flight path. We can say that the dataset is intrinsically 1-dimensional.

PCA can be used to find the number of intrinsic features in a data set. PCA only removes the correlation, but it keeps the variances of the features. Intrinsic dimension is number of PCA features with significant variance.

However, we can argue that the Piedmont wine samples have a intrinsic dimension of 2, 3 or even more.



### Dimension reduction with PCA

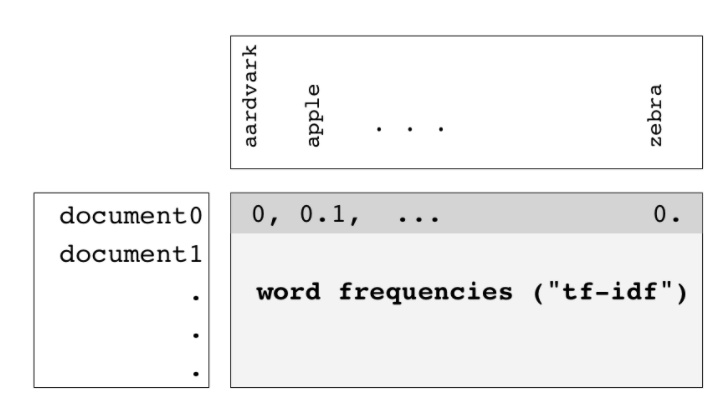
PCA features are in decreasing order of variance.

It assumes the low variance features are “noise” and high variance features are “informative”

### Word Frequency arrays

Rows represents documents, columns represent words

TF-IDF, term frequency-inverse document frequency



Most of the entries in the matrix have value zero. As not every word appears in one particular document. These sparse array can be represented by scipy.sparse.csr\_matrix. Csr\_matrix only remembers the non-zero entries in the array.

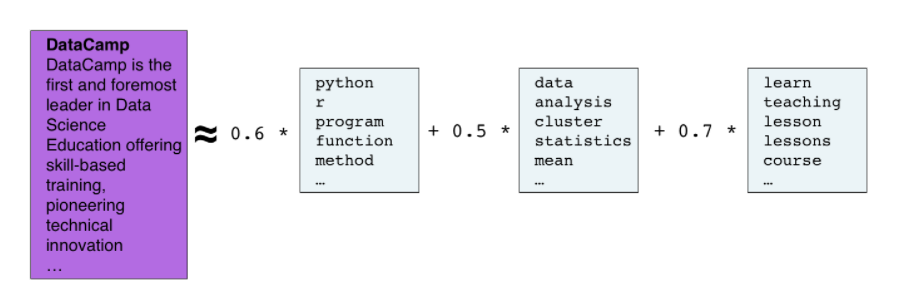
PCA doesn’t support csr\_matrix. Use TruncatedSVD instead.

## Discovering Interpretable Features

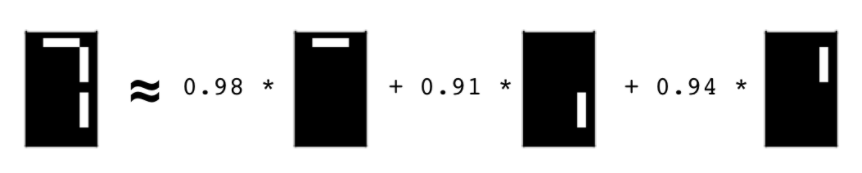
### Non-negative matrix factorization (NMF)

NMF is another dimension reduction technique. In contrast to PCA, however, NMF models are interpretable. NMF requires sample features to be “non-negative”.

NMF expresses documents as combinations of topics



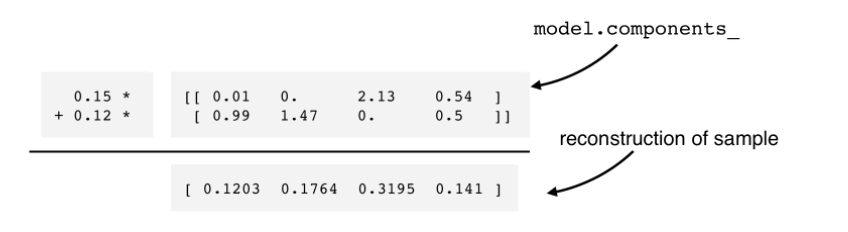
It expresses images as combination of patterns



#### Example usage of NMF

n\_components must be specified

nmf\_features \* model\_components\_ = reconstuted sample



### NMF learns interpretable parts

### Building recommender systems using NMF

# Linear Classifiers in Python

## Logistic Regression and SVM

Scikit learn offer three classifiers. LogisticRegression, LinearSVC and SVC.

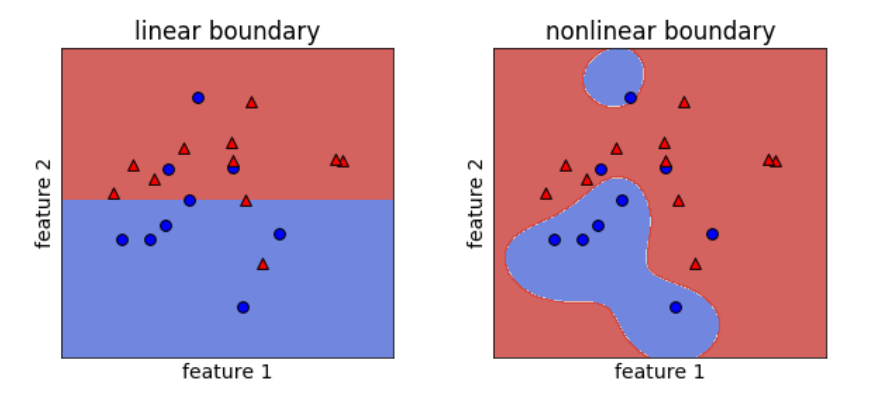
By choosing a hyperparameter for the classifier, we define the complexity of the model.

Underfitting: too simple model, low training accuracy.

Overfitting: too complex model, low test accuracy.

### Linear Decision Boundaries

A decision boundary tells us what class our classifier will predict for any value x. In the picture, the classifier predicts the blue class in the blue shaded area and predicts the red class in the red shaded area. The line dividing the two regions are called the decision boundary. The decision boundary is linear because it is a line. With 5 features, the space of possible x-values is 5-dimensional, a linear boundary would be a high-dimensional “linear hyperplane” cutting the space into two halves.



