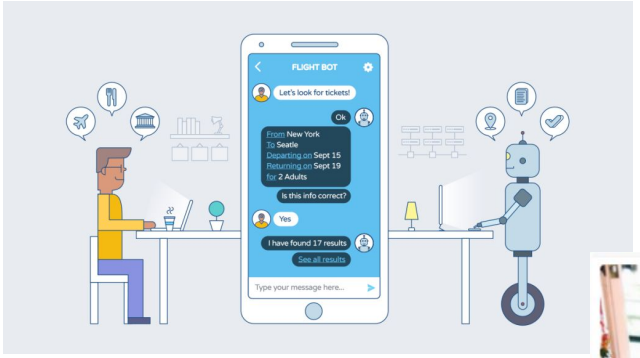
EEC1509 - Machine Learning

Lesson #3 - Fundamentals of Machine Learning

Ivanovitch Silva

August, 2018

In the past few years (....)



"man in black shirt is playing guitar."

Intense Media Hype

Machine Learning, Deep Learning and AI

Agenda

1. Key definitions
2. Types of Machine Learning
3. Machine Learning Workflow
4. Main challenges
5. End-to-end ML project

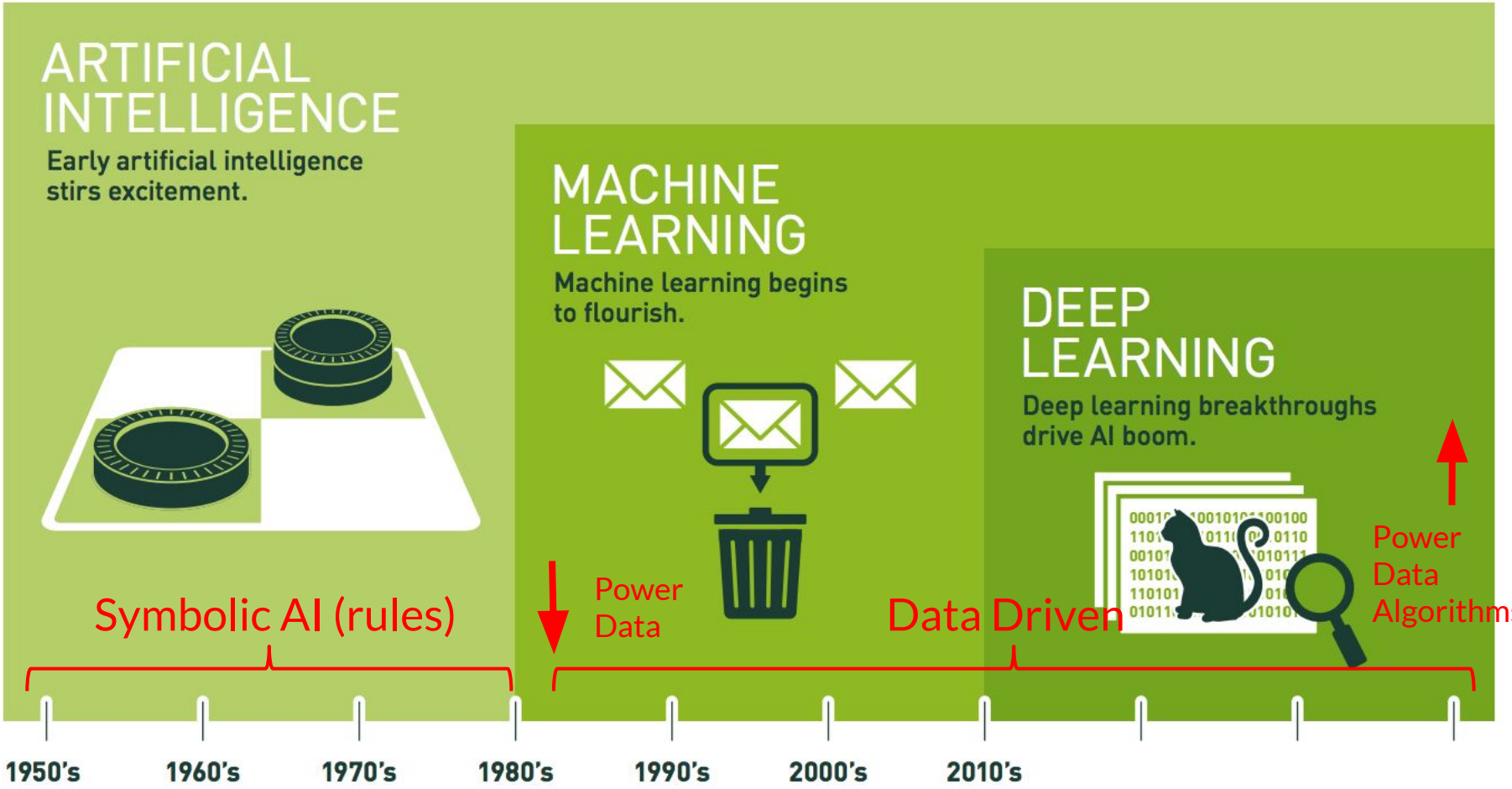
Update repository

```
git clone https://github.com/ivanovitchm/EEC1509_MachineLearning.git
```

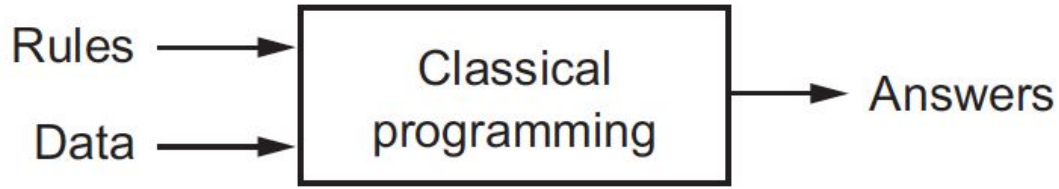
Ou ....

```
git pull
```

How do they related to each other?

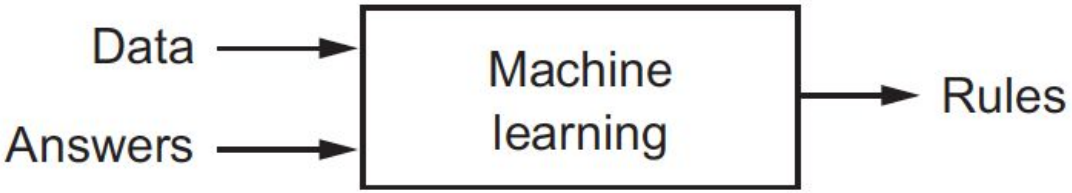


Symbolic AI (1950s to 1980s)



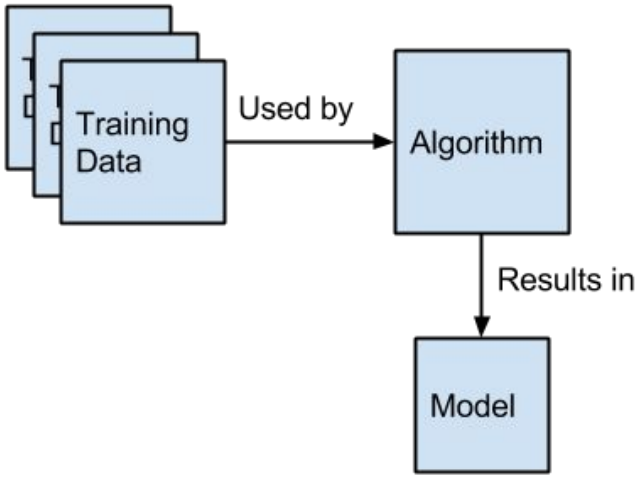
- It proved suitable to solve well-defined, logical problems, such as playing chess
- It turned out to be intractable to figure out explicit rules for solving more complex, such as image classification, speech recognition, and language translation

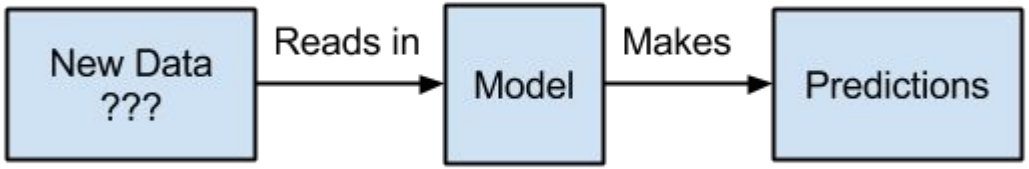
Machine Learning a new programming paradigm



- Humans input data as well as the answers expected from the data, and out come the rules.
- These rules can then be applied to new data to produce original answers.
- A machine-learning system is trained rather than explicitly programmed

Machine Learning a new programming paradigm





Machine Learning Definition

Arthur Samuel (1959)
Machine Learning: field of study that gives computers the ability to learn without being explicitly programmed.

https://en.wikipedia.org/wiki/Arthur_Samuel

Machine Learning Definition

Tom Mitchell (1998)
Well-posed Learning Problem: a computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

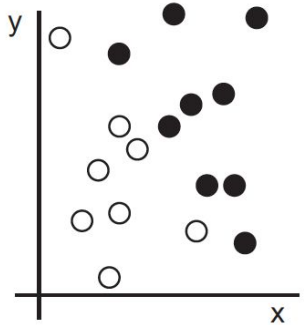
Example

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.

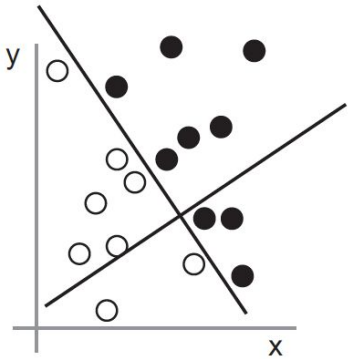
What is the task T in this setting? And P? And E?

Machine Learning Definition

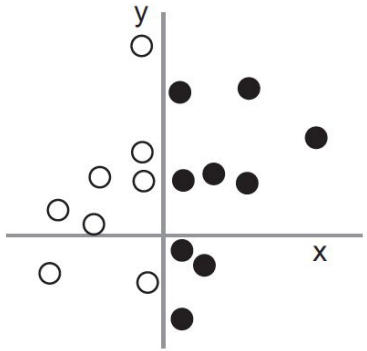
1: Raw data



2: Coordinate change

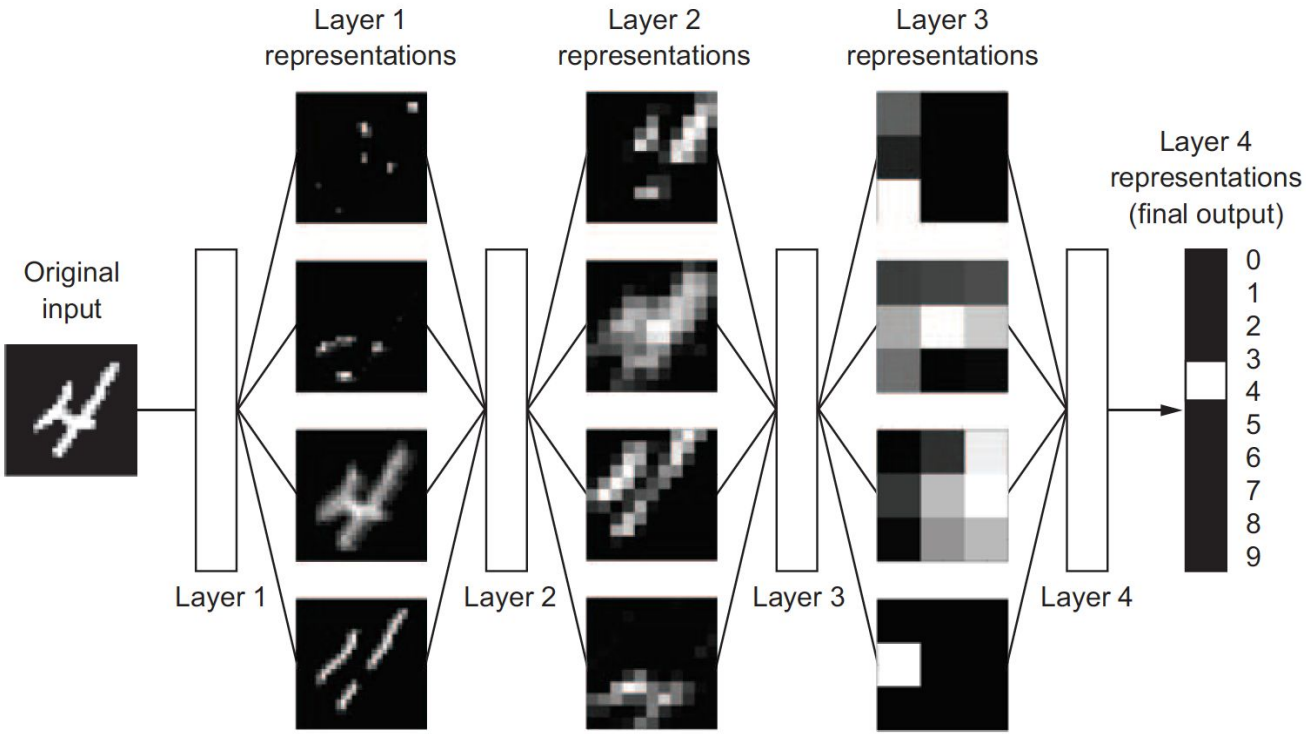


3: Better representation



An automatic search process for better data representations

What is Deep Learning?

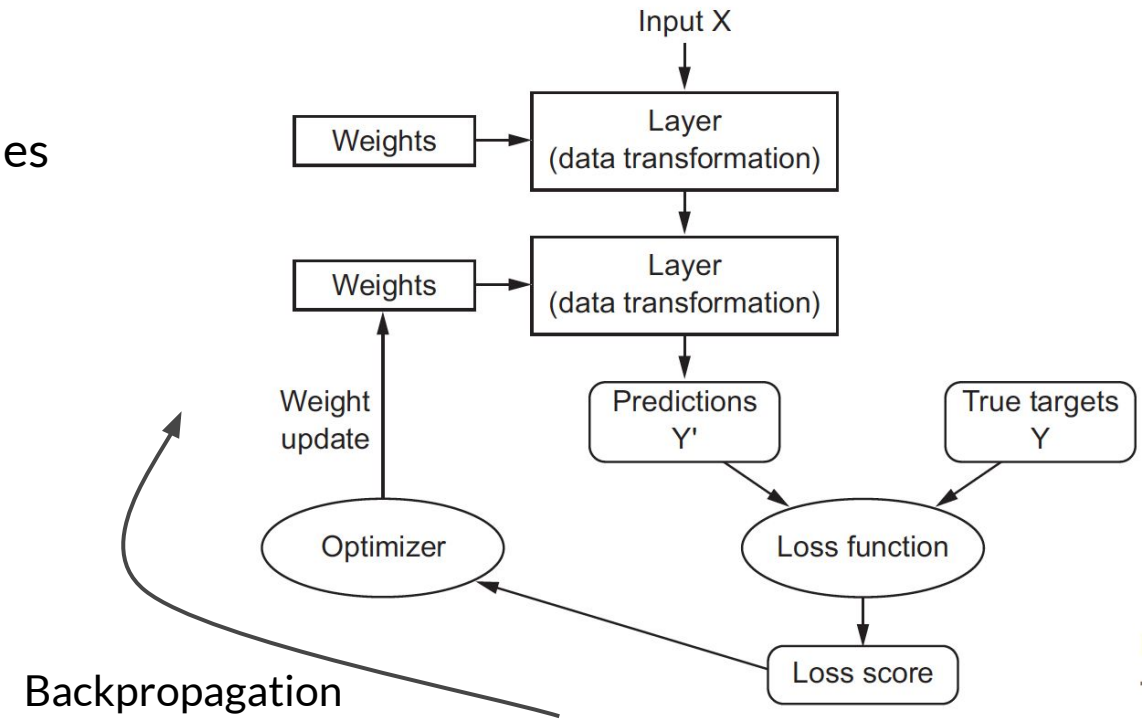


"Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. [. . .] The key aspect of deep learning is that these layers are not designed by human engineers: they are learned from data using a general-purpose learning procedure"

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, Nature 2015

Understanding how DL works

Finding the right values of weights which minimize the error

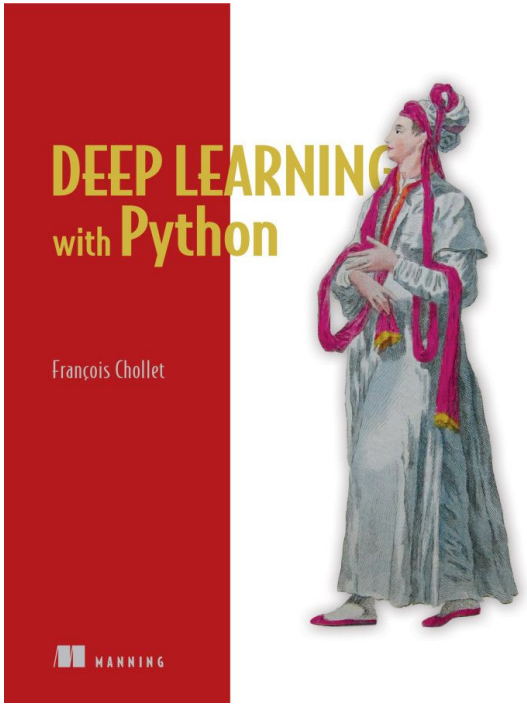


What DL has archieved so far

Problems involving skills that seem natural and intuitive to humans but have long been elusive for machines.

