Machine Learning Introduction

Lecturer: Ngo Huy Bien
Software Engineering Department
Faculty of Information Technology
University of Science
Ho Chi Minh City, Vietnam
nhbien@fit.hcmus.edu.vn

Objectives

- To *differentiate* a traditional program from a machine learning program
- To present *why* learn machine learning
- To write a *Hello World* machine learning program
- To explain machine learning *terminologies* via examples

Contents

I. "Spam Filter" example
II. "Does Money Make People Happier?" example
III. Machine learning terminologies

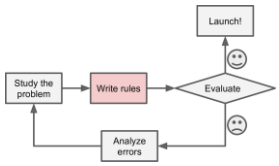
References

1. Aurelien Geron (2017). Hands on Machine Learning with Scikit Learn and TensorFlow. O'Reilly Media.
2. Francois Chollet (2018). Deep Learning with Python. Manning Publications Co.
3. Brett Lantz (2015). Machine Learning with R. 2nd Edition. Packt Publishing.

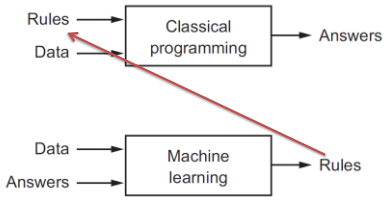
Spam Filter: The Traditional Approach [1]

1. First you would look at what spam typically looks like. You might notice that *some words or phrases* (such as "4U," "credit card," "free," and "amazing") tend to come up a lot in the subject.
2. You would write a detection algorithm for each of the patterns that you noticed, and your program would flag emails as spam if a number of these patterns are detected.
3. You would test your program, and repeat steps 1 and 2 until it is good enough.

Since the problem is not trivial, your program will likely become a long list of complex rules—pretty hard to maintain.

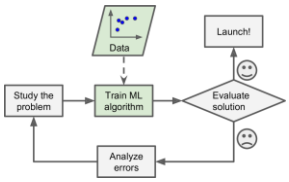


Machine Learning Approach [2]



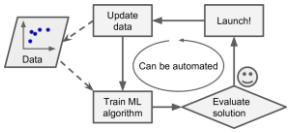
Spam Filter: Machine Learning Approach [1]

- A spam filter based on Machine Learning techniques *automatically learns* which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples.



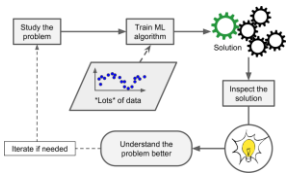
Automatically Adapting To Change

- If spammers notice that all their emails containing "4U" are blocked, they might start writing "For U" instead.
- A spam filter using traditional programming techniques would need *to be updated* to flag "For U" emails.
- In contrast, a spam filter based on Machine Learning techniques *automatically notices* that "For U" has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention.



Data Mining

- Once the spam filter has been trained on enough spam, it can easily be inspected to *reveal* the list of words and combinations of words that it believes are the best predictors of spam.
- Sometimes this will reveal *unsuspected correlations* or *new trends*, and thereby lead to a better understanding of the problem.
- Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent. This is called *data mining*.



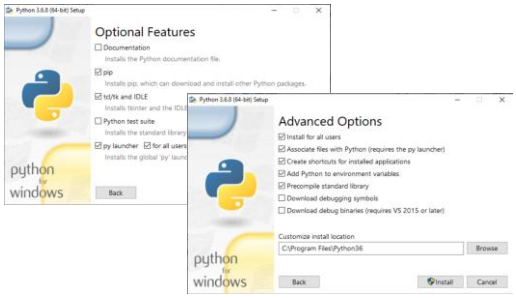
What ML Has Achieved So Far

- 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

Install Python [1]

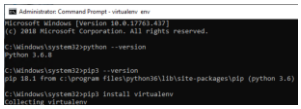
- **Note: As of now (01/2019) Tensorflow only supports 64-bit versions of Python 3.5.x and 3.6.x on Windows.**
- https://www.python.org/downloads/ (select "Install launcher for all users" and "Add Python 3.6 to PATH" to avoid path issues)