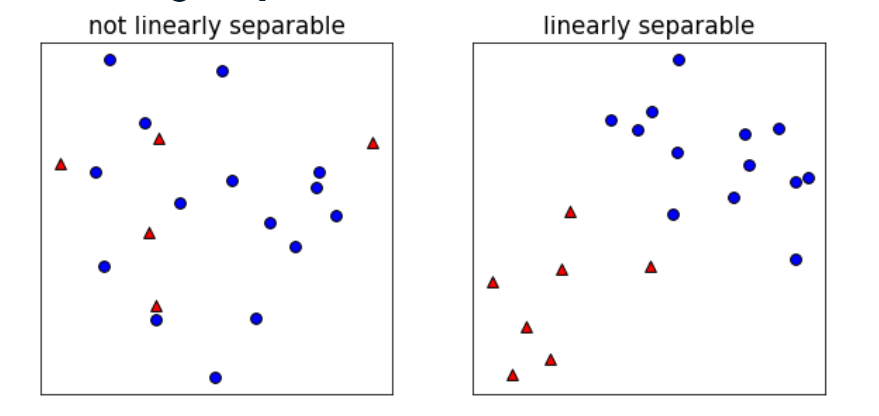
### Vocabulary:

**Classification**: learning to predict categories.

**Decision boundary**: the surfaces separating different predicted classes.

**Linear classifier**: a classifier that learns linear decision boundaries.

**Linearly separable**: a data set can be perfectly explained by a linear classifier.



## Loss Function

The dot product

np.sum(x\*y)

or x @ y

### Linear classifier prediction

Raw model output = coefficients \* features + intercepts

y = ax + b

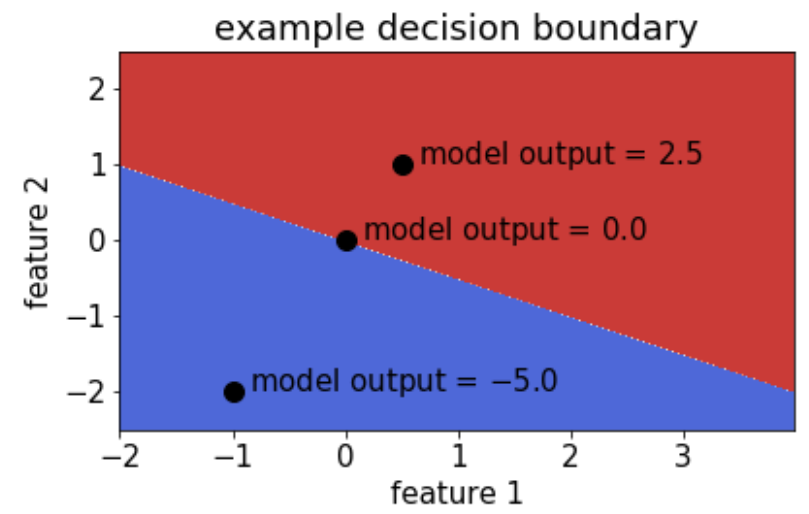
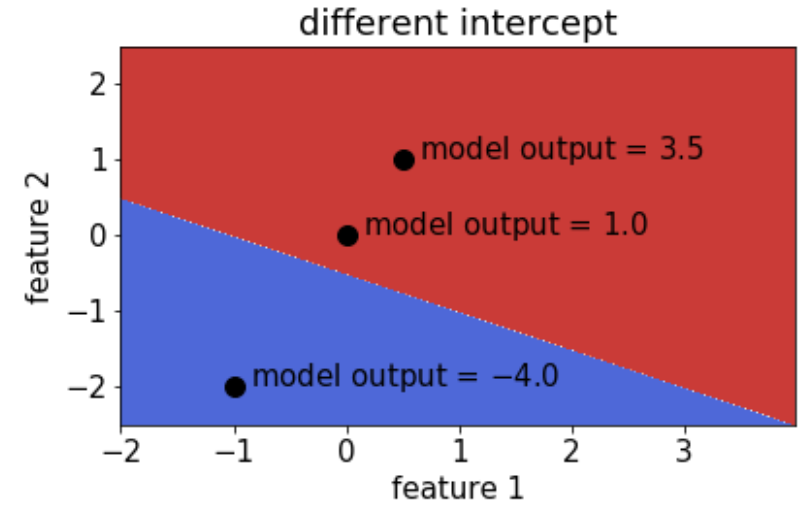
y = a1x1 + a2x2 + …+ anxn + b

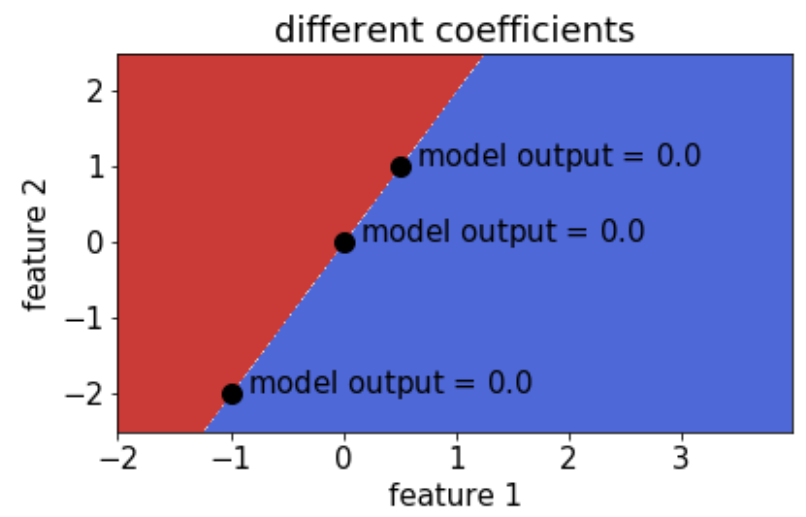
If y > 0, predict one class

If y < 0, predict another class

Logistic Regression and Linear SVM have the same predict function, but different fit function

Here is an example of how coefficients and intercepts can impact decision boundaries



### What is a loss function?

Ordinary Least Squares, OLS is a lost function in the LinearRegression classifier.

OLS is not appropriate for classification problems. A natural loss for classification problem is the number of errors.

It is called the **0-1 loss**. 0 for correct prediction and 1 for incorrect ones.

num\_error = np.sum(y != model.predict(x\_test))

### Scipy.Optimize.Minimize function

optimizeResult = minimize(fun, x0)

x0 is the initial guess

optimizeResult.x gives you the value of x that produces the smallest y.