- Near-human-level image classification
- Near-human-level speech recognition
- Near-human-level handwriting transcription
- Improved machine translation
- Improved text-to-speech conversion
- Digital assistants such as Google Now and Amazon Alexa
- Near-human-level autonomous driving
- Improved ad targeting, as used by Google, Baidu, and Bing
- Improved search results on the web
- Ability to answer natural-language questions
- Superhuman Go playing

A brief history of Machine Learning

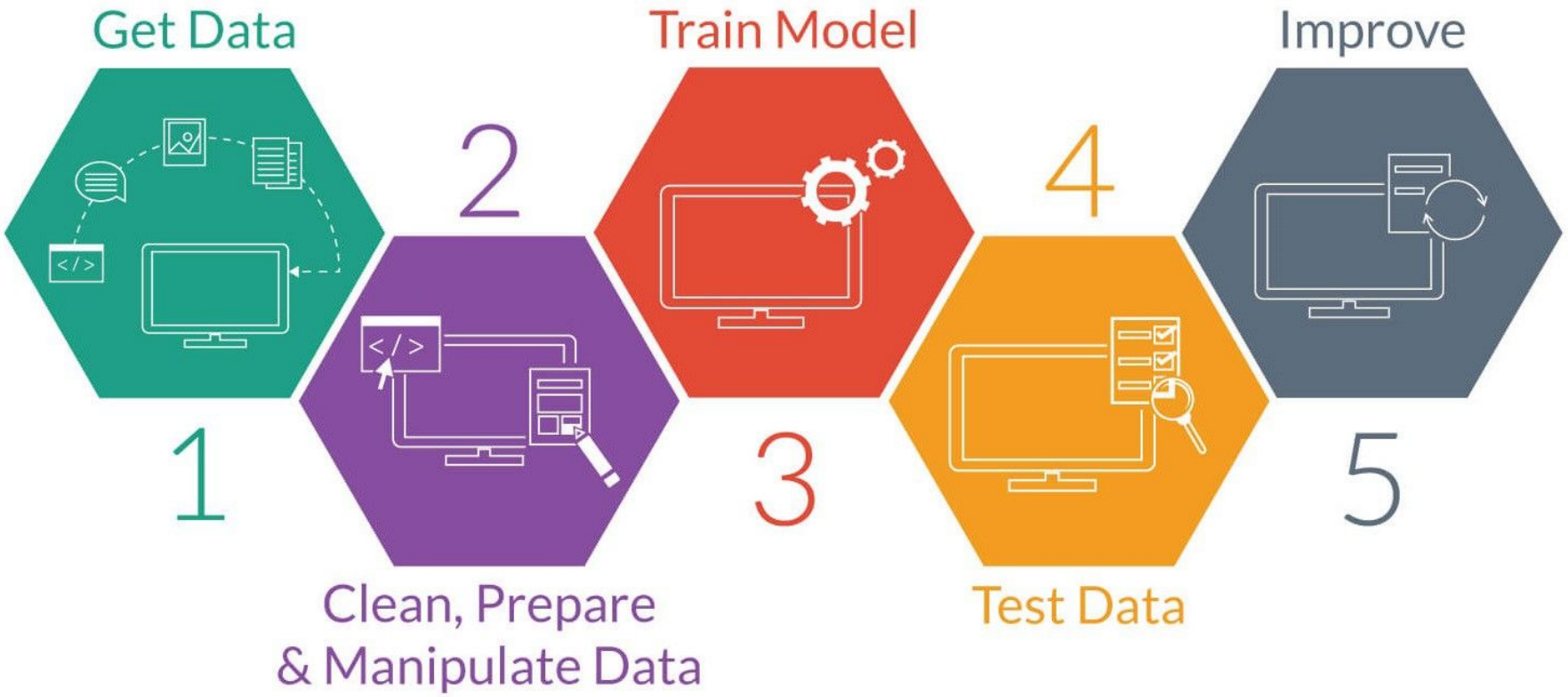


https://www.manning.com/books/deep-learning-with-python

Chapter 1, 2 and 3 are available for free!!!
Read the Chapter. 1, Section 1.2 !!!!!


It's time to
move on...

A general ML workflow



Types of Machine Learning Systems

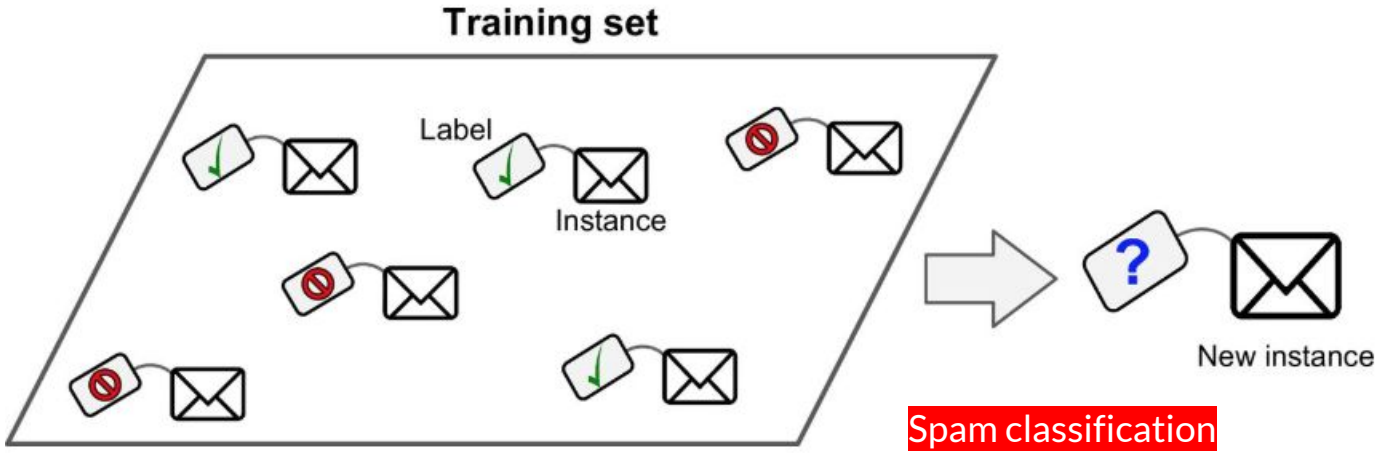
- Whether or not they are trained with human supervision
  - Supervised
  - Unsupervised
  - Semisupervised
  - Reinforcement learning
- Whether or not they can learn incrementally on the fly
  - Online
  - Batch learning
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model
  - Instance-based learning
  - Model-based learning

Types of Machine Learning Systems

- Whether or not they are trained with human supervision
  - Supervised
  - Unsupervised
  - Semisupervised
  - Reinforcement learning
- Whether or not they can learn incrementally on the fly
  - Online
  - Batch learning
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model
  - Instance-based learning
  - Model-based learning

Supervised Learning

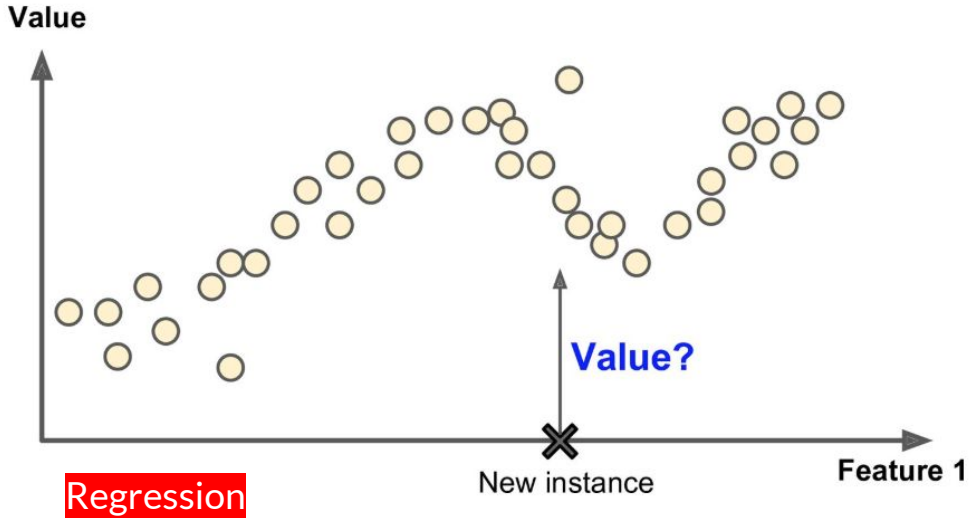
In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels.



Spam classification

Supervised Learning

Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc) called predictors.



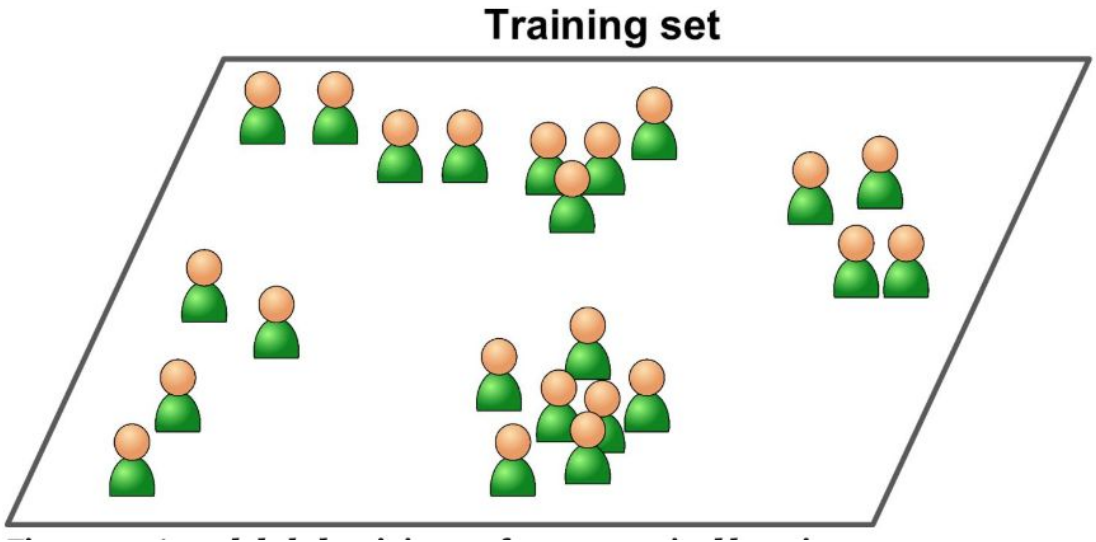
Regression

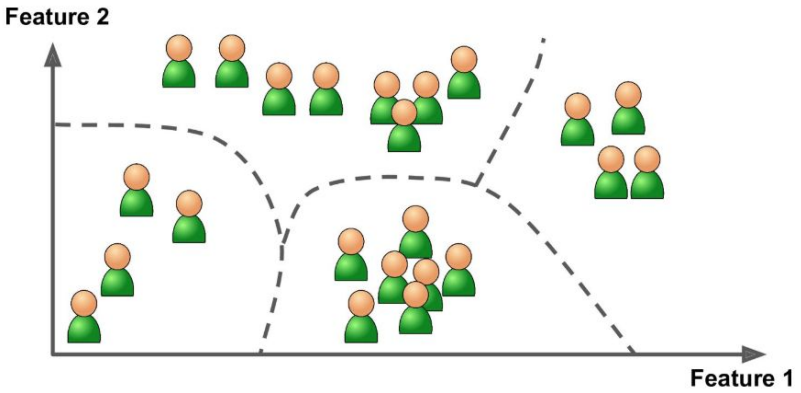
Supervised Learning

- K-Nearest Neighbors (KNN)
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVM)
- Decision Trees and Random Forests
- Neural Networks
- Deep Learning
- XGBoost

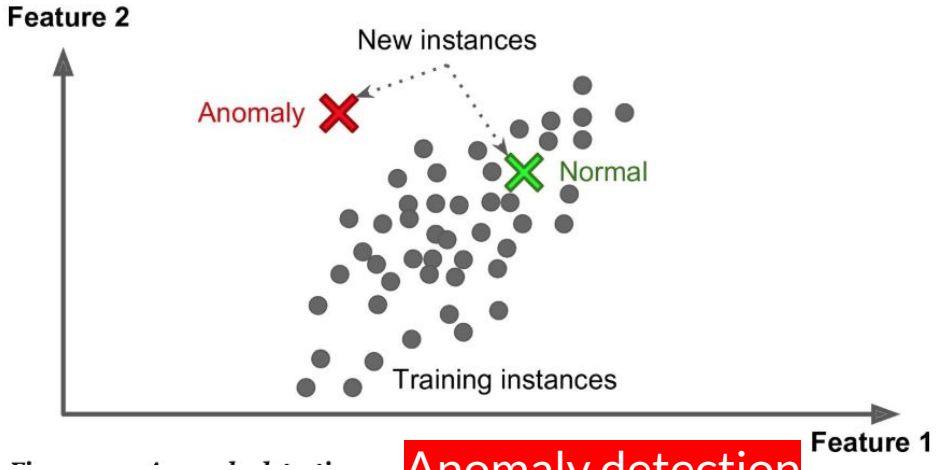
Unsupervised Learning

In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.