Installation Options



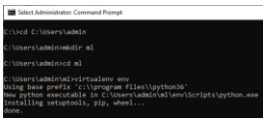
Install **`virtualenv`**

- Run **`cmd.exe`** as Administrator
- Verify installation: **`python --version`**
- **`pip3 --version`**
- **`pip3 install virtualenv`**
- Restart computer



Create An Isolated Environment

- Run **`cmd.exe`** as Administrator
- **`mkdir ml`**
- **`cd ml`**
- **`virtualenv env`**
  or
- **`virtualenv.py env`**



Activate The Environment

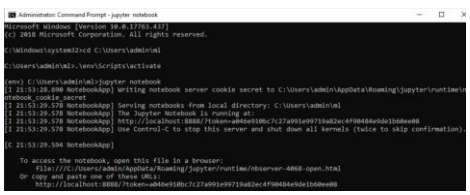
- Open PowerShell as Administrator
- **`Set-ExecutionPolicy -ExecutionPolicy "Unrestricted"`**
- **`cd C:\Users\admin\ml`**
- **`.\env\Scripts\activate`**

Install Modules

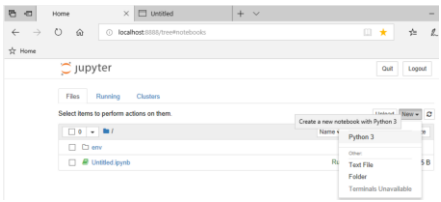
- **`pip3 install --upgrade jupyter matplotlib numpy pandas scipy scikit-learn tensorflow keras`**
- Press Enter if the screen seems not to respond.
- Rerun the command if any error occurs.
- Verify installation:
- **`python -c "import jupyter, matplotlib, numpy, pandas, scipy, sklearn, tensorflow, keras"`**

Run Jupiter Notebook

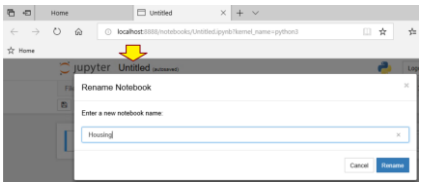
- **`cd C:\Users\admin\ml`**
- **`.\env\Scripts\activate`**
- **`jupyter notebook`**
- **`jupyter notebook --ip 192.168.1.19 --port 8888`**



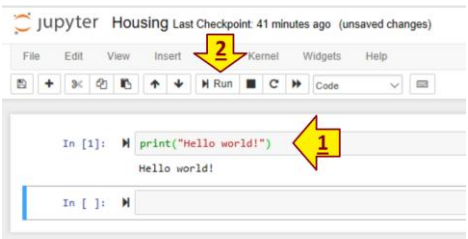
Create A New Notebook File



Rename The Notebook



Hello World!

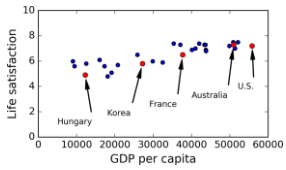


Colaboratory

- https://colab.research.google.com/notebooks/welcome.ipynb
- Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.
- With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

Does Money Make People Happier?

Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

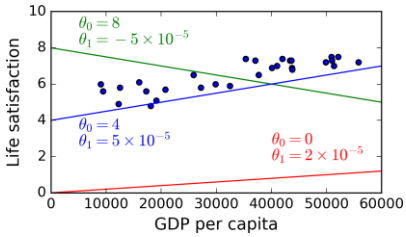


- **GDP per capita = 22587**
- **Life satisfaction = ?**

A Simple Linear Model

- So you decide to model life satisfaction as a *linear function* of GDP per capita.

  life_satisfaction = θ_0 + θ_1 × GDP_per_capita

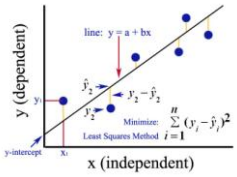


How To Find θ_0 And θ_1?

- Before you can use your model, you need to define the *parameter* values θ_0 and θ_1.
- How can you know which values will make your model perform best? To answer this question, you need to specify a *performance measure*.