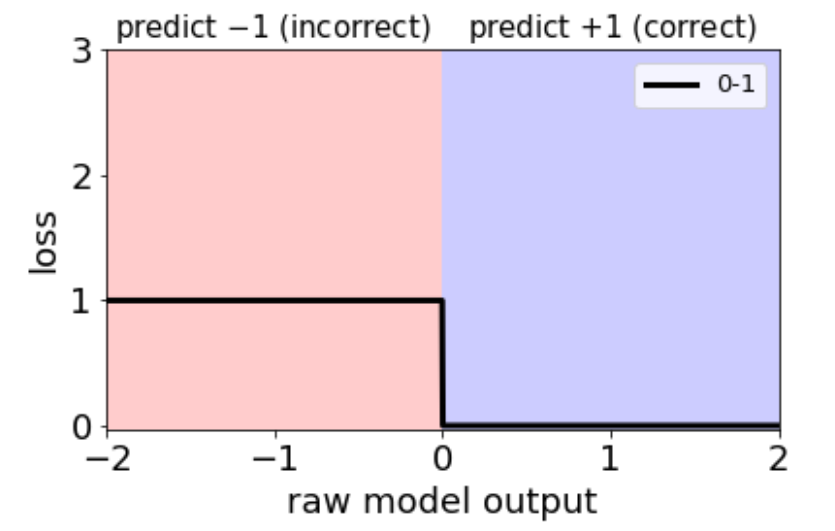
For example:

minimize(np.square, 0).x yields 0

minimize(np.square, 3).x yields 0.000018

### Loss function diagrams

#### 0-1 loss diagram

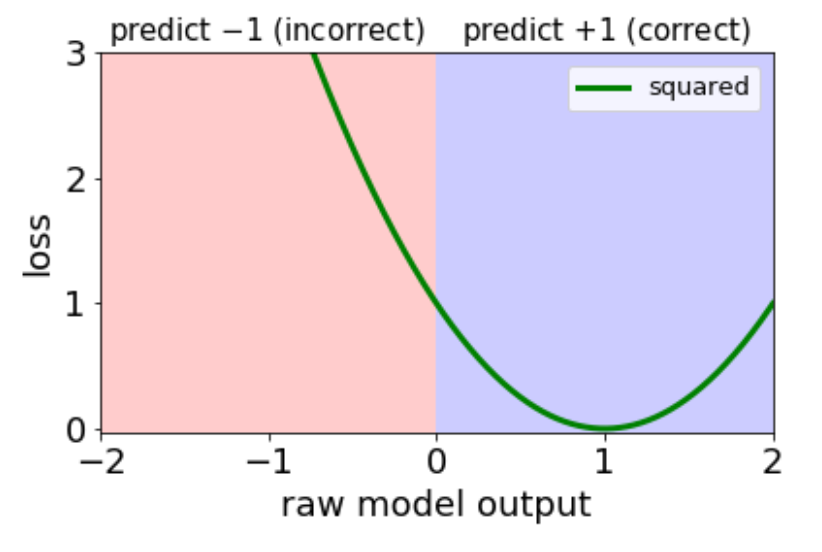


Since we predict using the sign of the raw model output, the plot is divided into two halves. In the left, we predict the -1 class, in the right, we predict the +1 class. For concreteness, let’s focus on a training example in class +1. Then the right half is the correct prediction and left half incorrect ones.

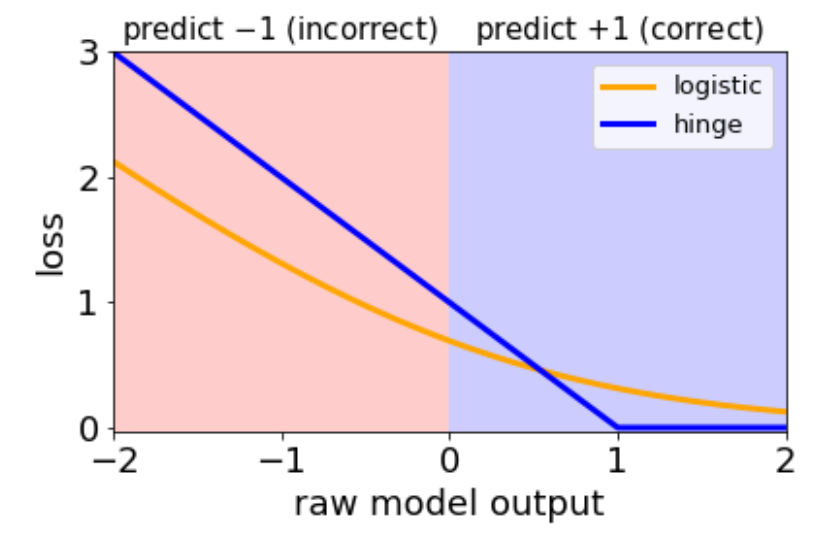
To bear in mind that this is the loss function for a particular training example. To get the whole loss, we need to sum up the contribution from all examples.

#### Linear regression loss diagram

Linear regression uses OLS as a loss function. This diagram make sense for linear regression, but it doesn’t make sense for a linear classifier (classification). Being really close to the true value doesn’t really matter, as long as we get the sign right, therefore, the right arm of the diagram, from range 1 to 2 are incorrect. The predictions are correct and we expect small values of the error.



#### Logistic loss and hinge loss diagram



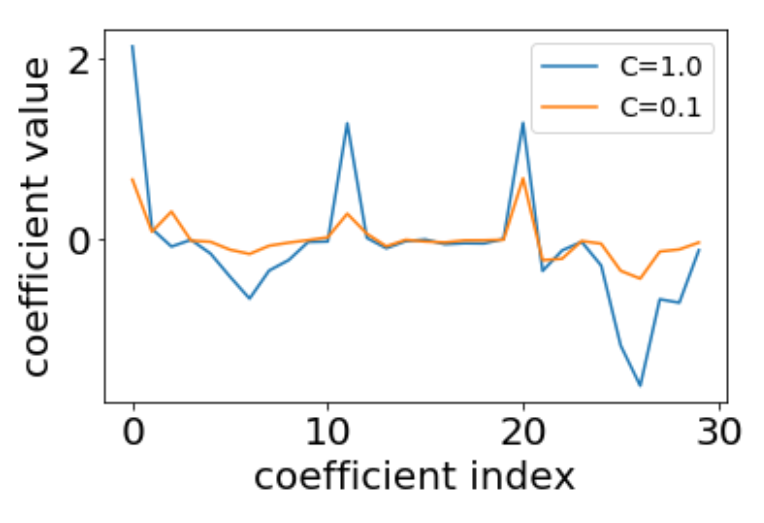
Logistic Regression uses logistic loss function

SVM uses hinge loss function

## Logistic Regression

#### Logistic Regression and Regularization

Regularization combats overfitting by making the model coefficients smaller. In Sklearn’s LogisticRegression classifier, C is the **inverse** of the regularization strength. The smaller the value is, the larger the regularization penalty.