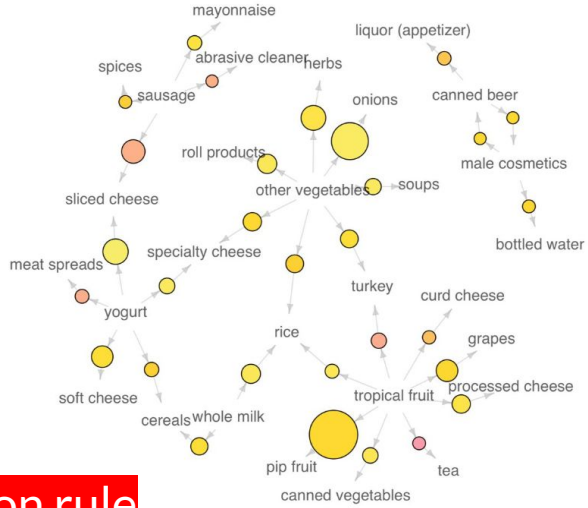




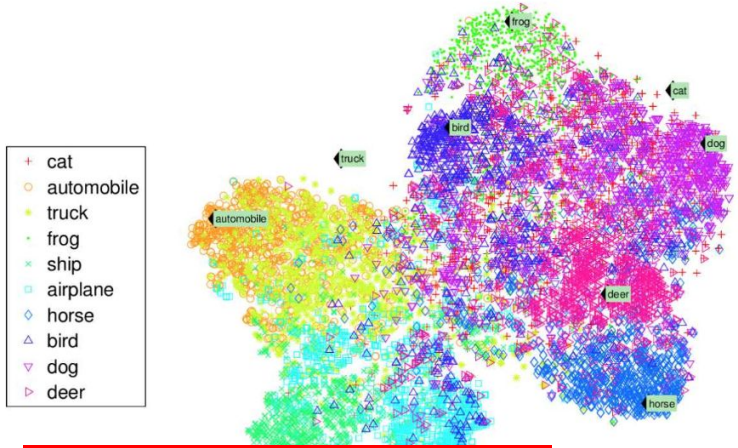
Clustering



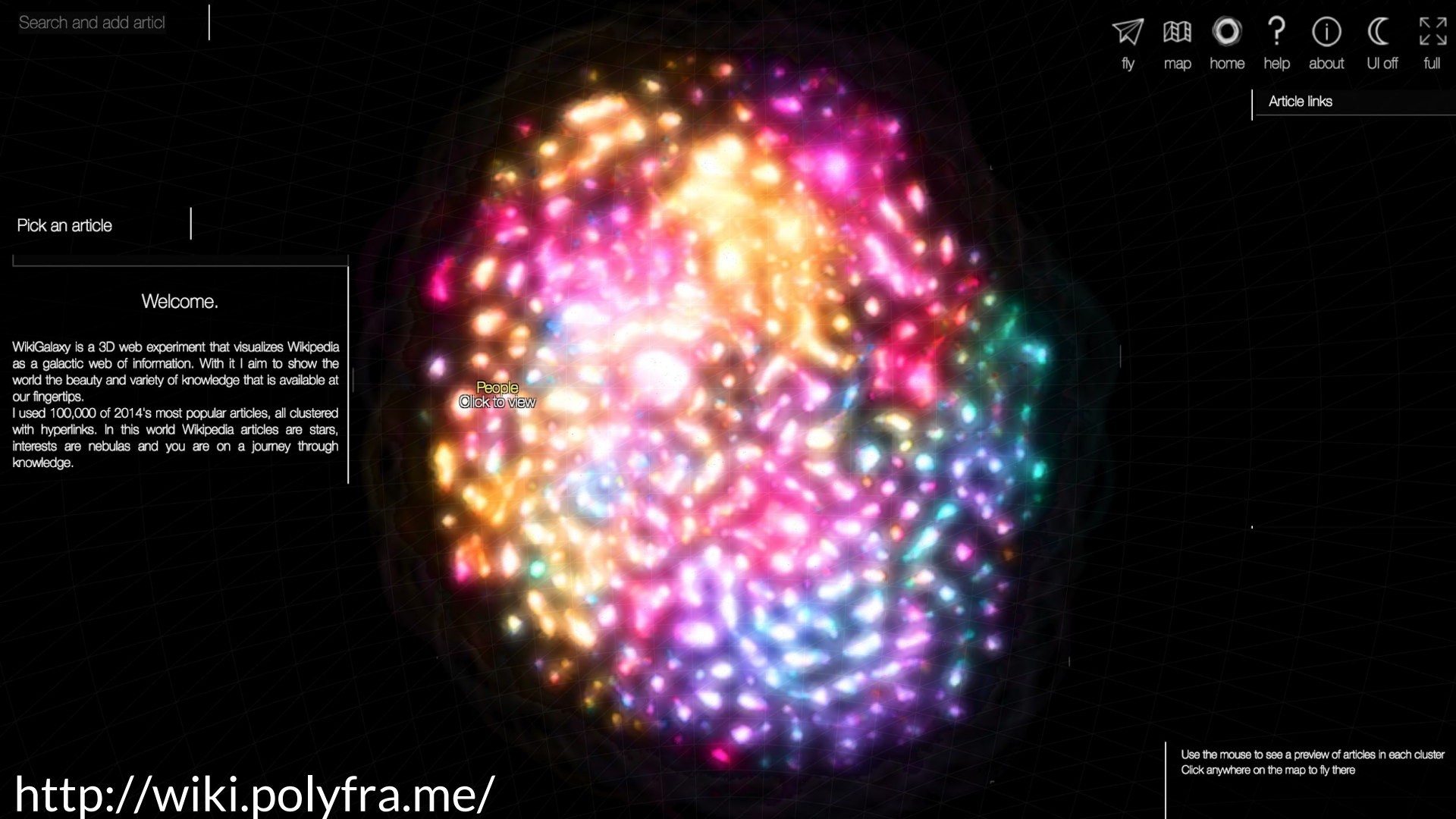
Anomaly detection



Association rule



Visualization highlighting


Search and add articl
fly
map
home
help
about
UI off
full
Article links
Pick an article
Welcome.
WikiGalaxy is a 3D web experiment that visualizes Wikipedia as a galactic web of information. With it I aim to show the world the beauty and variety of knowledge that is available at our fingertips.
I used 100,000 of 2014's most popular articles, all clustered with hyperlinks. In this world Wikipedia articles are stars, interests are nebulas and you are on a journey through knowledge.
People
Click to view
Use the mouse to see a preview of articles in each cluster
Click anywhere on the map to fly there
http://wiki.polyfra.me/

Unsupervised Learning

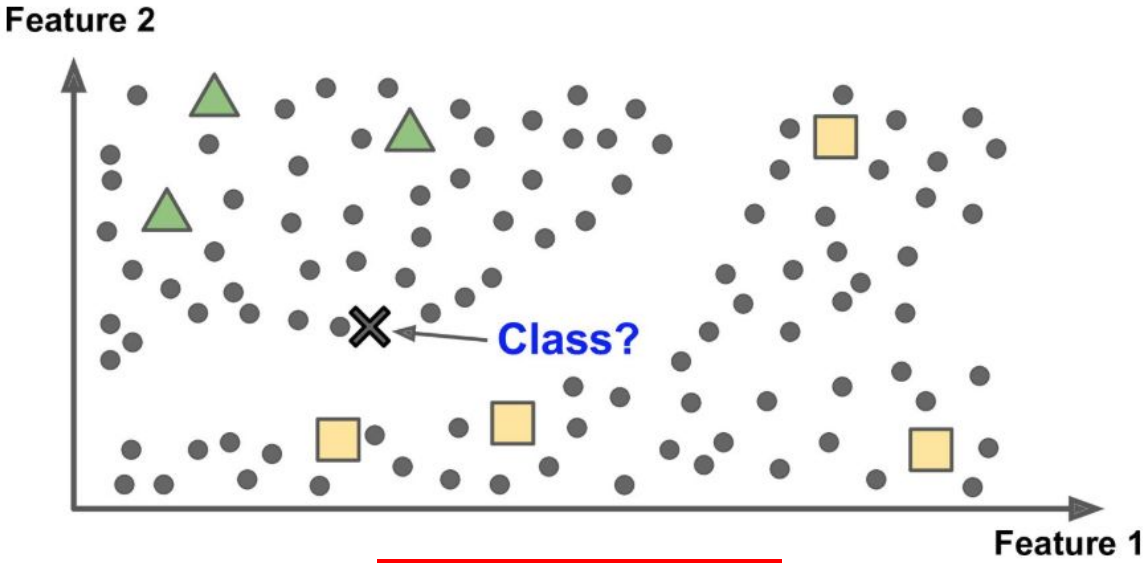
- Clustering
  - K-Means
  - Hierarchical Cluster Analysis (HCA)
  - Expectation Maximization
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally-Linear Embedding (LLE)
  - T-distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
  - Apriori
  - Eclat

Semisupervised Learning

Semi-supervised learning algorithms are trained on a combination of labeled and unlabeled data.

- The process of labeling massive amounts of data for supervised learning is often prohibitively time-consuming and expensive.
- What's more, too much labeling can impose human biases on the model.
- That means including lots of unlabeled data during the training process actually tends to improve the accuracy of the final model while reducing the time and cost spent building it

Semisupervised Learning

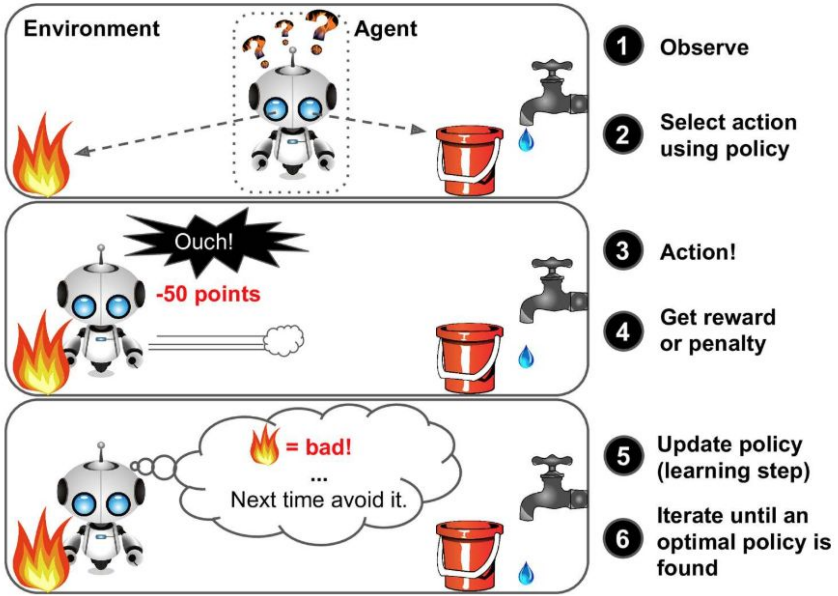


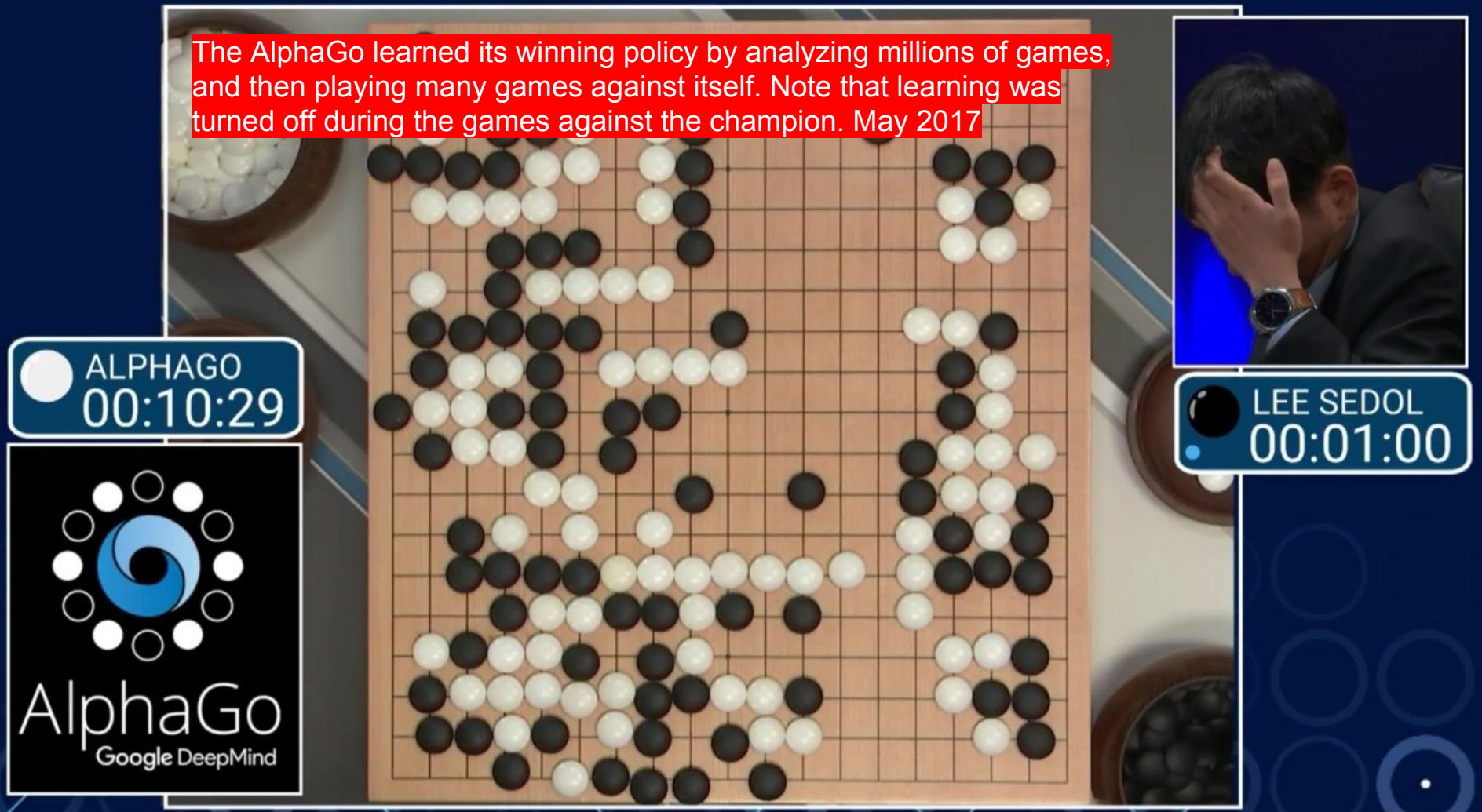
Web Page Classification
Google Photos

Reinforcement Learning

The learning system, called an agent in this context, can :

- Observe the environment
- Select and perform actions
- Get rewards in return (positive or negative)
- Learn by self what is the best strategy, called a policy

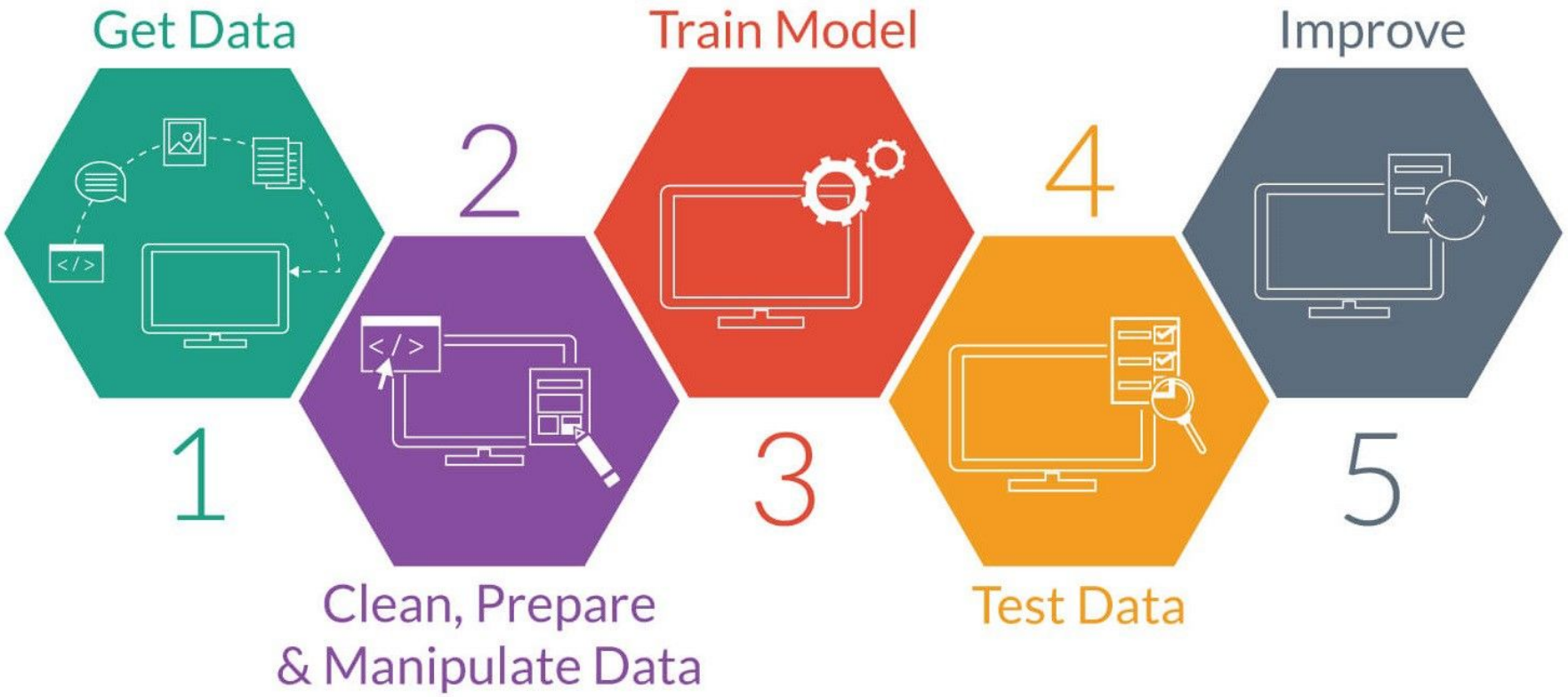



The AlphaGo learned its winning policy by analyzing millions of games, and then playing many games against itself. Note that learning was turned off during the games against the champion. May 2017
ALPHAGO
00:10:29
AlphaGo
Google DeepMind
LEE SEDOL
00:01:00

Reinforcement Learning

- Q-Learning
- State-Action-Reward-State-Action (SARSA)
- Deep Q Network (DQN)
- Deep Deterministic Policy Gradient (DDPG)
- A Brief Survey of Deep Reinforcement Learning (Dec, 2017)
  - https://arxiv.org/pdf/1708.05866.pdf

A general ML workflow

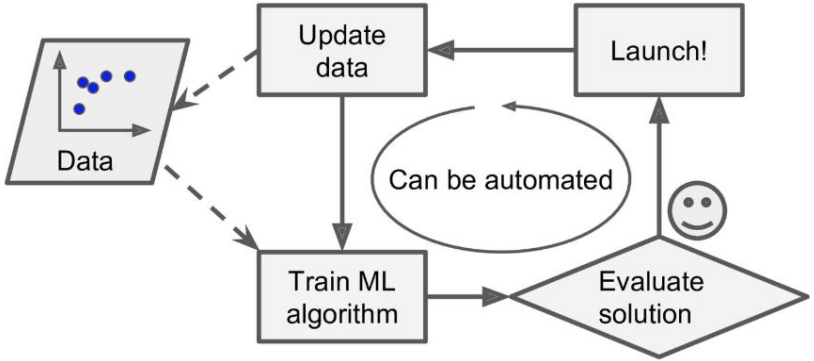


Types of Machine Learning Systems

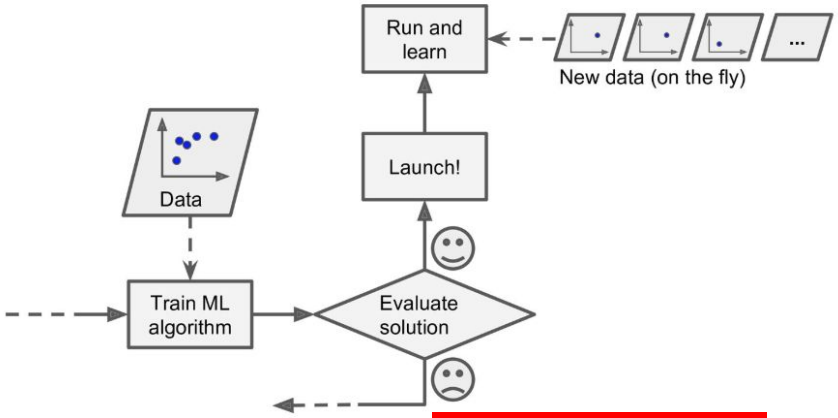
- Whether or not they are trained with human supervision
  - Supervised
  - Unsupervised
  - Semisupervised
  - Reinforcement learning
- Whether or not they can learn incrementally on the fly
  - Online
  - Batch learning
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model
  - Instance-based learning
  - Model-based learning

Batch and Online Learning

Another criterion used to classify Machine Learning systems is whether or not the system can learn incrementally from a stream of incoming data.