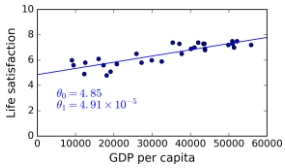
Cost Function

- You can either define a *utility function* (or *fitness function*) that measures how *good* your model is, or you can define a *cost function* that measures how *bad* it is.
- For linear regression problems, people typically use a cost function that measures the *distance* between the linear model's predictions and the training examples; the objective is to *minimize this distance*.



Training

- This is where the Linear Regression algorithm comes in: you feed it your training examples and it *finds the parameters* that make the linear model fit best to your data.
- This is called *training* the model.
- In our case the algorithm finds that the *optimal parameter values* are $\theta_0 = 4.85$ and $\theta_1 = 4.91 \times 10^{-5}$.



Loading Data

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
# Load the data
oecd_bli = pd.read_csv("./datasets/oecd_bli_2015.csv",
thousands=',')
gdp_per_capita =
pd.read_csv("./datasets/gdp_per_capita.csv",thousands=','
,delimiter='\t',
encoding='latin1', na_values="n/a")
```

Preparing The Data

```
# Prepare the data
def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator",
values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP per capita"},
inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                  left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    remove_indices = [0, 1, 6, 8, 33, 34, 35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP per capita", 'Life
satisfaction']].iloc[keep_indices]

country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
```

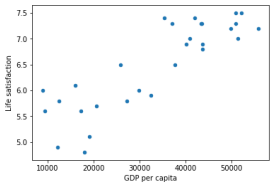
Visualizing The Data (I)

```
country_stats
```

Country	GDP per capita	Life satisfaction
Russia	9054.914	6.0
Turkey	9437.372	5.6
Hungary	12239.894	4.9
Poland	12495.334	5.8
Slovak Republic	15991.736	6.1
Estonia	17288.083	5.6
Greece	18064.288	4.8
Portugal	19121.592	5.1
Slovenia	20732.482	5.7
Spain	25864.721	6.5
Korea	27195.197	5.8

Visualizing The Data (II)

```
# Visualize the data
country_stats.plot(kind='scatter', x="GDP per capita",
y='Life satisfaction')
plt.show()
```



Preparing Training Data

```
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]
print ('X shape:', X.shape)
print ('y shape:', y.shape)
print('X[0]:', X[0])
print('y[0]:', y[0])
```

```
X shape: (29, 1)
y shape: (29, 1)
X[0]: [9054.914]
y[0]: [6.]
```

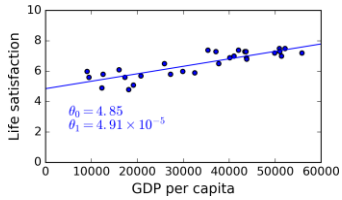
Training A Model

```
from sklearn.linear_model import
LinearRegression
# Select a linear model
lin_reg_model = LinearRegression()
# Train the model
lin_reg_model.fit(X, y)
print ('coef:', lin_reg_model.coef_)
print ('intercept:', lin_reg_model.intercept_)
```

```
coef: [[4.91154459e-05]]
intercept: [4.8530528]
```

Making A Prediction For Cyprus