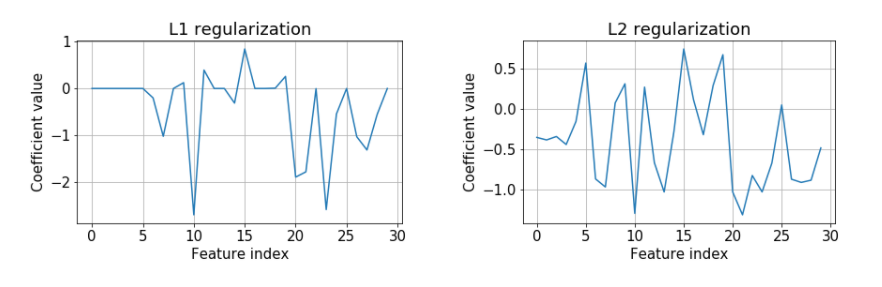
Strong regularization decreases training accuracy, it can potentially deal with unseen data better and produce higher test accuracy.

regularized loss = original loss + large coefficient penalty.

Large coefficient penalty distracts the algorithm from the goal of optimizing accuracy. The larger the regularization penalty, the more we deviate from our goal of maximizing training accuracy.

##### L1 vs L2 regularization

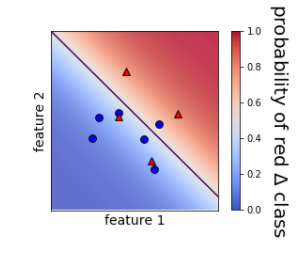
For linear regression, we use Lasso for L1 regularization and Ridge for L2 regularization. The general names for these concepts, outside linear regression, are L1 regularization and L2 regularization. L1 sets many coefficients to zero, thus performs feature selection. L2 shrinks all of the coefficients smaller, preventing overfitting. Before we perform fitting with regularization, we can do data preprocessing and scale the data first.



#### Logistic Regression and Probabilities

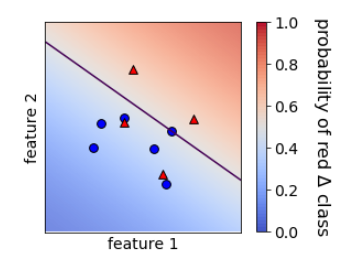
Without regularization. C = 10e8

model coefficients: [[1.55, 1.57]]



With regularization, C = 1

model coefficients are smaller: [[0.45, 0.64]] and probabilities of the two classes do not go all the way to 0.0 or 1.0

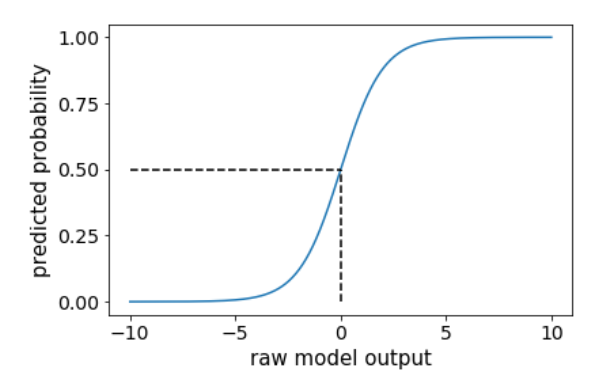


These figures also answer a question.

With two features, we had 2 coefficients even though you only really need one number to represent the slope of a line. We now have a reason for this, the **ratio** of the coefficients gives us the slope of the line, the **magnitude** of the coefficients gives us our confidence level.

How are the confidence computed?

The sigma function maps the raw model output to the predicted probability



#### Multi-class logistic regression

Approach1: Combining binary classifiers with one-vs-rest

lr0.fit(X, y == 0)

lr1.fit(X, y == 1)

lr2.fit(X, y == 2)

for one sample, choose the raw model output with the largest value of the decision function as the predicted class.

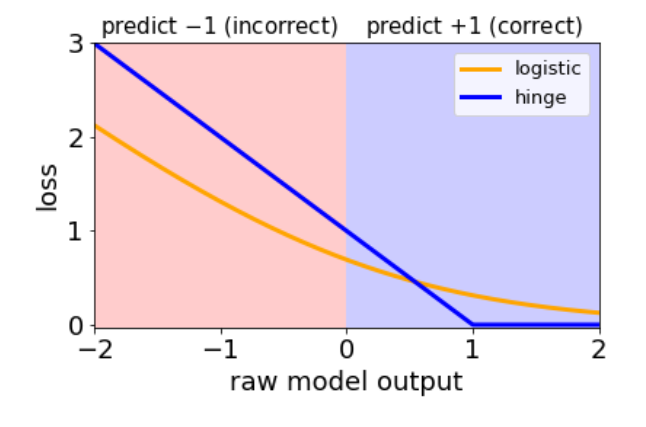
Approach 2: fit a single classifier for all classes, a.k.a. multinomial or softmax

## Support Vector Machines

#### What is SVM?

Support-vector machines are supervised learning models for classification and regression analysis. SVM maps training examples to points in space so as to maximize the width of the gap between the two categories.

SVM uses the hinge loss function and L2 regularization

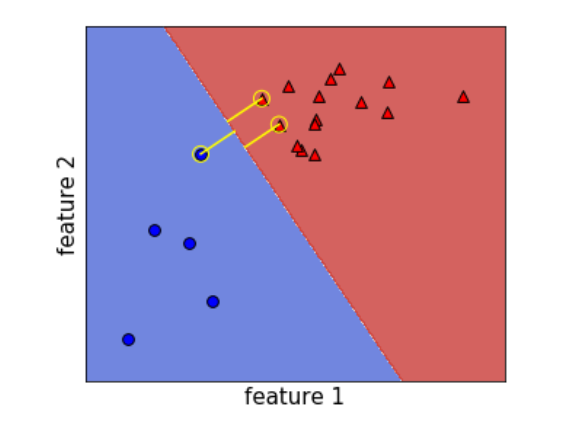


Support vectors can be defined as:

* a training example not in the flat part of the loss diagram
* an example that is incorrectly classified or close to the boundary. The regularization strength controls how close to the boundary.