Batch Learning



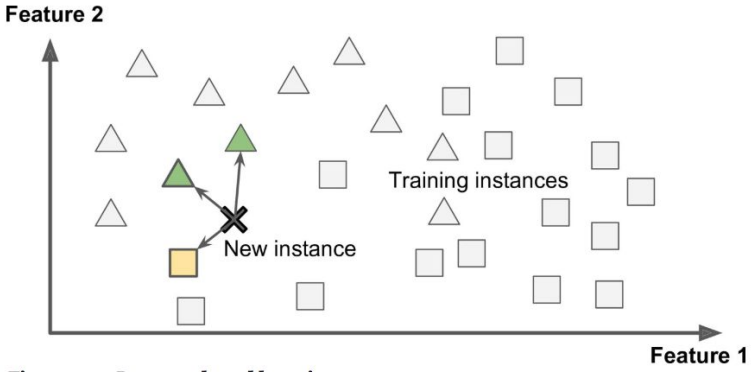
Online Learning

Types of Machine Learning Systems

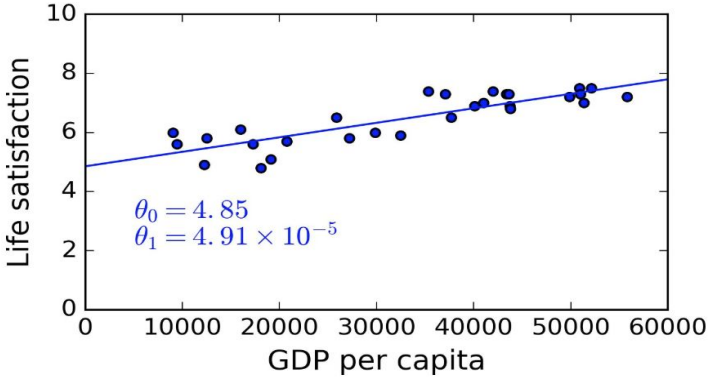
- Whether or not they are trained with human supervision
  - Supervised
  - Unsupervised
  - Semisupervised
  - Reinforcement learning
- Whether or not they can learn incrementally on the fly
  - Online
  - Batch learning
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model
  - Instance-based learning
  - Model-based learning

Instance-Based or Model-Based Learning

One more way to categorize ML is by how they generalize (classify or predict to examples it has never seen before).



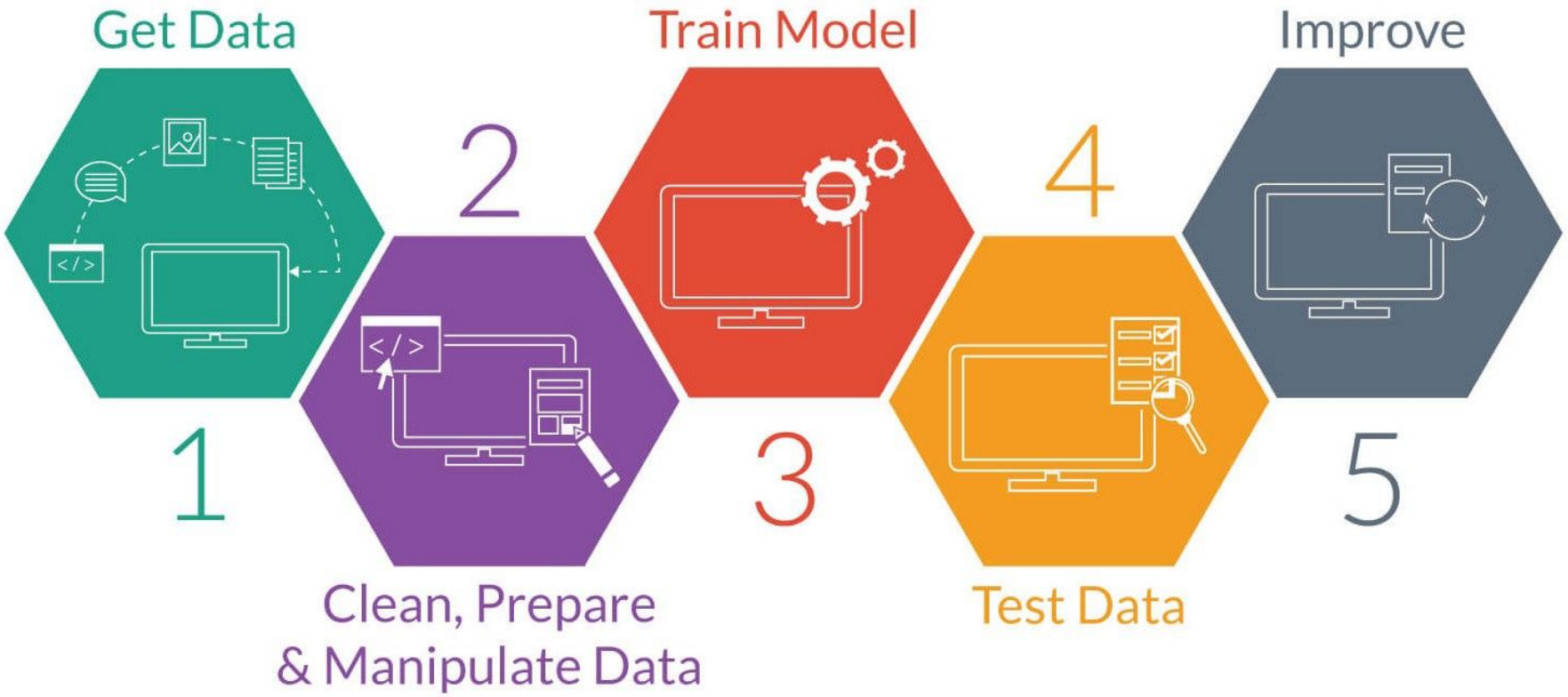
Instance-Based



Model-Based

main challenges of Machine Learning

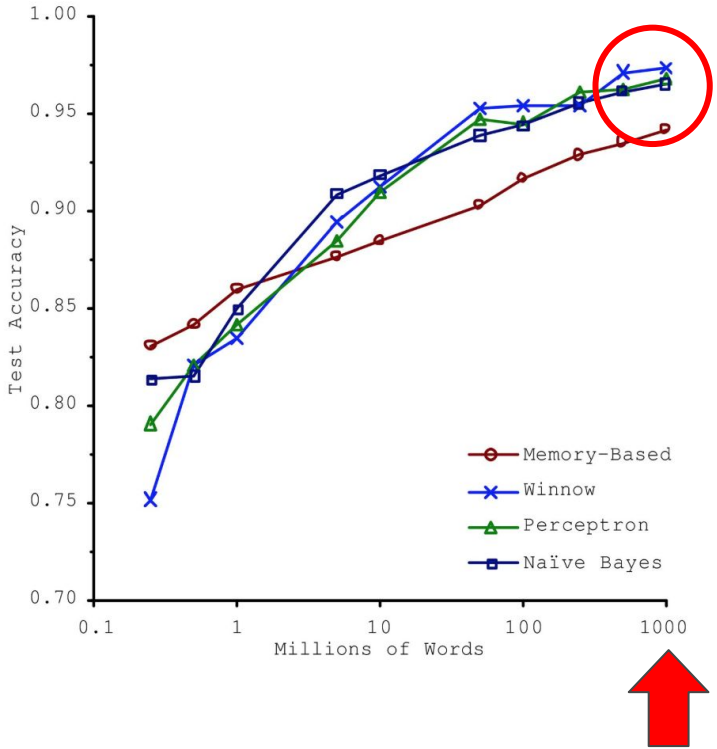
A general ML workflow



Main Challenges of Machine Learning

- Insufficient quantity of training data
- Nonrepresentative training data
- Poor quality-data
- Irrelevant features
- Overfitting the training data
- Underfitting the training data

Insufficient quantity of training data



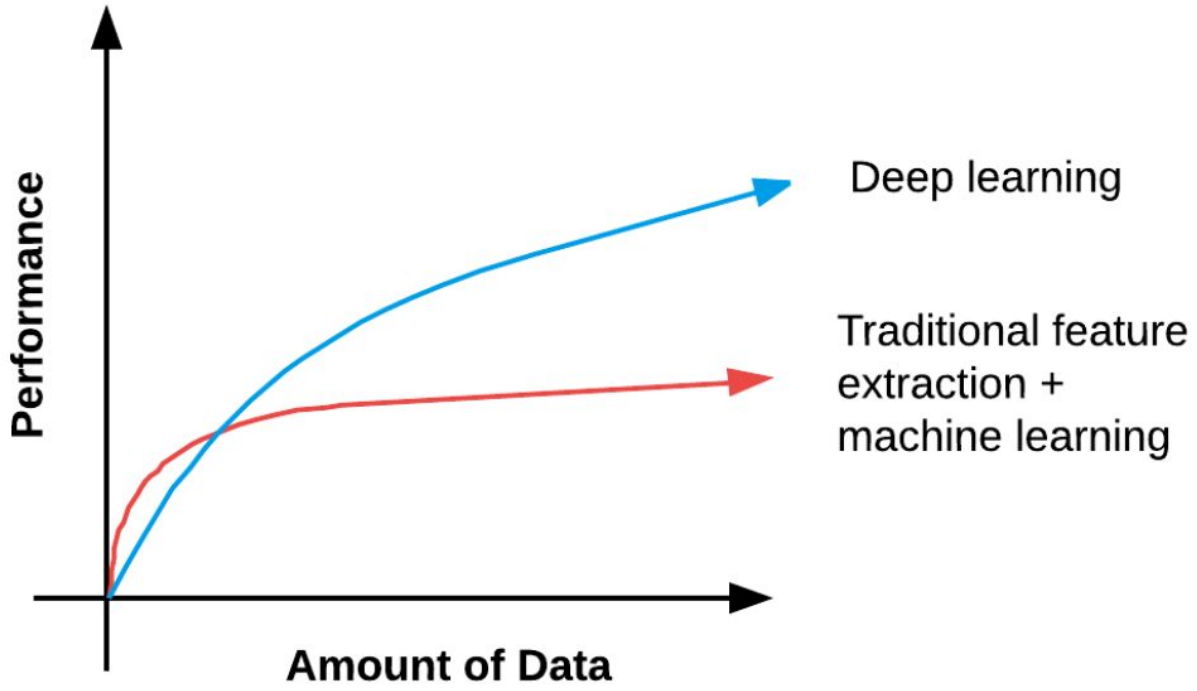
In a famous paper published in 2001, Microsoft researchers Michele Banko and Eric Brill showed that very differente ML algorithms, including fairly simples ones, performed almost identically well on a complex problem of natural language disambiguation once they were given enough data

Insufficient quantity of training data

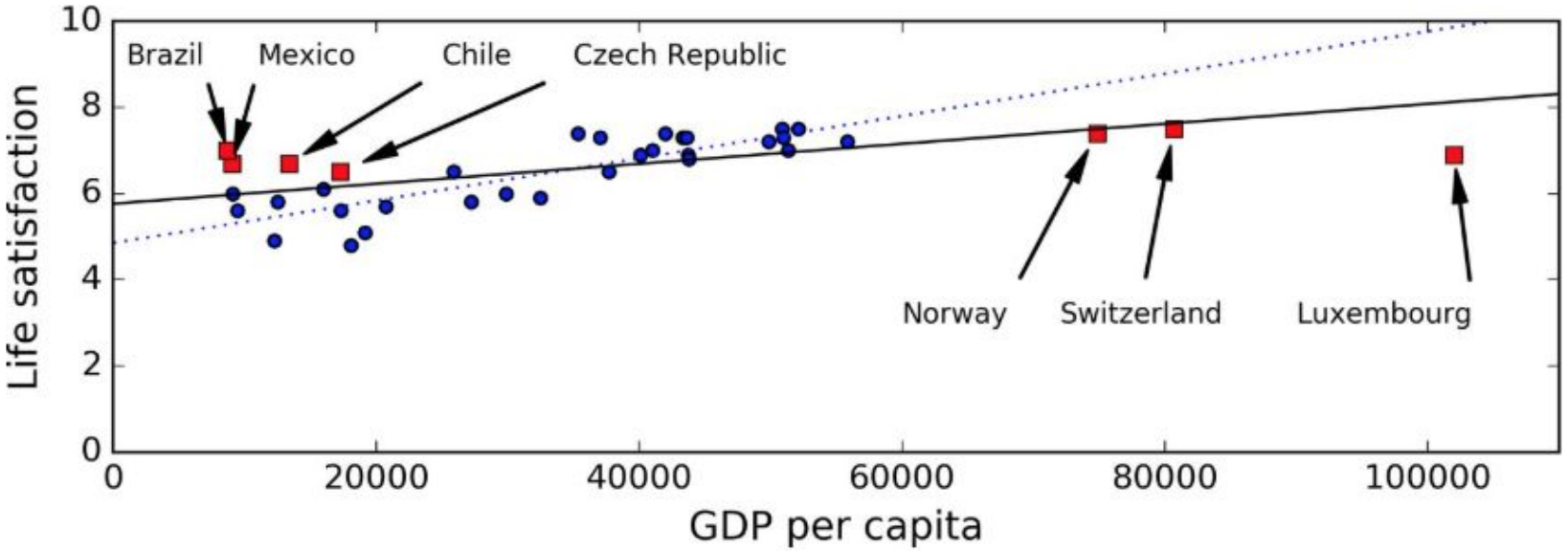
The idea that data matters more than algorithms for complex problems was further popularized by Peter Norvig et al. in a paper titled "The Unreasonable Effectiveness of Data" published in 2009.

Researchers from Google and Carnegie Mellon (2017) took a step towards clearing the clouds of mystery surrounding the relationship between "enormous data" and visual deep learning. we find that the performance on vision tasks increases logarithmically based on volume of training data size (300M images were labeled with 18291 categories resulting in more than 1 billion of labels)

Insufficient quantity of training data



Nonrepresentative Training Data



By using a nonrepresentative training set, we trained a model that is unlikely to make accurate predictions (sampling bias), especially for very poor and very rich countries

Poor-Quality Data

Obviously, if your training data is full of:

- Erros
- Outliers
- Noise
- Missing data

It will make it harder for the system to detect underlying patterns.

The truth is, most data scientist spend a significant part of their time doing cleaning up your training data.

Irrelevant Features

A critical part of the success of a ML project is coming up with a good set of features to train on. This process, called feature engineering involves:

- Feature selection
  - Selecting the most useful features to train on among existing features.
- Feature extraction
  - Combining existing features to produce a more useful one
- Creating new features by gathering new data.