```
X_new = [[22587]] # Cyprus' GDP per capita
print(lin_reg_model.predict(X_new)) # outputs
[[ 5.96242338]]
```

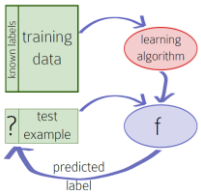


Terminologies

- Training set, Training data
- Test set, Test data
- Examples, Instances
- Features
- Labels, Targets
- Model

Summary

- You *studied* the data.
- You selected a *model*.
- You trained it on the training data (i.e., the learning algorithm searched for the *model parameter values* that minimize a *cost function*).
- Finally, you applied the model to *make predictions* on new cases (this is called *inference*), hoping that this model will generalize well.



What Is Machine Learning?

- Machine Learning is the science (and art) of programming computers so they can *learn from data*.
- 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.

—Tom Mitchell, 1997

When To Use Machine Learning?

- Problems for which existing solutions require a lot of hand-tuning or *long lists of rules*: one Machine Learning algorithm can often simplify code and perform better.
- *Complex problems* for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.
- *Fluctuating environments*: a Machine Learning system can adapt to new data.
- Getting *insights* about complex problems and large amounts of data.

Data Challenge

- *Insufficient quantity* of training data
- *Nonrepresentative* training data:
  - if the sample is too small, you will have *sampling noise* (i.e., nonrepresentative data as a result of chance),
  - but even very large samples can be nonrepresentative if the sampling method is flawed. This is called *sampling bias*.
- *Poor-quality* data

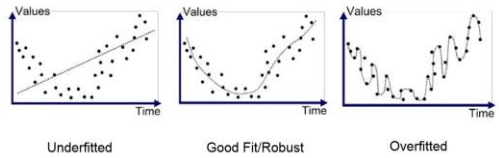
Irrelevant Features

- As the saying goes: *garbage in*, *garbage out*.
- Your system will only be capable of learning if the training data contains enough *relevant features* and not too many irrelevant ones.

- *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.

Overfitting The Training Data

- It means that the model performs well on the training data, but it *does not generalize* well.
- Overfitting happens when the *model is too complex* relative to the amount and noisiness of the training data.



Overfitting: Solutions

- The *possible solutions* are:
  - To *simplify the model* by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model
  - To gather *more training data*
  - To *reduce the noise* in the training data (e.g., fix data errors and remove outliers)

Model Simplification: Regularization

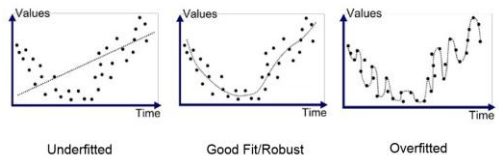
- Constraining a model to make it simpler and reduce the risk of overfitting is called *regularization*.
- For example,
  - the linear model we defined earlier has *two parameters*, θ_0 and θ_1.
  - This gives the learning algorithm two degrees of freedom to adapt the model to the training data: it can *tweak both the height* (θ_0) and *the slope* (θ_1) of the line.
  - If we forced $\theta_1 = 0$, the algorithm would have only *one degree of freedom* and would have a much harder time fitting the data properly: all it could do is move the line up or down to get as close as possible to the training instances, so it would end up around the mean.

Hyperparameter

- The *amount of regularization* to apply during learning can be controlled by a hyperparameter.
- A *hyperparameter* is a parameter of a learning algorithm (not of the model).
- As such, it is *not affected* by the learning algorithm itself; it must be set prior to training and remains constant during training.
- If you set the regularization hyperparameter to a very large value, you will get an almost flat model (a slope close to zero); the learning algorithm will almost certainly not overfit the training data, but it will be less likely to find a good solution.

Underfitting The Training Data

- It occurs when your *model is too simple* to learn the underlying structure of the data.

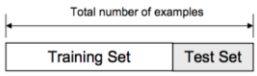


Underfitting: Solutions

- The main *options* to fix this problem are:
  - Selecting a *more powerful model*, with more parameters
  - Feeding *better features* to the learning algorithm (feature engineering)
  - *Reducing the constraints* on the model (e.g., reducing the regularization hyperparameter)

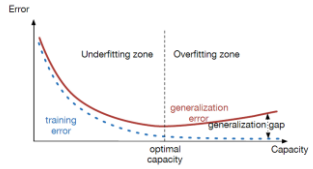
Testing

- 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. This works well, but if your model is horribly bad, your users will complain—not the best idea.