If an example is not a support vector, removing it has no effect on the model.   
kernel SMV computes really fast. Its computation time is decided by the number of support vectors, instead of the total number of training samples.

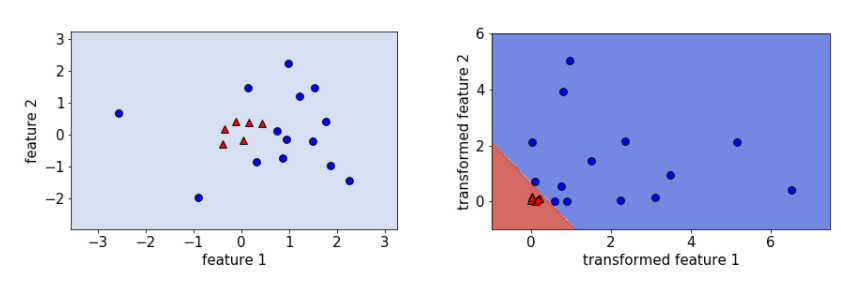
SVM maximized the “margin” for linearly separable datasets.



#### Kernel SVMs

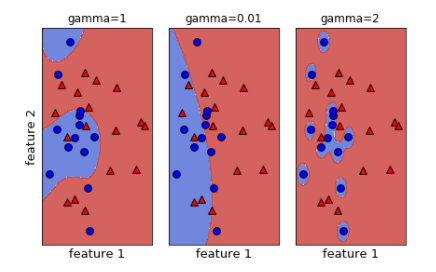
Given a dataset of two features, it is not linearly separable in the original space.

If we take square value of both of the features, the dots can be plotted in the figure in the right. And it becomes linearly linearly separable.

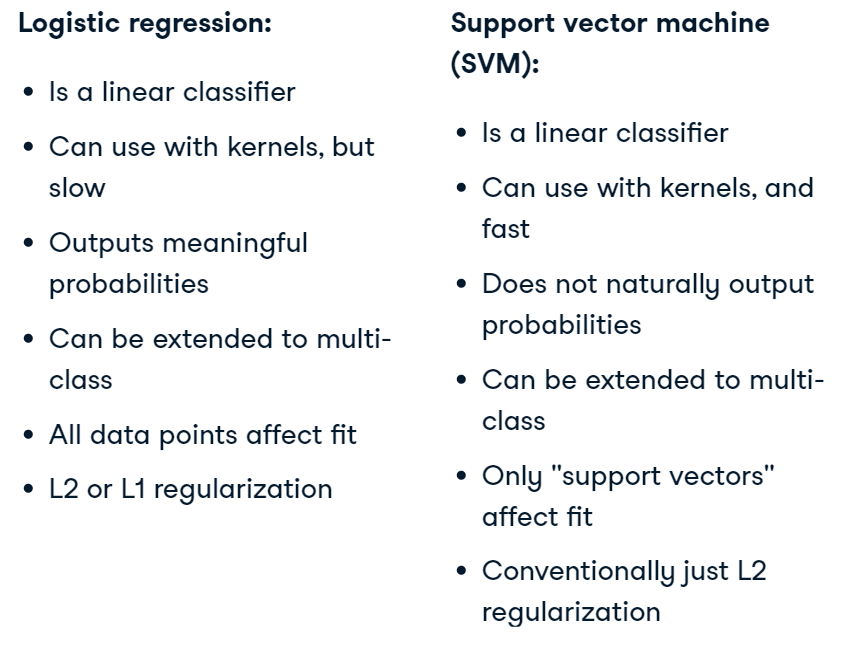


Kernels and kernel SVMs implement feature transformations in a computationally efficient way.

Sklearn’s SVC algorithm has gamma as an hyperparameter. It defines the smoothness of the decision boundaries. The larger the gamma, the less smooth the decision boundary

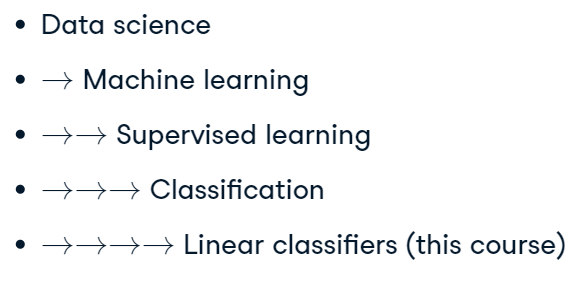


#### Comparing LogisticRegression and SVM



SGDClassifier scales well to large datasets

### Conclusion



# Case Study: School Budgeting with Machine Learning in Python

<https://github.com/datacamp/course-resources-ml-with-experts-budgets>

<https://github.com/drivendata/boxplots-for-education-1st-place>

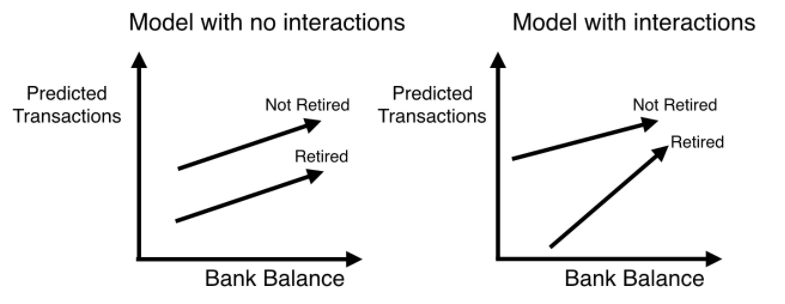
# Introduction to Deep Learning in Python

## Basic of Deep Learning and Neural Networks

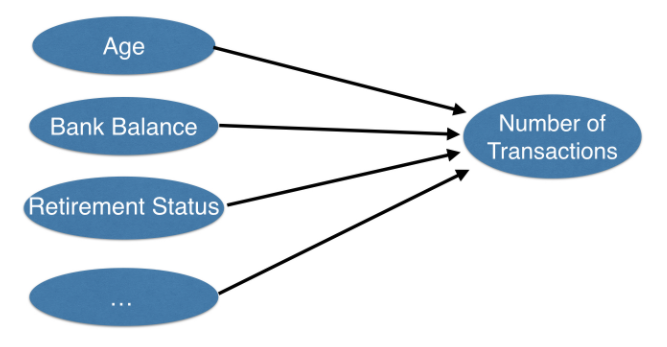
### Difference between Linear Model and Deep Learning Models

Linear models do not capture the interactions between features.