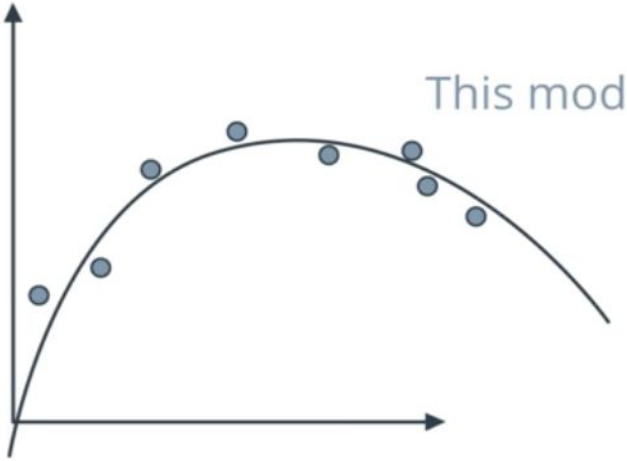
UNDERFITTING

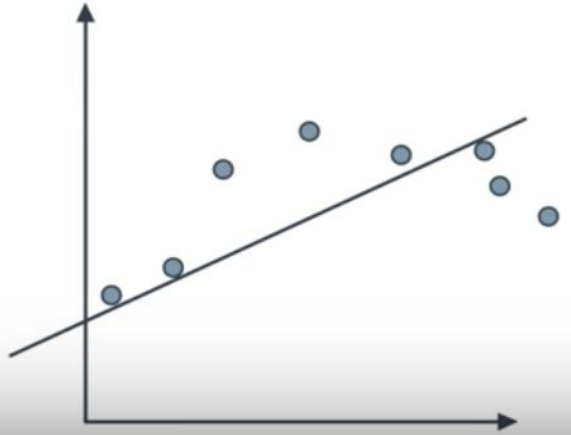
OVERFITTING

UNDERFITTING

Error due to bias

This model will not do well in the training set

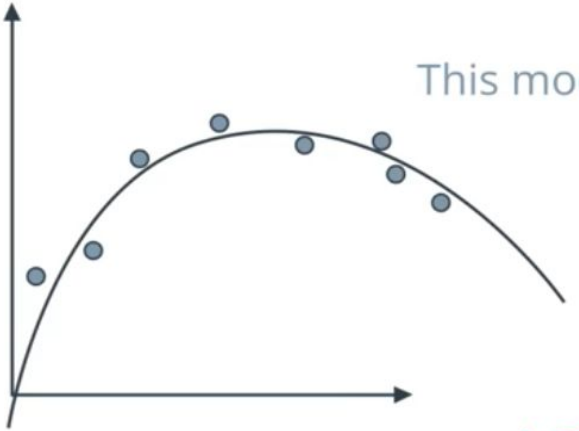


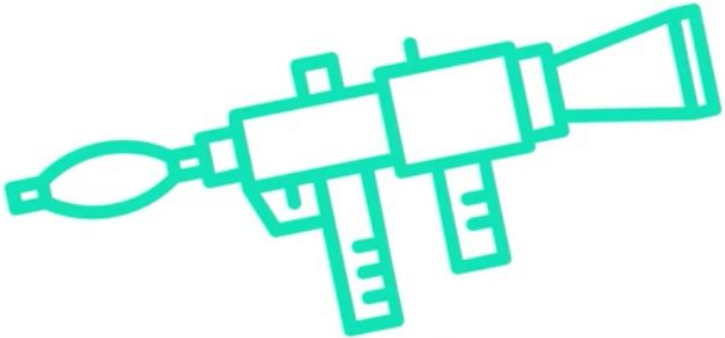


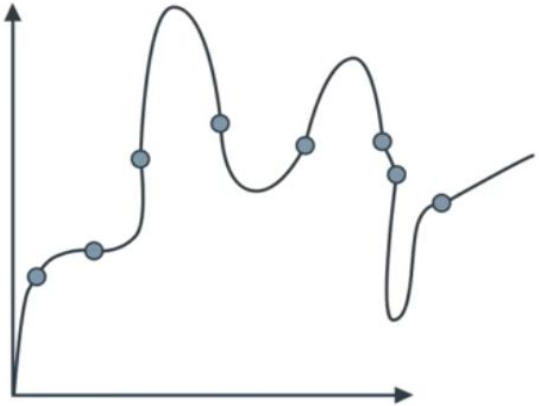
OVERFITTING

Error due to variance

This model performs poorly in the testing set

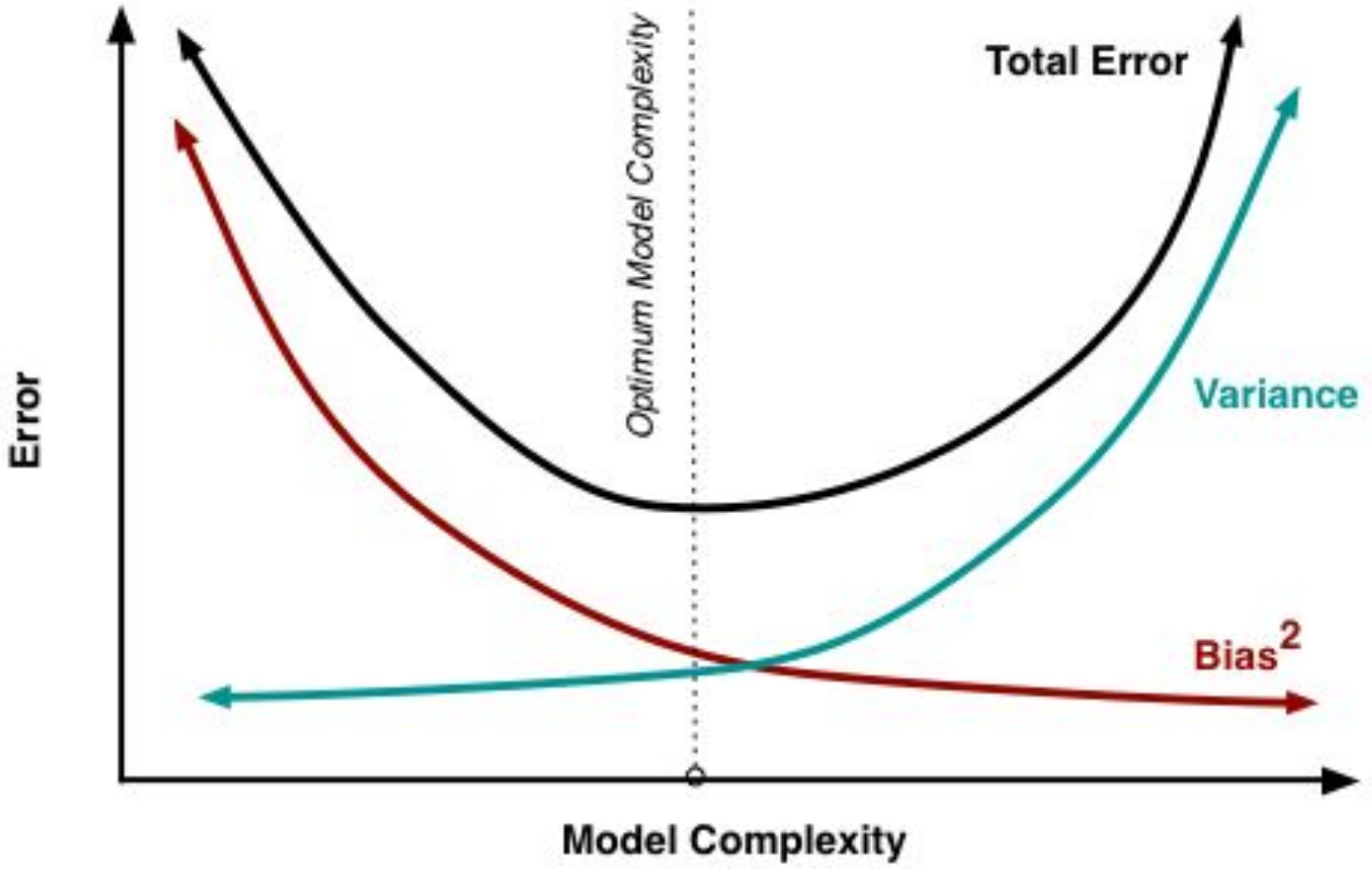






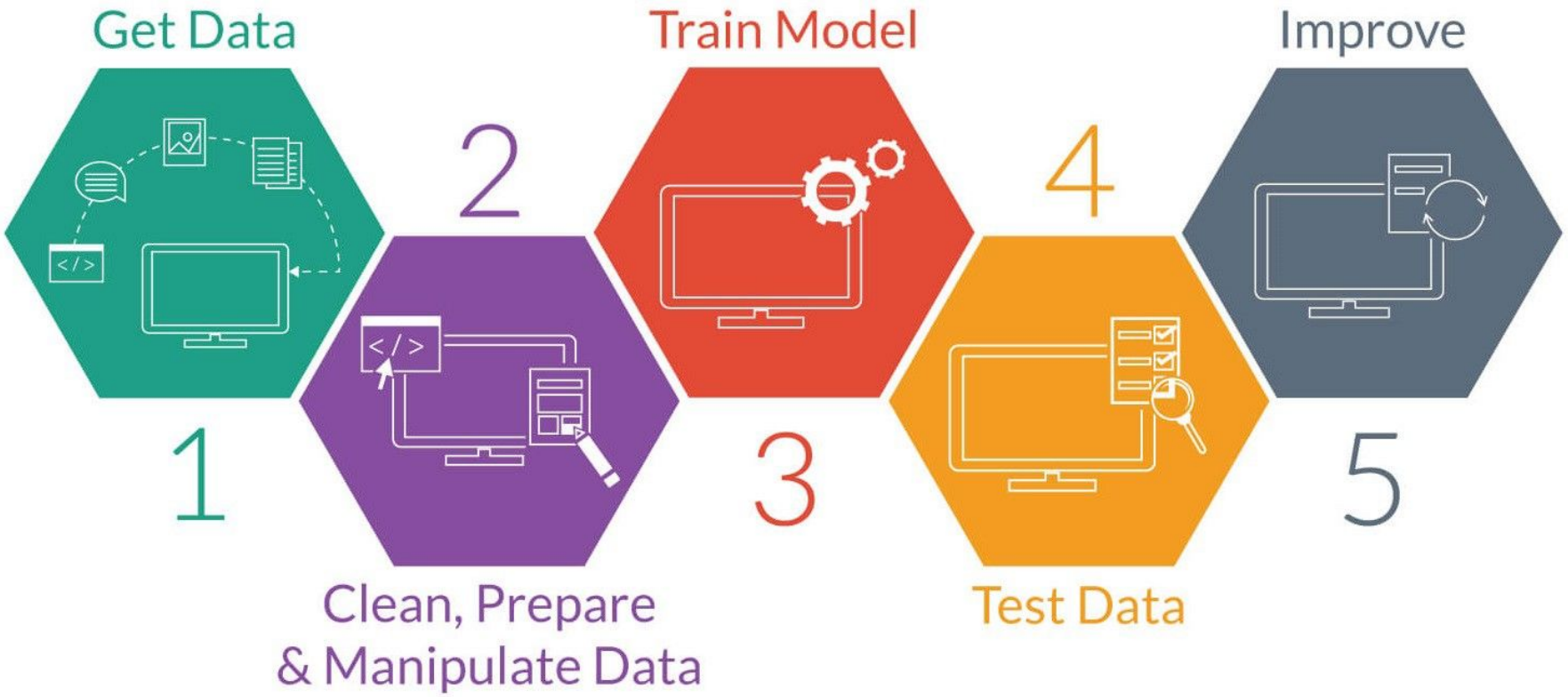
Underfitting

Overfitting



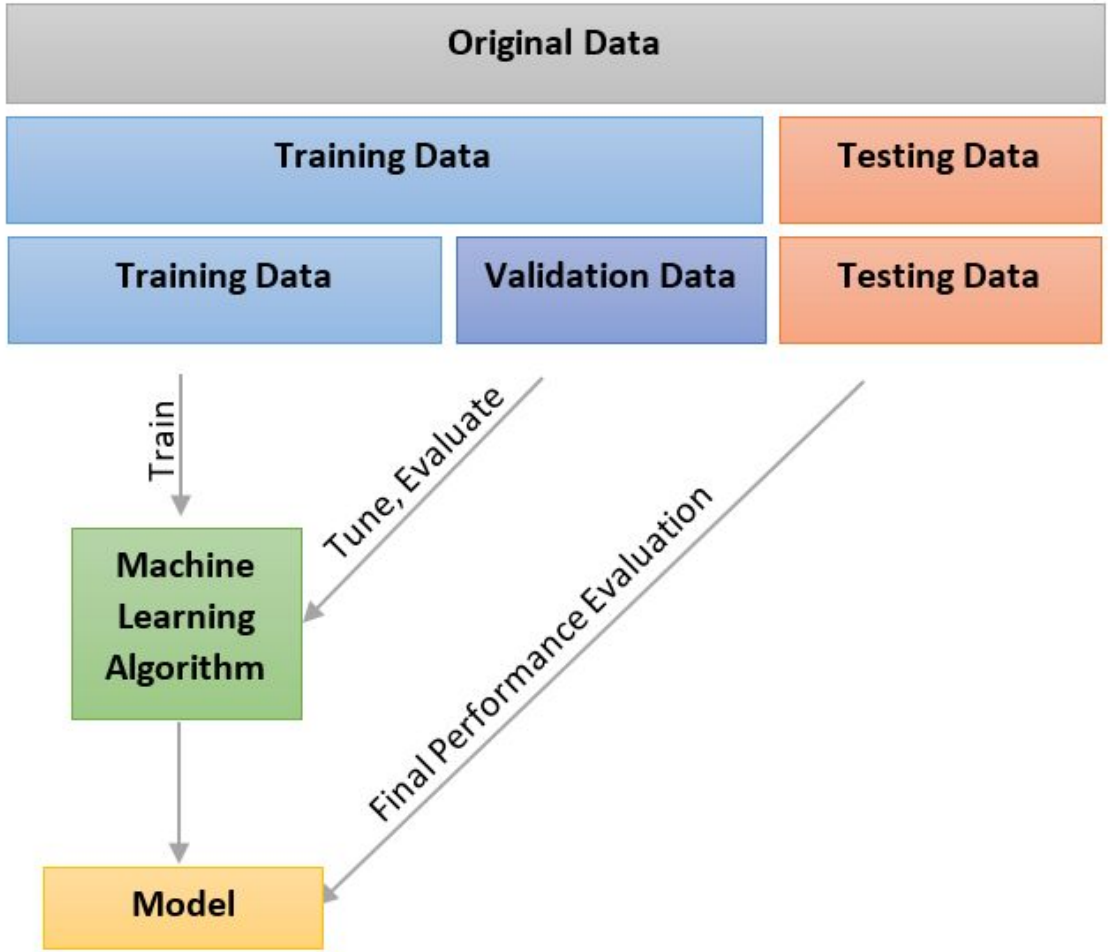
Testing & Validating

A general ML workflow

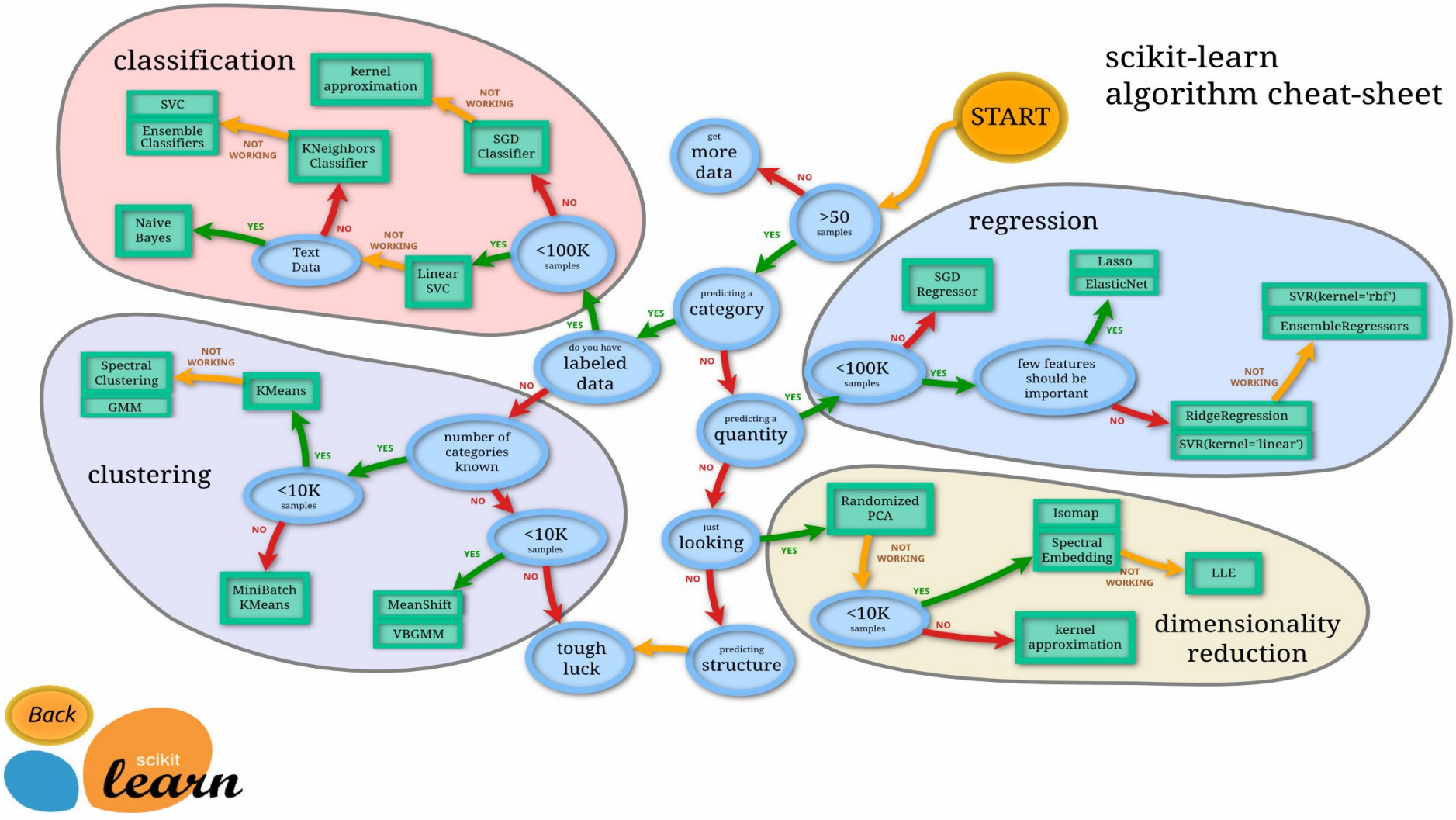


The only way to know how well a model will generalize to new cases is to actually try it out on new cases.