- A better option is to split your data into two sets: the *training set* and the *test set*. As these names imply, you train your model using the training set, and you test it using the test set. It is common to use 80% of the data for training and hold out 20% for testing.



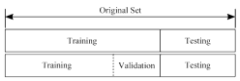
Generalization Error

- The *error rate on new cases* is called the *generalization error* (or out-of-sample error), and by evaluating your model on the test set, you get an estimation of this error.
- If the *training error is low* (i.e., your model makes few mistakes on the training set) but *the generalization error is high*, it means that your model is overfitting the training data.



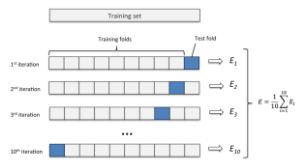
Validating

- Now suppose you are hesitating between *two models* (say a linear model and a polynomial model): how can you decide?
- One option is to train both and compare how well they generalize *using the test set*. Now suppose that the linear model generalizes better, but you want to *apply some regularization* to avoid overfitting.
- One option is to train 100 different models using 100 different values for this hyperparameter.
- The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that set.
- A common solution to this problem is to have a second holdout set called the *validation set*.



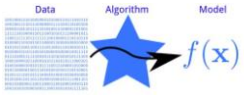
Cross-Validation

- To avoid "wasting" too much training data in validation sets, a common technique is to use *cross-validation*: the training set is split into complementary subsets, and each model is trained against a different combination of these subsets and validated against the remaining parts.
- Once the model type and hyperparameters have been selected, a final model is trained using these hyperparameters on the *full training set*, and the generalized error is measured on the *test set*.



No Free Lunch Theorem

- A *model* is a simplified version of the observations.
- The *simplifications* are meant to discard the superfluous details that are unlikely to generalize to new instances.
- However, to decide what data to discard and what data to keep, you must make *assumptions*.
- For some datasets the best model is a linear model, while for other datasets it is a neural network.
- There is no model that is a *priori* guaranteed to work better.
- The only way to know for sure which model is best is to *evaluate them all*.
- Since this is *not possible*, in practice you make some reasonable assumptions about the data and you *evaluate* only a few reasonable models.



Supervised Learning

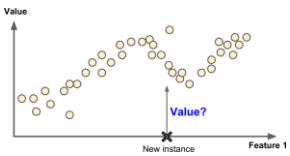
- In *supervised learning*, the training data you feed to the algorithm includes the desired solutions, called *labels*.
- A typical supervised learning task is *classification*.
- The spam filter is a good example of this: it is trained with many example emails along with their *class* (spam or ham), and it must learn how to *classify new emails*.

Regression

- 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*.
- This sort of task is called *regression*.
- To train the system, you need to give it many examples of cars, including both their predictors and their *labels* (i.e., their prices).

Attribute vs. Feature

- In Machine Learning an *attribute* is a data type (e.g., "Mileage"), while a *feature* has several meanings depending on the context, but generally means an attribute plus its value (e.g., "Mileage =15,000").
- Many people use the words attribute and feature interchangeably, though.



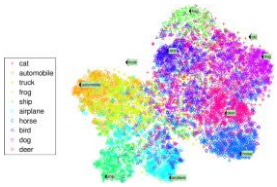
Unsupervised Learning

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

- For example, say you have a lot of data about your blog's visitors. You may want to run a *clustering* algorithm to try to detect groups of similar visitors.

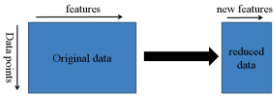
Visualization Algorithms

- *Visualization* algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted.



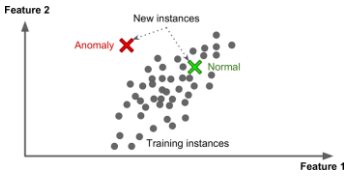
Dimensionality Reduction

- A related task is *dimensionality reduction*, in which the goal is to simplify the data without losing too much information.
- One way to do this is to *merge several correlated features* into one.
- For example, a car's mileage may be very correlated with its age, so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear. This is called *feature extraction*.



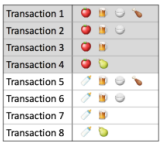
Anomaly Detection

- Yet another important unsupervised task is *anomaly detection*—for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm.
- The system is trained with *normal instances*, and when it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly.