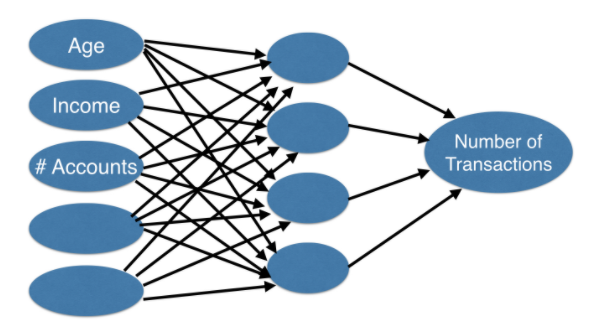
For example, the slope of the line represents the relationship between the number of transaction and bank balance. If we add one more feature into the equation, “retired”. we can see that whether a customer is retired or not does not have an effect on the slope of the life, but only the intercept.



To illustrate in another way, linear model can be represented like this,

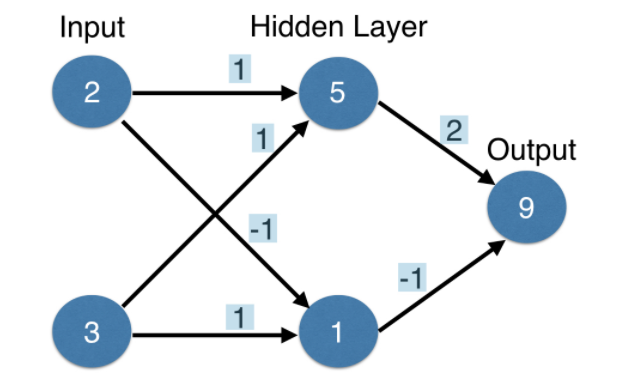


Deep learning models can be represented like these. An interaction was added for bank balance and retired status. Or many interactions are added between all those features in the hidden layer.

### Forward Propagation

We start by showing how neural networks use data to make predictions. This is called the forward propagation algorithm.



The network has an input layer with 2 features, 2 children and 3 accounts. A hidden layer with 2 nodes and an output layer with one node. We have weights connecting the nodes. The weights are updated when we train the neural network.

To make a prediction for the top node of the hidden layer, we take the dot product of the values of the input layer and the relevant weights.

### Activation Functions

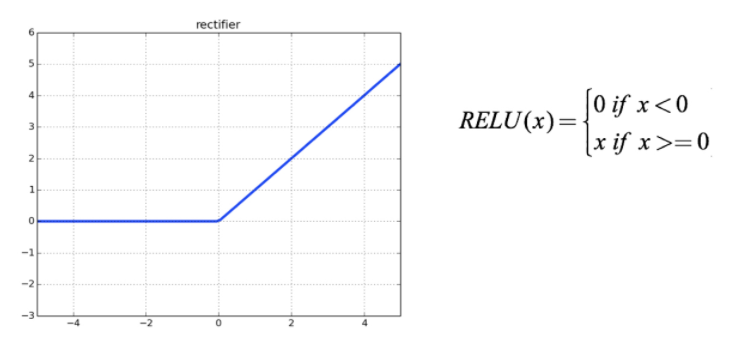
Calculating the input of the hidden layers using dot product is half of the story. For neural networks to achieve their maximum predictive power, we must apply an activation function.

An activation function allows the model to capture non-linearity. The function takes the input value of a node and generates an output value.

#### ReLU (Recitified Linear Activation)

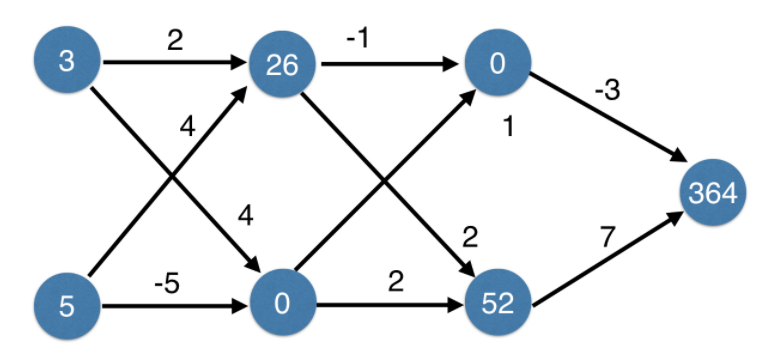
Relu is a powerful activation function. More readings here

<https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253>



### Deeper Networks

In practice, deep networks with dozens of layers and thousands of hidden nodes are used to solve various problems



Some characteristics of neural networks.

* Deep networks internally build representations of patterns in the data
* Partially replace the need for feature engineering
* Subsequent layers build increasingly sophisticated representations of raw data.

## Optimizing a Neural Network with Backward Propagation

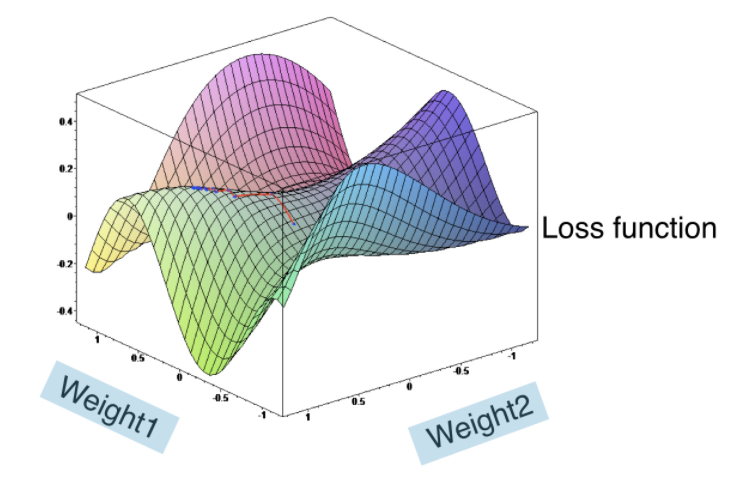
### The Need for Optimization

When the model predicts a value that differs from the target value, we can an error

error = predicted value – target value

We use a loss function to aggregated all the errors into a single measure of the model’s predictive performance.