One way to do that is to put your model in production and monitor how well it performs!!!


Original Data
Training Data
Testing Data
Training Data
Validation Data
Testing Data
Train
Tune, Evaluate
Final Performance Evaluation
Machine Learning Algorithm
Model

End-to-end Machine Learning Project


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

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www

LINK

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

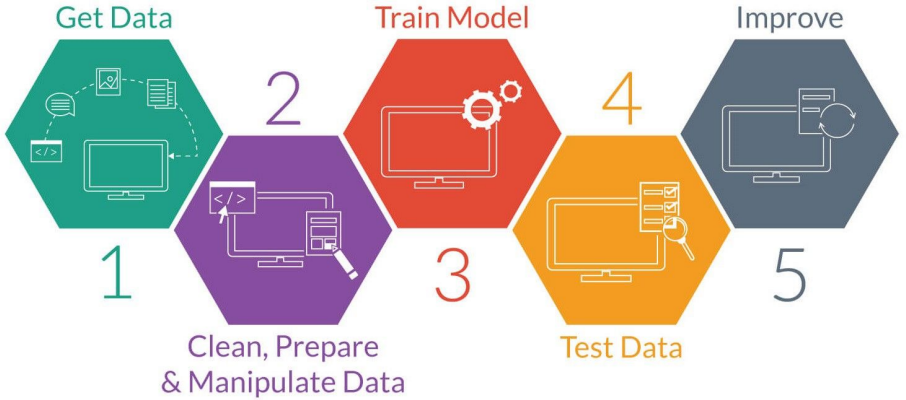
Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

End-to-end ML Project



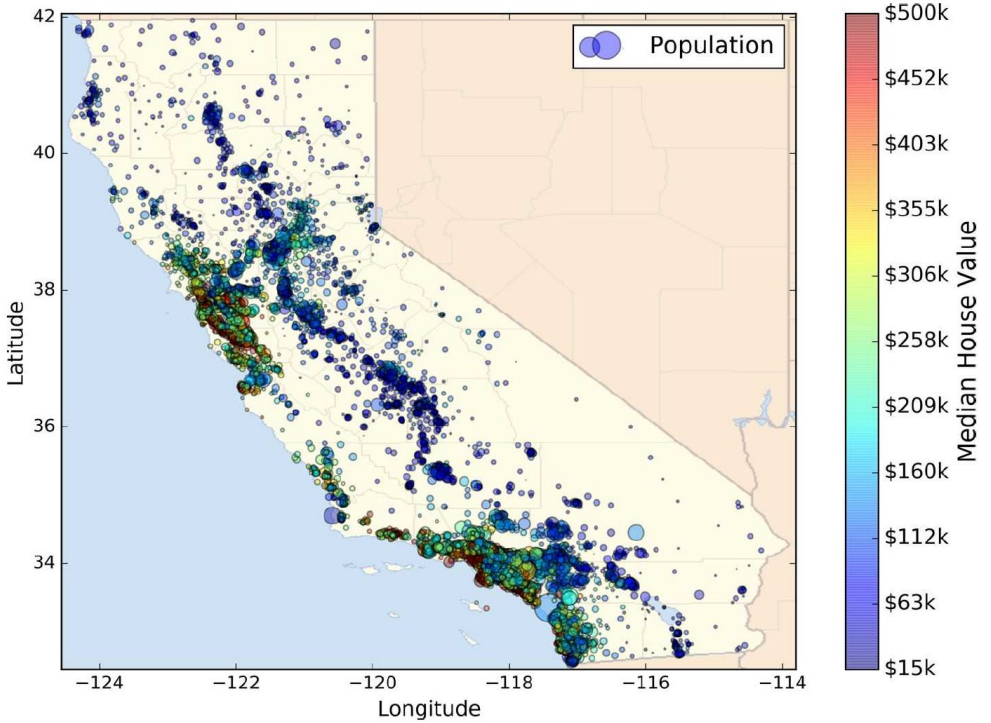
1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

#1 Look at the Big Picture

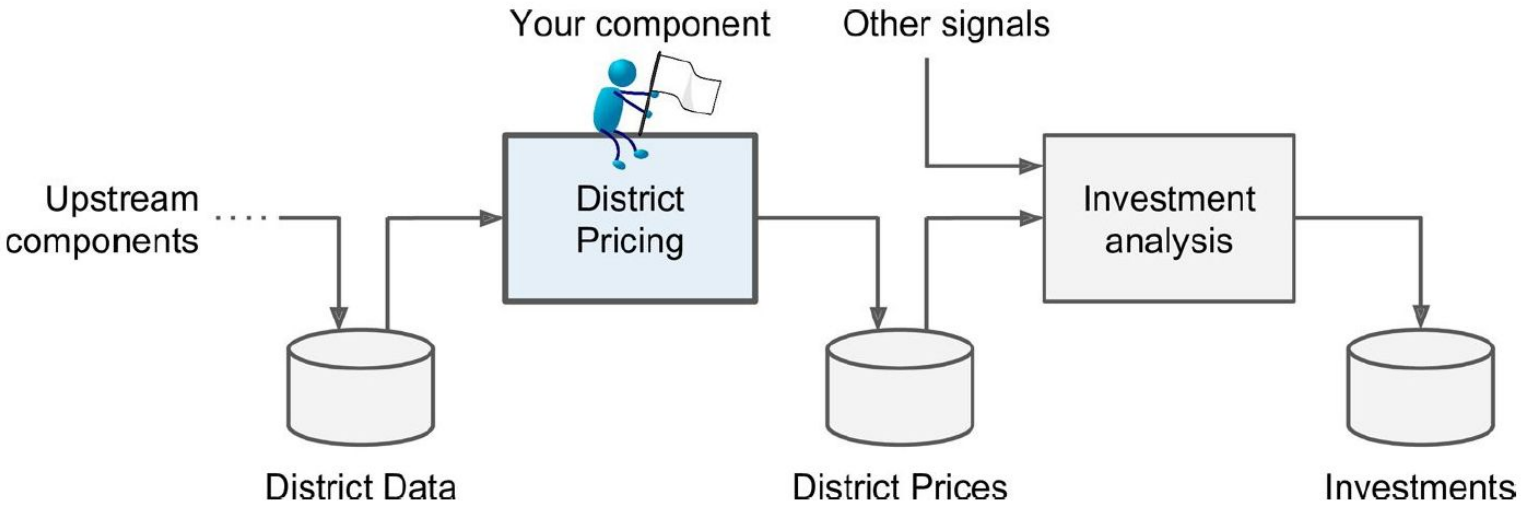
California Housing Prices dataset (90s)
https://www.kaggle.com/datasets

The first task you are asked to perform is to build a model of housing prices (block groups or districts) in California using the California census data

- Population
- Median income
- Median housing price
- and so on



Frame the problem



Building a model is probably not the end goal

Frame the problem

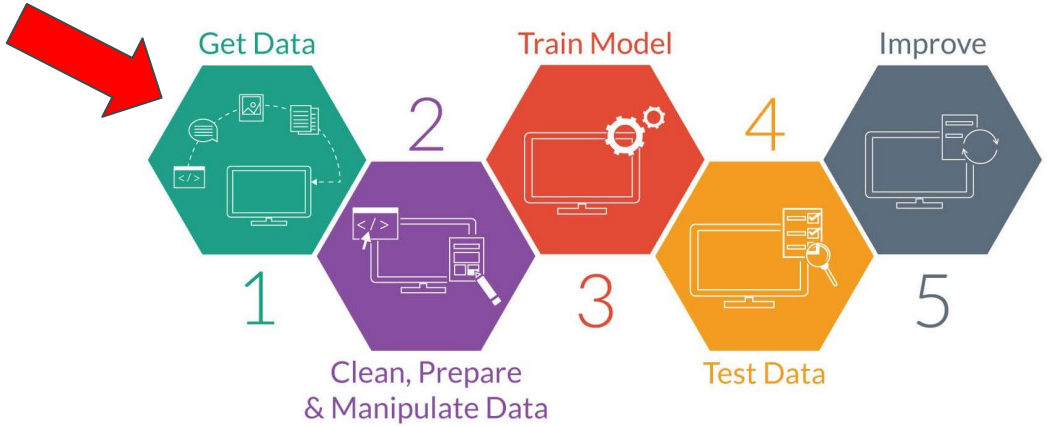
1. Is it supervised, unsupervised, or reinforcement learning?
2. Is it a classification task, a regression task, or something else?
3. Should you use batch learning or online learning techniques?
4. What algorithms you will select
5. What performance measure you will use to evaluate your model

Select a performance measure

$$RMSE(X, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

$$x^{(1)} = \begin{bmatrix} -118.29 \\ 33.91 \\ 1.416 \\ 38.372 \end{bmatrix} \qquad y^{(1)} = 156.400 \qquad X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(1999)})^T \\ (x^{(2000)})^T \end{bmatrix} = \begin{bmatrix} -118.29 & 33.91 & 1.416 & 38.372 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

End-to-end ML Project



1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine. Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

#2. Get the data

```
import pandas as pd

# read the dataset to a Pandas' dataframe
data = pd.read_csv("housing.csv")
data.head()
```

- **longitude**: computed distances among the centroids of each district as measured in longitude.
- **latitude**: computed distances among the centroids of each district as measured in latitude.
- **housing_median_age**: median age of district housing at location
- **total_rooms**: total rooms in the district
- **total_bedrooms**: total bedrooms in the district
- **population**: total population in the district
- **households**: total households in the district
- **median_income**: median income of households in the district
- **median_house_value**: median value of housing in the district
- **ocean_proximity**: distance to the ocean

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

#2. Get the data

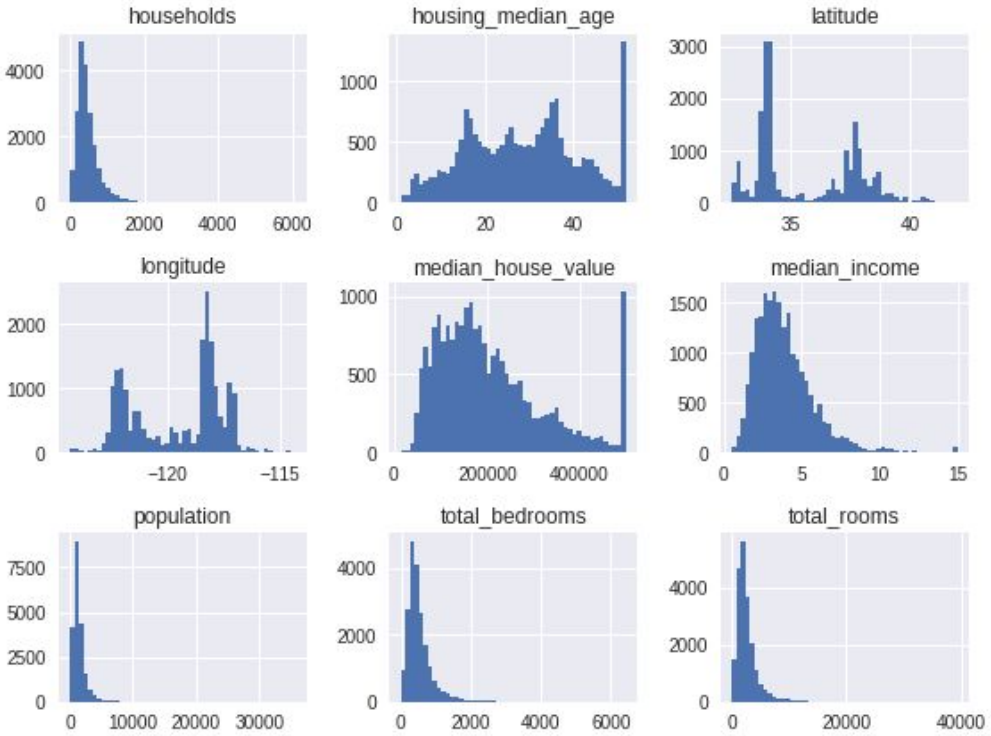
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```
# You can find out what categories exist and
# how many districts belong to each category
# by using the value_counts()

data.ocean_proximity.value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

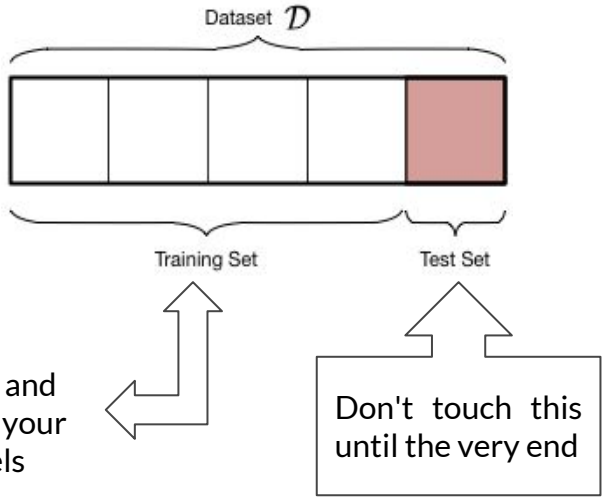
Histogram as an EDA tool




```
import matplotlib.pyplot as plt
data.hist(bins=50, figsize=(8,6))
plt.tight_layout()
plt.show()
```

1. The median income attribute does not look like it is expressed in US dollars (USD)
2. The housing median age and the median house value were also capped
3. Different scales
4. Many histograms are tail heavy

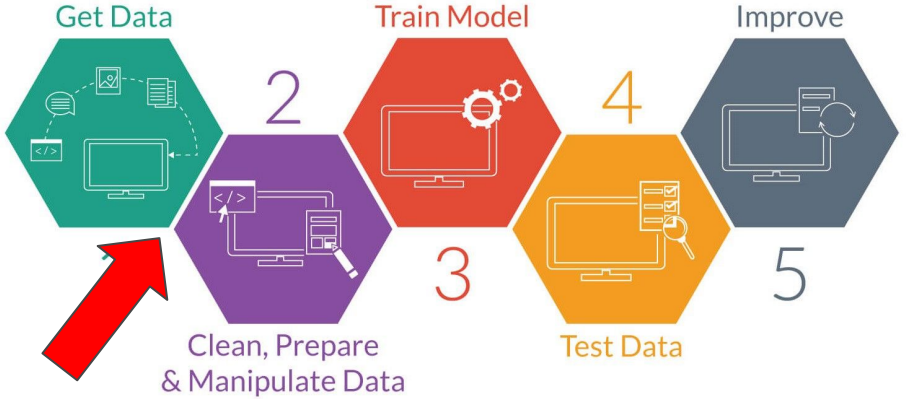
Create a Train and Test sets




```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(data,
                                       test_size=0.2,
                                       random_state=35)

print("data has {} atributes\n {} train instances\n {} test intances".
      format(len(data),len(train_set),len(test_set)))
```

```
data has 20640 atributes
 16512 train instances
 4128 test intances
```