Association Rule Learning

- Finally, another common unsupervised task is *association rule learning*, in which the goal is to dig into large amounts of data and discover interesting relations between attributes.
- For example, suppose you own a supermarket. Running an association rule on your sales logs may reveal that people who purchase barbecue sauce and potato chips also tend to buy steak.
- Thus, you may want to place these items close to each other.

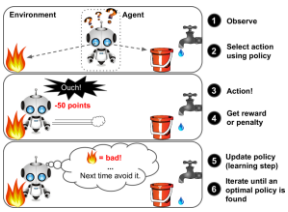


Semisupervised Learning

- Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called *semisupervised learning*.
- Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the *unsupervised part* of the algorithm (*clustering*).
- Now all the system needs is for you to tell it who these people are. Just *one label per person*, and it is able to name everyone in every photo, which is useful for searching photos.
- Most *semisupervised learning algorithms* are combinations of unsupervised and supervised algorithms.

Reinforcement Learning

- The learning system, called an *agent* in this context, can observe the environment, select and perform actions, and get *rewards* in return (or *penalties* in the form of negative rewards).
- It must then learn by itself what is the best strategy, called a *policy*, to get the most reward over time.
- A policy defines what action the agent should choose when it is in a given situation.



Robots & AlphaGo

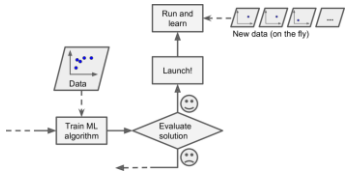
- For example, many *robots* implement Reinforcement Learning algorithms to learn how to walk.
- DeepMind's AlphaGo program is also a good example of Reinforcement Learning: it made the headlines in March 2016 when it beat the world champion Lee Sedol at the game of Go.
- It learned its *winning policy* by analyzing millions of games, and then playing many games against itself.
- Note that learning was turned off during the games against the champion; AlphaGo was just applying the policy it had learned.

Batch Learning

- In *batch learning*, the system is incapable of learning incrementally: it must be trained using all the available data.
- This will generally take a lot of time and computing resources, so it is typically done offline.
- First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned.
- This is called *offline learning*.
- If the amount of data is huge, it may even be impossible to use a batch learning algorithm.

Online Learning

- In *online learning*, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches.
- Each learning step is fast and cheap, so the system can learn about new data *on the fly*, as it arrives.



Online Learning: Learning Rate

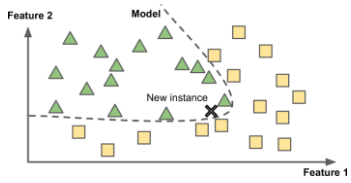
- One important parameter of online learning systems is how fast they should adapt to changing data: this is called the *learning rate*.
- If you set a *high learning rate*, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data (you don't want a spam filter to flag only the latest kinds of spam it was shown).
- Conversely, if you set a *low learning rate*, the system will have more inertia; that is, it will learn more slowly, but it will also be less sensitive to noise in the new data or to sequences of non-representative data points.

Instance-Based Learning

- Instead of just flagging emails that are identical to known spam emails, your spam filter could be programmed to also flag emails that are very similar to known spam emails.
- This requires a *measure of similarity* between two emails.
- A (very basic) similarity measure between two emails could be to count the number of words they have in common.
- The system would flag an email as spam if it has many words in common with a known spam email.
- This is called *instance-based learning*: the system learns the examples by heart, then generalizes to new cases using a similarity measure.

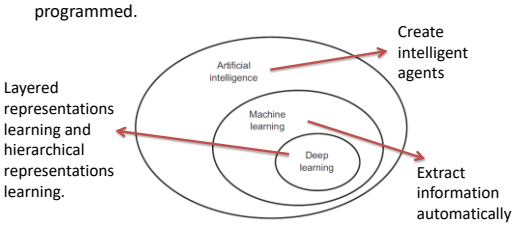
Model-Based Learning

- Another way to generalize from a set of examples is to *build a model* of these examples, then use that model to make *predictions*.
- This is called *model-based learning*.



Artificial Intelligence, Machine Learning, And Deep Learning [2]

- AI: The effort to *automate* intellectual tasks normally performed by humans.
- A machine-learning system is *trained* rather than explicitly programmed.



Where Does Data Come From? [3]

- *Astronomers* recorded patterns of planets and stars;
- *Biologists* noted results from experiments crossbreeding plants and animals; and