Consider the graph of a loss function with 2 weights



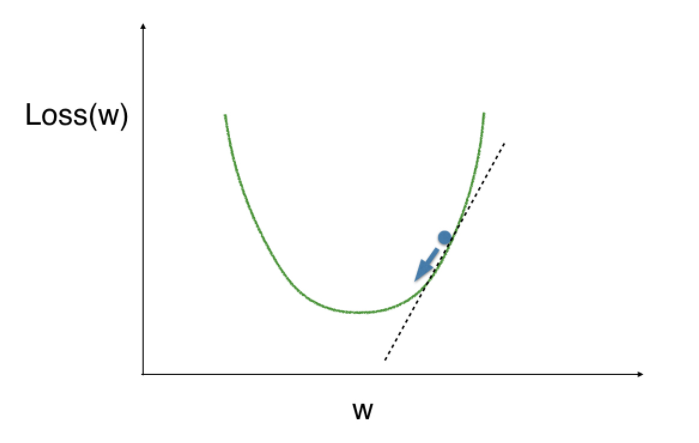
### Gradient Descent

We use a process called gradient descent to find the lowest point the get the smallest loss. We take a small step downhill, repeat until it is uphill in every direction.

Gradient Descent steps

* Start at a random point
* Until you are somewhere flat
  + Find the slope
  + Take a step downhill

For a loss function that takes only one weight



* At a random point, find the slope (the tangent line, or the derivative)
* The slope is positive, to go downhill, we subtract portion of the slope from w
* Repeat the process until we get to the minimum value.

### Gradient Descent Breakdown

* To calculate the slope for a weight, we need to multiply three components



* The slope of the loss function w.r.t value at the node we feed into

Slope of mean-squared error loss function w.r.t prediction

2 \* (predicted value – actual value) = 2 \* error

* The value of the node that feeds into our weight

value 3

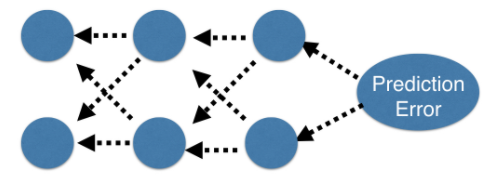
* The slope of the activation function w.r.t value we feed into

For identify function, ignore this component

* updated weight = weight – learning rate \* slope

### Backpropagation

Backpropagation allows gradient descent to update all weights in neural network



Backpropagation process

* Trying to estimate the slope of the loss function w.r.t each weight
* Always do forward propagation to calculate predications and errors
* Go back one layer at a time
* Gradients for weight is the product of:

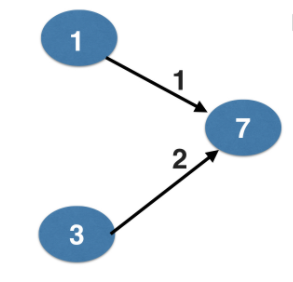
1. Node value feeding into that weight
2. Slope of the loss function w.r.t node it feeds into
3. Slope of activation function at the node it feeds into

For ReLU,   
slope = 0 if output = 0

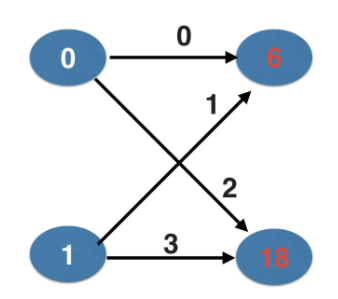
slope = 1 if output > 0

### Backpropagation in practice

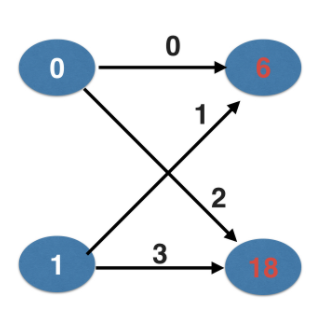
1. We use ReLU activation function, target = 4, prediction = 7, error = 3



1. Slope is 2 \* error, we get the slope as 6 and 18



1. Gradients for weight is the product of
   * Node value feeding into that weight
   * Slope of activation function for the node being fed into
   * Slope of loss function w.r.t output node
2. To calculate weight00, w01, w10, w11



|  |  |  |  |
| --- | --- | --- | --- |
| Node value feeding into the weight | Slope of ReLU | Slope of loss function | Product |
| 0 | 1 | 6 | 0 |
| 1 | 1 | 6 | 6 |
| 0 | 1 | 18 | 0 |
| 1 | 1 | 18 | 18 |

1. The product of step 4 is also called the gradient weight. Update the weight by

updated weight = weight – learning rate \* gradient

1. Repeat the process until we get to a flat part

### Stochastic gradient descent

* It is common to calculate slopes on only a subset of the data (a batch)
* Use a different batch to calculate the next update
* Start over from the beginning once all data is used
* Each time through the training data is called an epoch
* When slopes are calculated on one batch at a time:

Stochastic gradient descent

## Building Deep Learning models with Keras

To build a neural network in Keras, we use the following steps

* Specify Architecture
* Compile
* Fit
* Predict

### Specify architecture

We choose a **Sequential** model. Sequential models require that each layer has weights or connections only to the one layer coming directly after it in the network diagram. The alternative is to use Functional API to build any network graph that you desire.

We add Dense layers into the model. In a dense layer, all the nodes in the previous layer connect to the nodes in the current layer.

The first layer is the input layer, we specify the iput\_shape parameter as (n\_cols, ). N-cols is the number of features, the n\_rows are not specified so that we can have an arbitrary number of rows / observations here.

### Compile

The compile method sets the optimization method and the loss function.

The optimizer controls the learning rate. It affects how fast the model cand find the right weights or how good a set of weights it can find. Adam is a versatile optimizer.

For the loss function, mean squared error is a common choice for regression problems while categorical\_crossentropy is the choice for classification problems.

### Classification Problem

For classification problem, we use ‘**categorical\_crossentropy**’ as the loss function. It is similar to the log loss function.

We add **metrics = [‘accuracy’]** to print out the accuracy score at the end of each epoch.

The last layer should have n nodes corresponding to the n target categories, the activation function is **softmax**.

### Using models

We can use models to save, reload and make predictions.

## Fine-tuning Keras Models

### Why optimization is hard

We use stochastic gradient descent to tune the weight. The default value for SGD is 0.01, we can use different settings.

The dying neuron problem. Once the weight becomes zero or less, it stays at below zero and never gets updated.

The vanishing gradients problem occurs when many layers have very small slopes and updates to backprop were close to 0.

### Model Validation

In practice, few people run k-fold cross validation on deep learning models because deep learning is typically used on datasets. We usually trust a scope from a single validation run.