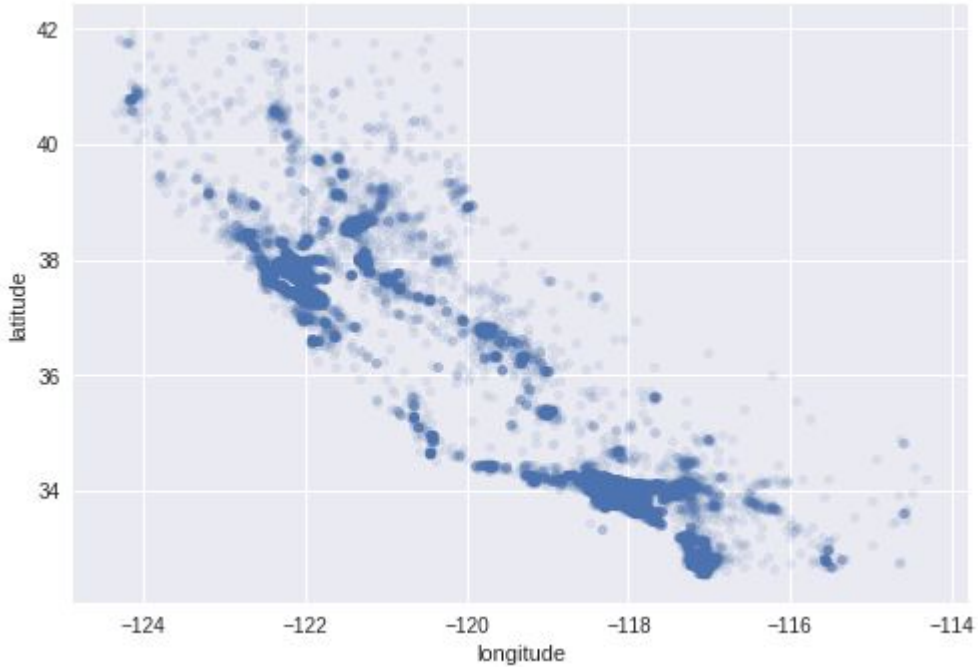
End-to-end ML Project



1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine. Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

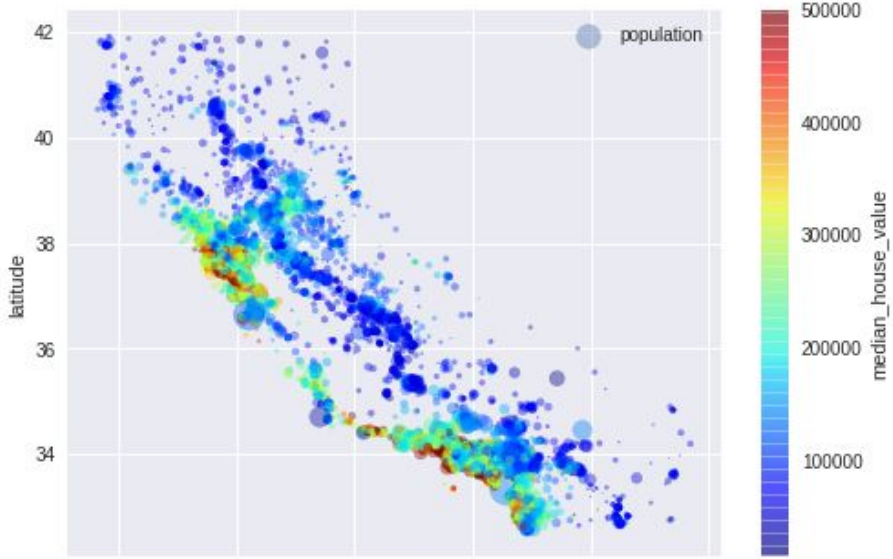
#3 Discovery and visualize the Data to Gain Insight

```
train.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

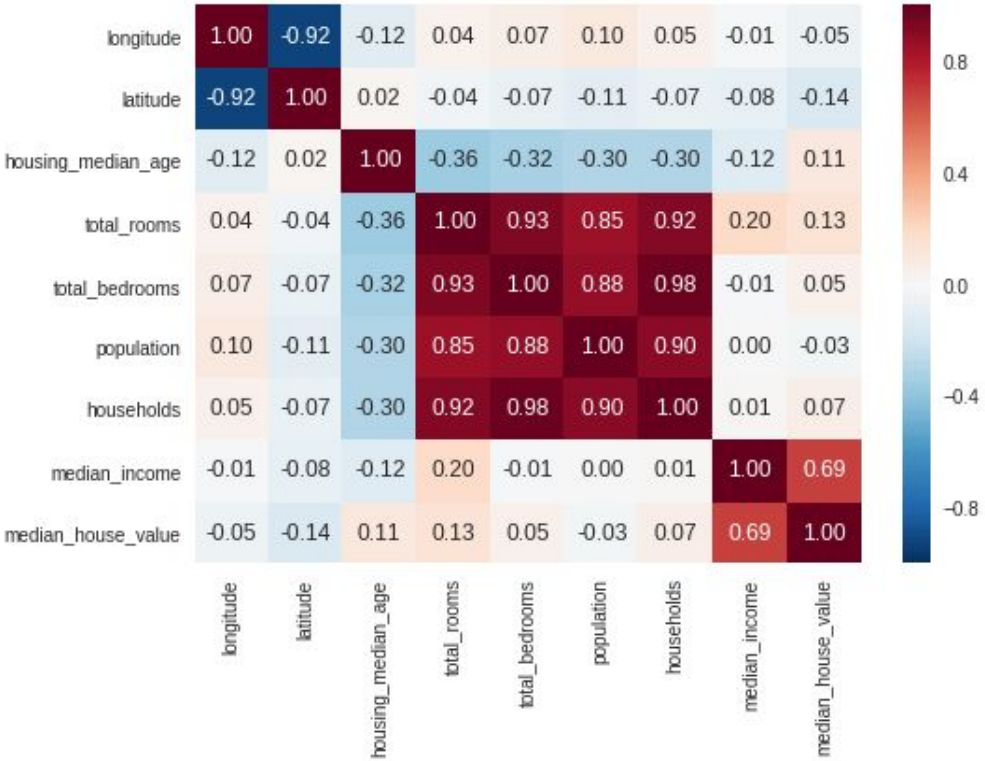


Visualizing geographical data (plus)

```
train.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
           s=train["population"]/100, label="population",
           c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True)
plt.legend()
```



Looking for correlations




```
import seaborn as sns
sns.heatmap(train.corr(),
            annot=True, fmt=".2f")
```

```
corr_matrix = train.corr()
corr_matrix["median_house_value"].\
  sort_values(ascending=False)
```

```
median_house_value          1.000000
median_income               0.687109
rooms_per_household         0.145460
total_rooms                 0.132943
housing_median_age          0.106175
households                  0.066714
total_bedrooms              0.051019
population                 -0.026685
population_per_household   -0.027255
longitude                  -0.047650
latitude                   -0.142797
bedrooms_per_room          -0.249959
Name: median_house_value, dtype: float64
```

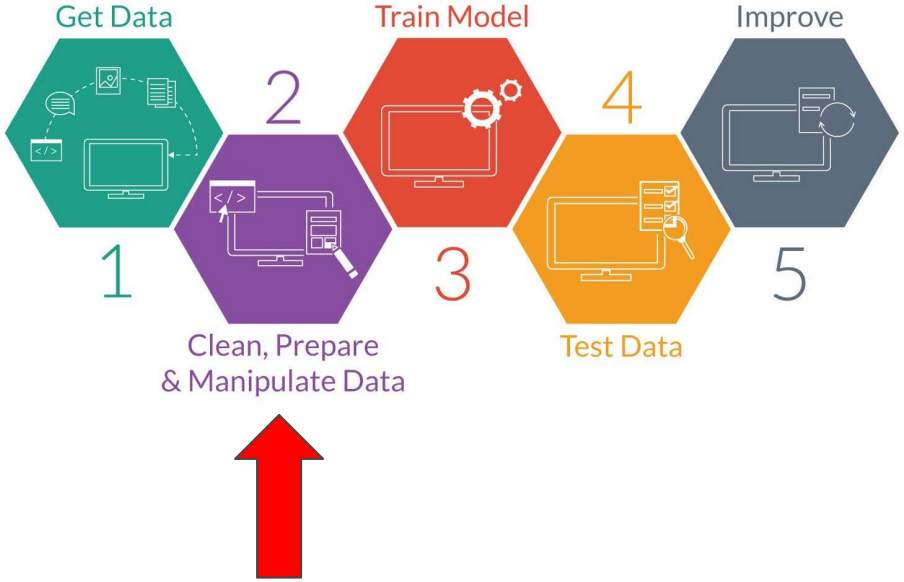
Experimenting with attributes combinations

```
# attribute combinations
train["rooms_per_household"] = train["total_rooms"]/train["households"]
train["bedrooms_per_room"] = train["total_bedrooms"]/train["total_rooms"]
train["population_per_household"]=train["population"]/train["households"]
```

```
train_correlation_matrix = train.corr()
train_correlation_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value          1.000000
median_income               0.687109
rooms_per_household         0.145460
total_rooms                 0.132943
housing_median_age          0.106175
households                  0.066714
total_bedrooms              0.051019
population                 -0.026685
population_per_household   -0.027255
longitude                  -0.047650
latitude                   -0.142797
bedrooms_per_room          -0.249959
Name: median_house_value, dtype: float64
```

End-to-end ML Project



1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine. Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

#4 Prepare the data for ML Algorithms

```
# drop creates a copy of the remain data and does not affect train_set
train_X = train_set.drop("median_house_value", axis=1)

# copy the label (y) from train_set
train_y = train_set.median_house_value.copy()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Data Cleaning

```
# count the number of missing values
train_X.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      166
population            0
households            0
median_income         0
ocean_proximity       0
dtype: int64
```

You have three options:

1. Get rid of the corresponding districts.
2. Get rid of the whole attribute.
3. Set the values to some value (zero, the mean, the median, etc.).

You can accomplish these easily using DataFrame's **dropna()**, **drop()**, and **fillna()** methods:

```
train_X.dropna(subset=["total_bedrooms"]) # option 1
train_X.drop("total_bedrooms", axis=1) # option 2
median = train_X["total_bedrooms"].median()
train_X["total_bedrooms"].fillna(median) # option 3
```

Handling with missing values

Step #1 - Fit

```
# First, you need to create an Imputer instance, specifying that you want
# to replace each attribute's missing values with the median of that attribute:
from sklearn.preprocessing import Imputer
imputer = Imputer(strategy="median")

# Since the median can only be computed on numerical attributes, we need to
# create a copy of the data without the text attribute ocean_proximity:
train_X_num = train_X.drop("ocean_proximity", axis=1)

# Now you can fit the imputer instance to the training data using
# the fit() method:
imputer.fit(train_X_num)
```

Step #2 - Transforme

```
# Now you can use this "trained" imputer to transform the training set by
# replacing missing values by the learned medians:
train_X_num_array = imputer.transform(train_X_num)
```

Handling with missing values

Before

```
# count the number of missing values
train_X.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      166
population            0
households            0
median_income         0
ocean_proximity       0
dtype: int64
```

After

```
train_X_num_df.isnull().sum()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
dtype: int64
```

Handling Text and Categorical Attributes

```
train_X.ocean_proximity.head(10)
```

```
1380        NEAR BAY
12294         INLAND
7387        <1H OCEAN
14454     NEAR OCEAN
2927          INLAND
12462         INLAND
19813         INLAND
11229       <1H OCEAN
16696       <1H OCEAN
13564         INLAND
Name: ocean_proximity, dtype: object
```

```
# For this, we can use Pandas' factorize() method which maps each
# category to a different integer:

train_X_cat_encoded, train_X_categories = train_X.ocean_proximity.factorize()

# train_X_cat_encoded is now purely numerical
train_X_cat_encoded[0:10]
```

```
array([0, 1, 2, 3, 1, 1, 1, 2, 2, 1])
```

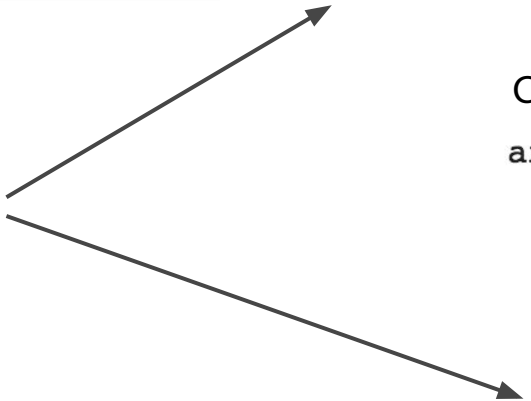
One issue with this representation is that ML algorithms will assume higher the categorical value, better the category. “Wait, What!?”.

Handling Text and Categorical Attributes

Solutions

```
train_X.ocean_proximity.head(10)
```

```
1380        NEAR BAY
12294         INLAND
7387        <1H OCEAN
14454     NEAR OCEAN
2927          INLAND
12462         INLAND
19813         INLAND
11229       <1H OCEAN
16696       <1H OCEAN
13564         INLAND
Name: ocean_proximity, dtype: object
```



maps each category to a different integer

```
array([0, 1, 2, 3, 1, 1, 1, 2, 2, 1])
```

Create a binary attribute per category

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 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.]])
```

OneHotEnconder

Handling Text and Categorical Attributes

Solutions

```
# Scikit-Learn provides a OneHotEncoder encoder to convert
# integer categorical values into one-hot vectors.

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()

# Numpy's reshape() allows one dimension to be -1, which means "unspecified":
# the value is inferred from the lenght of the array and the remaining
# dimensions
train_X_cat_1hot = encoder.fit_transform(train_X_cat_encoded.reshape(-1,1))

# it is a column vector
train_X_cat_1hot
```

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

```
Using a sparse matrix: 660592 bytes
Using a dense numpy array: 56 bytes
```

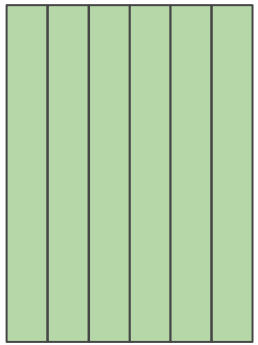
Custom Transformers

Although Scikit-Learn provides many useful transformers, **you will need to write your own for tasks such as custom cleanup operations or combining specific attributes**.

You will want your transformer to work seamlessly with Scikit-Learn functionalities. All you need is to create a class and implement three methods:

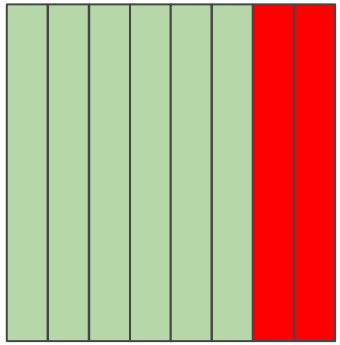
- fit()
- transform()
- fit_transform()

Train data

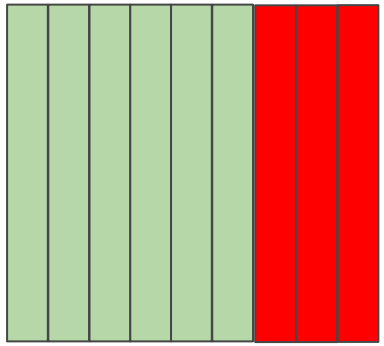


Custom Transformer for combine attribute

Train data



Train data



```
import numpy as np

# these classes will be useful later for automatic hyperparameter tuning
from sklearn.base import BaseEstimator, TransformerMixin

# indices for the columns
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6


class CombinedAttributesAdder(BaseEstimator, TransformerMixin):

    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room

    def fit(self, X, y=None):
        return self # nothing else to do

    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            # Translates slice objects to concatenation along the second axis.
            return np.c_[X, rooms_per_household,
                         population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
train_X_extra_attribs = attr_adder.transform(train_X.values)
```

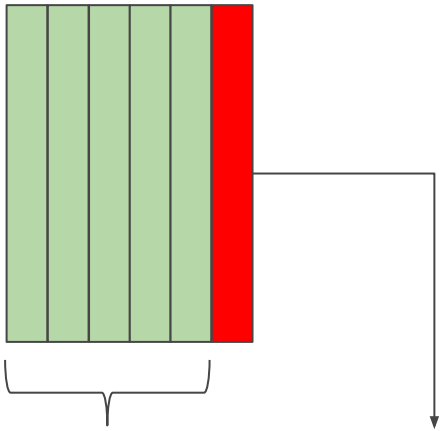
```
train_X.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity'],
      dtype='object')
```