- *Cities* recorded tax payments, disease outbreaks, and populations.
- The invention of *electronic sensors* has additionally contributed to an increase in the richness of recorded data.
  - Specialized sensors see, hear, smell, or taste.
  - Weather sensors record temperature and pressure data, surveillance cameras watch sidewalks and subway tunnels, and all manner of electronic behaviors are monitored: transactions, communications, friendships, and many others.
  - Governments, businesses, and individuals are recording and reporting all manners of information from the monumental to the mundane.

Where Does Machine Learning Come From?

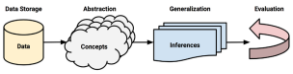
- The field of study interested in the development of computer algorithms for transforming data into intelligent action is known as *machine learning*.
- This field originated in an environment where the *available data*, *statistical methods*, and *computing power* rapidly and simultaneously evolved.

Machine Learning Successes

- Identification of unwanted *spam* messages in e-mail
- Segmentation of customer behavior for *targeted advertising*
- Forecasts of weather behavior and long-term *climate changes*
- Reduction of *fraudulent* credit card transactions
- Actuarial estimates of financial damage of storms and *natural disasters*
- Prediction of popular *election outcomes*
- Development of algorithms for *auto-piloting drones* and *self-driving cars*
- Optimization of *energy use* in homes and office buildings
- Projection of areas where *criminal activity* is most likely
- Discovery of genetic sequences linked to *diseases*

How Does A Machine Learn?

- A machine is said to learn if it is able to *take experience* and *utilize it* such that its performance improves up on similar experiences in the future.
- Regardless of whether the learner is a *human* or *machine*, the basic learning process is similar. It can be divided into *four interrelated components*:
  - *Data storage* utilizes observation, memory, and recall to provide a factual basis for further reasoning.
  - *Abstraction* involves the translation of stored data into broader representations and concepts.
  - *Generalization* uses abstracted data to create knowledge and inferences that drive action in new contexts.
  - *Evaluation* provides a feedback mechanism to measure the utility of learned knowledge and inform potential improvements.



Abstraction

- The observer's *mind* is able to connect the picture of a pipe to the idea of a pipe, to a memory of a physical pipe that could be held in the hand.

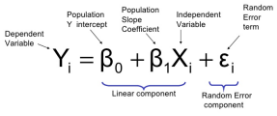
*A representation of a pipe is **not** truly a pipe.*

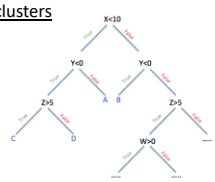
From Raw Data To Abstraction

- The data is merely *ones and zeros* on a disk or in memory; they have no meaning.
- This work of assigning meaning to stored data occurs during the *abstraction process*, in which raw data comes to have a more abstract meaning.
- During the process of knowledge representation, the computer summarizes raw inputs in a *model*, an explicit description of the structured patterns among data.

Knowledge Representation

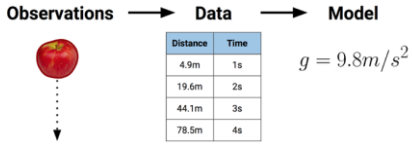
- There are many different *types of models*.
  - Equations
  - Diagrams such as trees and graphs
  - Logical if/else rules
  - Groupings of data known as clusters





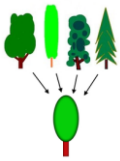
Training

- The process of fitting a particular model to a dataset is known as *training*.
- When the model has been trained, the data has been transformed into an *abstract form* that summarizes the original information.



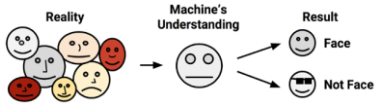
Generalization

- The term *generalization* describes the process of turning abstracted knowledge into a form that can be utilized for action.
- Generally, it is *not feasible* to reduce the number of potential concepts by examining them one-by-one and determining which are the most useful.
- Instead, *machine learning algorithms* generally employ shortcuts that more quickly divide the set of concepts. Toward this end, the algorithm will employ *heuristics*, or educated guesses about the where to find the most important concepts.



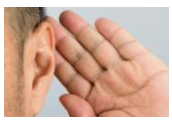
Bias

- The heuristics employed by machine learning algorithms also sometimes result in *erroneous conclusions*.
- If the conclusions are systematically imprecise, the algorithm is said to have a *bias*.
- A machine learning algorithm learned to identify faces by finding *two dark circles* representing eyes, positioned above *a straight line* indicating a mouth.