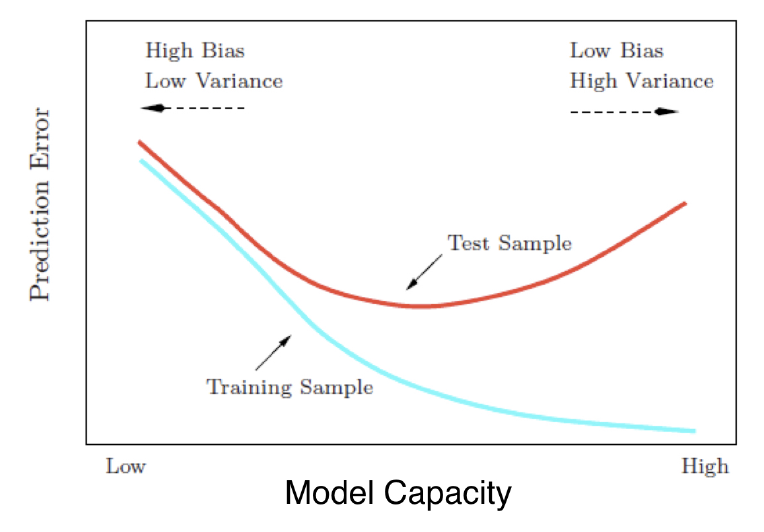
When calling the fit function, we can specify the split using the keyword argument validation\_split.

We can also use early stopping to stop the training if the model no longer makes progress after 2 or 3 epochs.

### Model Capacity

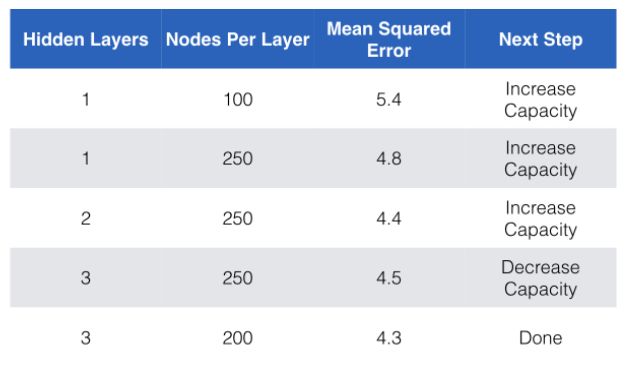
Model capacity is closely related to the terms overfitting and underfitting.

When we increase the number of layers and the nodes in the network, we move towards the right side of the x-axis, thus, risking overfitting.



#### Workflow for optimizing model capacity

* Start with a small network
* Gradually increase capacity
* Keep increasing capacity until validation score is no longer improving



# Extreme Gradient Boosting with XGBoost

## Classification with XGBoost

### Introduction to XGBoost

XGBoost can be applied to solve supervised learning problems, including classification and regression.

Classification problems involve predicting either binary or multi-class outcomes.

AUC or Area Under the Receiver Operating Characteristics Curve, is the most common metrics to judge the quality of the binary classification model. Larger the AUC, the more sensitive (the better) the model.

When dealing with multi-class classification problems, it is common to use the accuracy score and to look at the overall confusion matrix to evaluate the quality of a model.

#### Other supervised learning problems

**Ranking** predicts an ordering on a set of choices.

**Recommendation** recommends a set of items to a user.

### What is a decision tree?

A decision tree models a dataset with a tree structure. Each node is a binary question. Answering the question leads to next node. The leaf node contains the target label.

During construction, the tree is built one split at a time, and the split is selected so that the target values are segregated better until all (or nearly all) values within a given split are exclusively of one category or another.

Decision trees are very good at learning the relationship of the training dataset. But it tends to overfit and generalize poorly on new dataset.

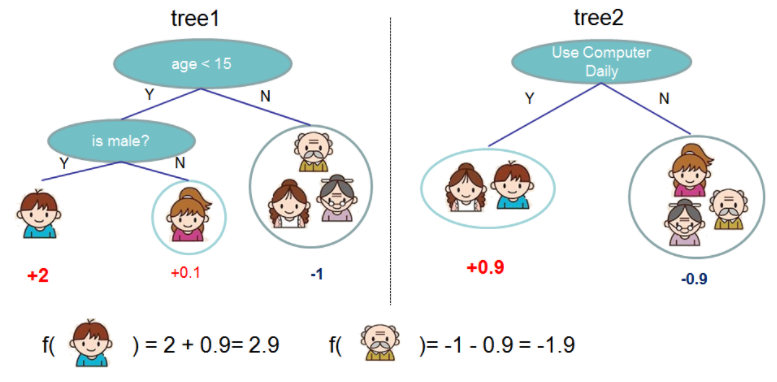
XGBoost uses a slightly different kind of decision tree called a classification and regression tree or **CART**. Whereas the decision tree’s leaf nodes contain decision values. CART trees contain a real-valued score in each leaf.

### What is Boosting?

Boosting is not a specific machine learning algorithm. It is a meta-algorithm.

It is an ensemble meta-algorithm used to convert many weak learners into a strong learner.

Boosting example:



<https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

#### Cross-validation in XGBoost

XGBoost has cross-validation capabilities baked in. In order to use cross-validation, we have to explicitly specify a DMatrix data structure.

### When to use XGBoost

Use XGBoost when

* You have a large number of training samples
  + Greater than 1000 training samples and less 100 features
  + The number of features < number of training samples
* You have a mixture of categorical and numeric features
  + Or just numeric features

Do not use XGBoost when

* Image recognition
* Computer Vision
* Natural language processing

## Regression with XGBoost

Linear regression and decision trees are common regression algorithms.

Decision trees can be used to solve both classification and regression problems, making them the prime candidate as base solvers in XGBoost.

#### Loss functions and base learners

Loss function names in xgboost:

* reg:linear – use for regression problem
* reg:logistic – classification problems with just decision, not probability
* binary:logistic –classification with probability rather than decision

The **booster** parameter can have value gbtree, gblinear or dart. <https://xgboost.readthedocs.io/en/latest/parameter.html#general-parameters>

#### Regularization is a control on model complexity

Gamma – minimum loss reduction allowed for a split to occur. Higher values lead to fewer splits.

Alpha – L1 regularization on leaf weights. Larger values mean more regularization and more leaf nodes to have 0 weights.

Lambda – L2 regularization, causing the leaf weights to smoothly decrease. If leaf weights changes sharply, it will lead to overfitting.

#### Base learners in XGBoost

Linear Base Learner

* Sum of linear terms
* Boosted model is weighted sum of linear models (thus is itself linear)
* Rarely used, same performance as regularized linear model.

Tree Base leaner

* Decision tree
* Boosted model is weighted sum of decision trees (thus is itself non-linear)

## Fine-tuning your XGBoost model

## Using XGBoost in pipelines