```
train_X_extra_attribs_df.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity', 'rooms_per_household', 'population_per_household'],
      dtype='object')
```

Train data

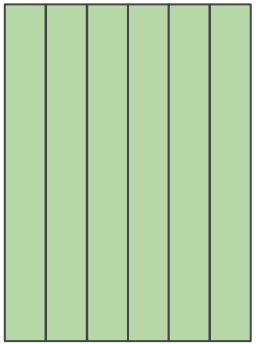


Numerical

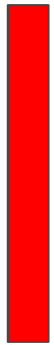
Categorical

Custom Transformer for select attributes

Numerical Attributes



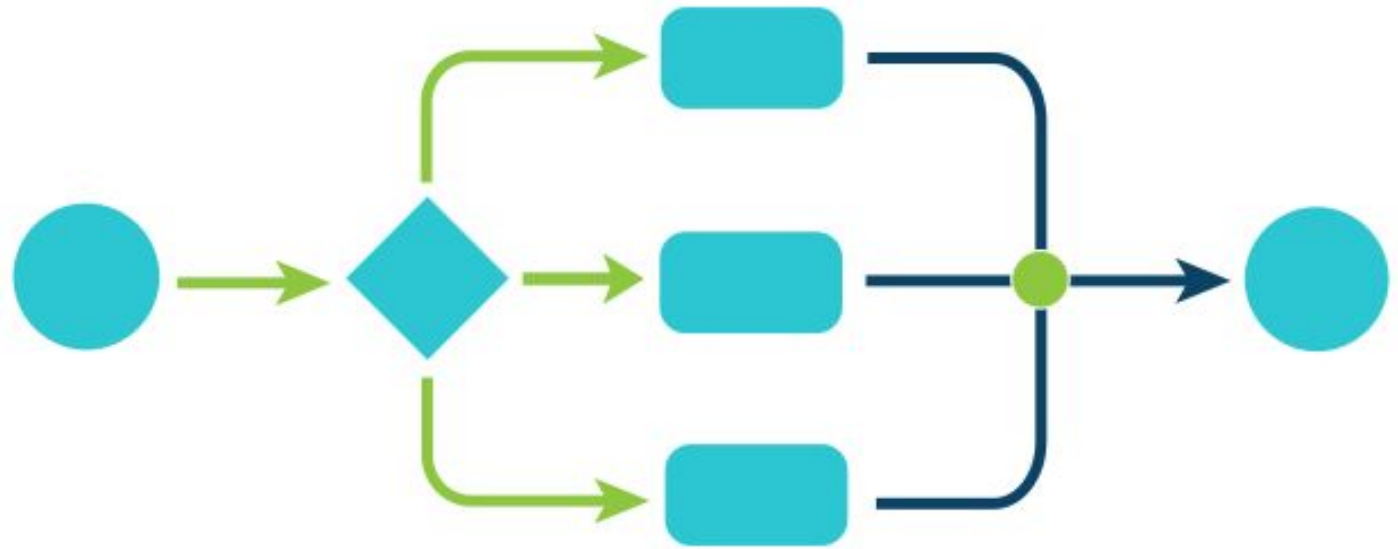
Categorical Attribute



There is nothing in Scikit-Learn to handle Pandas DataFrames, but we can write a custom transformer for this task

```
from sklearn.base import BaseEstimator, TransformerMixin

# This class will transform the data by selecting the desired attributes,
# dropping the rest, and converting the resulting DataFrame to a NumPy array.
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names

  def fit(self, X, y=None):
    return self

  def transform(self, X):
    return X[self.attribute_names].values
```

Transformation Pipelines



Transformation Pipelines

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler


num_pipeline = Pipeline([('imputer', Imputer(strategy="median")),
                         ('attribs_adder', CombinedAttributesAdder()),
                         ('std_scaler', StandardScaler())
                        ])
train_X_num_pipeline = num_pipeline.fit_transform(train_X_num)
```

```
# Used to join two or more pipelines into a single pipeline
from sklearn.pipeline import FeatureUnion

# https://github.com/scikit-learn/scikit-learn/issues/10521
from future_encoders import OneHotEncoder

# numerical columns
num_attribs = list(train_X_num.columns)

# categorical columns
cat_attribs = ["ocean_proximity"]

# pipeline for numerical columns
num_pipeline = Pipeline([('selector', DataFrameSelector(num_attribs)),
                         ('imputer', Imputer(strategy="median")),
                         ('attribs_adder', CombinedAttributesAdder()),
                         ('std_scaler', StandardScaler())
                        ])

# pipeline for categorical column
cat_pipeline = Pipeline([('selector', DataFrameSelector(cat_attribs)),
                         ('cat_encoder', OneHotEncoder(sparse=False))
                        ])

# a full pipeline handling both numerical and categorical attributes
full_pipeline = FeatureUnion(transformer_list=[("num_pipeline", num_pipeline),
                                               ("cat_pipeline", cat_pipeline)
                                              ])
```

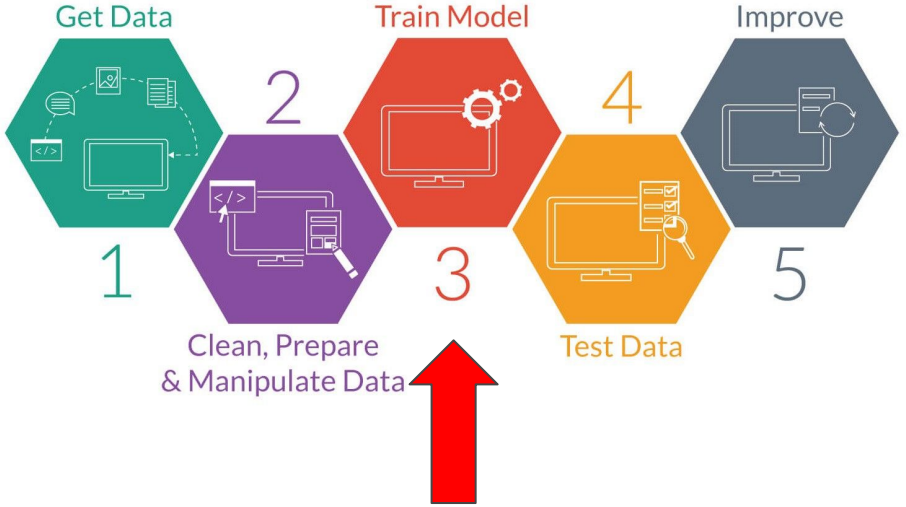
```
# you can run the whole pipeline simply
train_X_prepared = full_pipeline.fit_transform(train_X)
train_X_prepared
```

```
array([[-1.25390838,  1.10757958, -1.79230424, ...,  0.        ,
         1.        ,  0.        ],
       [ 1.31649014, -0.79863258, -1.2367069 , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.65894633, -0.77989831,  0.66819829, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.80340671, -0.54103634,  0.27134305, ...,  0.        ,
         0.        ,  0.        ],
       [-1.12937357,  0.78909696, -0.52236745, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.62407658, -0.67217624,  0.58882724, ...,  0.        ,
         0.        ,  0.        ]])
```

```
train_X_prepared.shape
```

```
(16512, 16)
```

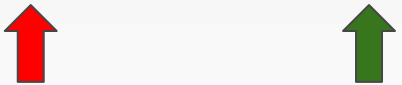
End-to-end ML Project



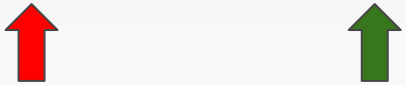
1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine. Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

Select a model and train it

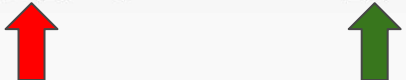
```
from sklearn.linear_model import LinearRegression

# create a LinearRegression model
lin_reg = LinearRegression()

# fit it
lin_reg.fit(train_X_prepared, train_y)
```



```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(train_X_prepared, train_y)
```



```
from sklearn.ensemble import RandomForestRegressor

# create a RandomForestRegressor model
forest_reg = RandomForestRegressor()

# fit it
forest_reg.fit(train_X_prepared, train_y)
```



Evaluate the training

Select data

```
# Done!! You now have a working Linear Regression Model.
# Let's try it out on a few instances from the trainning set.

# prepare the data
some_data = train_X.iloc[:5]
some_labels = train_y.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

# make predictions
print("Predictions:", lin_reg.predict(some_data_prepared))
```

```
Predictions: [296903.77937561 117785.90822548 162198.49440035 256500.66710587
  58859.67211795]
```

```
# Compare against the actual values:
print("Labels:", list(some_labels))
```

```
Labels: [204200.0, 95600.0, 112000.0, 410000.0, 51600.0]
```

Using RMSE to evaluate your model

Linear Regression

```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(train_X_prepared)
lin_mse = mean_squared_error(train_y, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
68089.48048082175
```

Using RMSE to evaluate your model

Decision Tree Regressor

```
# now that the model is trained, let's evaluate it on the training set

housing_predictions = tree_reg.predict(some_data_prepared)
tree_mse = mean_squared_error(some_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
0.0
```

OVERFITTING

Better Evaluation using Cross-Validation

```
lin_reg = LinearRegression()

lin_scores = cross_val_score(lin_reg,
                             train_X_prepared,
                             train_y,
                             scoring="neg_mean_squared_error",
                             cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
```

```
forest_reg = RandomForestRegressor()

forest_scores = cross_val_score(forest_reg,
                                train_X_prepared,
                                train_y,
                                scoring="neg_mean_squared_error",
                                cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
```

```
tree_reg = DecisionTreeRegressor()

scores = cross_val_score(tree_reg,
                         train_X_prepared,
                         train_y,
                         scoring="neg_mean_squared_error",
                         cv=10)
rmse_scores = np.sqrt(-scores)
```

Linear Regressor

```
Scores: [70715.10105097 67073.39881848 70838.04099604 67554.17889268
 66899.49055729 65194.36521007 70433.39627599 71899.00507697
 65779.96671776 67532.51453746]
Mean: 68391.94581337104
Standard deviation: 2242.784819848661
```

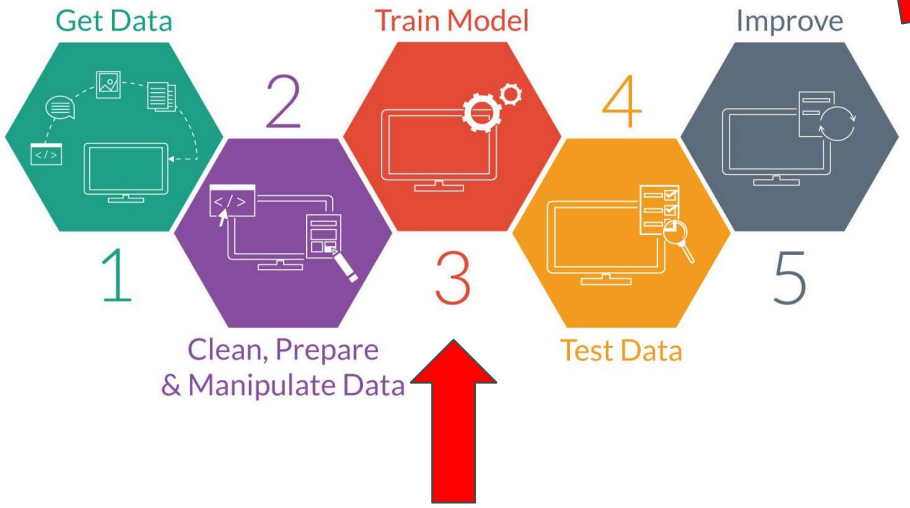
Decision Tree Regressor

```
Scores: [71300.9897682  72182.25329239 71729.47336888 68647.09566751
 66740.35856553 66433.14534197 72121.22641768 68921.44236274
 69672.01490628 70304.96502729]
Mean: 69805.29647184593
Standard deviation: 2002.7061155158112
```

Random Forest Regressor

```
Scores: [52545.50217667 52216.87963698 54502.66476615 52140.38783264
 49866.0822568  52189.88358181 53031.70342465 54294.35976526
 50748.84852618 51886.28180741]
Mean: 52342.25937745378
Standard deviation: 1339.8770651999557
```

End-to-end ML Project



1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine. Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

Grid Search with Cross Validation

```
from sklearn.model_selection import GridSearchCV

# hyperparameters values
# param_grid[0] - 12 combinations
# param_grid[1] - 6 combinations
param_grid = [{'n_estimators': [3, 10, 30],
               'max_features': [2, 4, 6, 8]
              },
              {'bootstrap': [False],
               'n_estimators': [3, 10],
               'max_features': [2, 3, 4]
              }
             ]

# create a randomforeestregressor model
forest_reg = RandomForestRegressor()

# run the grid search with cross validation
# (12 + 6) x 5 = 90 combinations
grid_search = GridSearchCV(forest_reg,
                           param_grid,
                           cv=5,
                           scoring='neg_mean_squared_error')

# see 90 combinations!!!
# it may take quite a long time
grid_search.fit(train_X_prepared, train_y)
```

Best combination of parameters

```
# when gridsearch is done you can get the best combination of parameters
grid_search.best_params_
```

```
{'max_features': 6, 'n_estimators': 30}
```

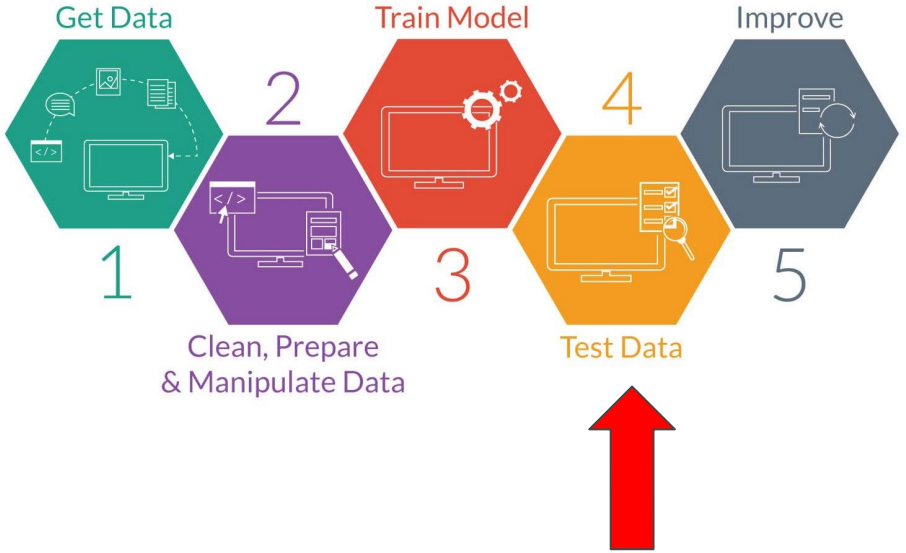
```
64252.163256566375 {'max_features': 2, 'n_estimators': 3}
55086.57916703393 {'max_features': 2, 'n_estimators': 10}
52260.28249700651 {'max_features': 2, 'n_estimators': 30}
59511.89329239456 {'max_features': 4, 'n_estimators': 3}
52226.15721344689 {'max_features': 4, 'n_estimators': 10}
49896.473307367676 {'max_features': 4, 'n_estimators': 30}
58786.81347625304 {'max_features': 6, 'n_estimators': 3}
51740.43037878579 {'max_features': 6, 'n_estimators': 10}
49404.62639980429 {'max_features': 6, 'n_estimators': 30}
58704.30259452863 {'max_features': 8, 'n_estimators': 3}
51854.336241117635 {'max_features': 8, 'n_estimators': 10}
49667.66166853269 {'max_features': 8, 'n_estimators': 30}
61834.61250169976 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54011.37659626647 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59803.93142565374 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52357.09768508134 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57407.46795177193 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51393.59047933123 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Understanding the importance of features

```
# can indicate the relative importance of each attribute
# for making accurate predictions
feature_importances = grid_search.best_estimator_.feature_importances_
```

```
[(0.2932240517215898, 'median_income'),
 (0.16849927912474488, 'INLAND'),
 (0.1129734422811391, 'pop_per_hhold'),
 (0.08350531964855805, 'bedrooms_per_room'),
 (0.0729308899867341, 'longitude'),
 (0.07283949940912572, 'rooms_per_hhold'),
 (0.06882020855312361, 'latitude'),
 (0.04330022841532853, 'housing_median_age'),
 (0.018220886556910475, 'population'),
 (0.017270129640924746, 'total_rooms'),
 (0.016563758506314884, 'total_bedrooms'),
 (0.01601840705660779, 'households'),
 (0.008515031117812108, '<1H OCEAN'),
 (0.004777746659219996, 'NEAR OCEAN'),
 (0.0024000797226816526, 'NEAR BAY'),
 (0.0001410415991847045, 'ISLAND')]
```

End-to-end ML Project



1. Look at the big picture.
2. Get the data.
3. Discover and visualize the data. to gain insights.
4. Prepare the data for Machine. Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.

Evaluate the final model on the Test Set

```
# best model found in gridsearch step
final_model = grid_search.best_estimator_

# predictors and label
test_X = test_set.drop("median_house_value", axis=1)
test_y = test_set["median_house_value"].copy()

# prepared test's predictors
test_X_prepared = full_pipeline.transform(test_X)


final_predictions = final_model.predict(test_X_prepared)
final_mse = mean_squared_error(test_y, final_predictions)
final_rmse = np.sqrt(final_mse)
print(final_rmse)
```

```
49964.127058988925
```

Week Assignments

1. Lessons 3, 4 and 5
   a. https://github.com/ivanovitchm/EEC2006
2. Choose a regression problem and try to reproduce the same steps as described earlier.
   a. https://github.com/ShuaiW/kaggle-regression
   b. https://www.kaggle.com/rtatman/datasets-for-regression-analysis
3. Read Chapter #1
   a. https://www.manning.com/books/deep-learning-with-python