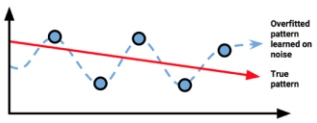
Noise

- The failure for models to perfectly generalize is due to the *problem of noise*, or unexplained variations in data.
- Noisy data is *caused by* seemingly random events, such as:
  - *Measurement error* due to imprecise sensors that sometimes add or subtract a bit from the reading
  - *Issues with reporting data*, such as respondents reporting random answers to survey questions in order to finish more quickly
  - Errors caused when *data is recorded incorrectly*, including missing, null, truncated, incorrectly coded, or corrupted values.



Over-fitting

- Trying to model the noise in data is the basis of a problem called *over-fitting*.
- A model that seems to perform well during training but does poorly during testing is said to be *over-fitted* to the training dataset as it does not generalize well.



Machine Learning In Practice

- Data collection
  - The data will need to be combined into a single source like a *text file*, *spreadsheet*, or *database*.
- Data exploration and preparation
  - *Fixing* or *cleaning* so-called "messy" data, eliminating unnecessary data, and recoding the data to conform to the learner's expected inputs.
- Model training
  - The selection of an appropriate *algorithm*, and the algorithm will represent the data in the form of *a model*.
- Model evaluation
- Model improvement

Examples and Features

- All machine learning algorithms require *input training data*.
- The exact format may differ, but in its most basic form, input data takes the form of *examples* and *features*.
- An *example* is literally a single exemplary instance of the underlying concept to be learned; it is one set of data describing the atomic unit of interest for the analysis.
- A *feature* is a characteristic or attribute of an example, which might be useful for learning the desired concept.

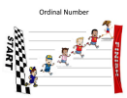
features

year	model	price	mileage	color	transmission
2011	SEL	21992	7413	Yellow	AUTO
2011	SEL	20995	10926	Gray	AUTO
2011	SEL	19995	7351	Silver	AUTO
2011	SEL	17809	11613	Gray	AUTO
2012	SE	17500	8367	White	MANUAL
2010	SEL	17495	25125	Silver	AUTO
2011	SEL	17000	27393	Blue	AUTO
2010	SEL	16995	21026	Silver	AUTO
2011	SES	16995	32655	Silver	AUTO

examples

Unit of Measurement

- The phrase *unit of observation* is used to describe the units that the examples are measured in.
  - Commonly, the unit of observation is in the form of transactions, persons, time points, geographic regions, or measurements.
- If a feature represents a characteristic measured in numbers, it is unsurprisingly called *numeric*.
- If it measures an attribute that is represented by a set of categories, the feature is called *categorical* or *nominal*.
- A special case of categorical variables is called *ordinal*, which designates a nominal variable with categories falling in an ordered list.

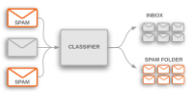


Predictive Models

- *A predictive model* is used for tasks that involve, as the name implies, the prediction of one value using other values in the dataset.
- The learning algorithm attempts to discover and model the relationship among the *target* feature (the feature being predicted) and the other features.
- Because predictive models are given clear instruction on what they need to learn and how they are intended to learn it, the process of training a predictive model is known as *supervised learning*.
- Stated more formally, given a set of data, a supervised learning algorithm attempts to optimize a function (the model) to find the combination of feature values that result in the target output.

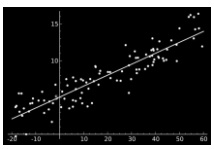
Classification

- The often used supervised machine learning task of predicting which category an example belongs to is known as *classification*.
  - A football team will win or lose.
  - A person will live past the age of 100.
  - An earthquake will strike next year.
- The target feature to be predicted is a categorical feature known as the *class* and is divided into categories called *levels*.
- A class can have two or more levels, and the levels need *not* necessarily be ordinal.



Numeric Prediction

- Supervised learners can also be used to *predict numeric data* such as income, laboratory values, test scores, or counts of items.
- To predict such numeric values, a common form of *numeric prediction* fits linear regression models to the input data.
  - Although *regression models* are not the only type of numeric models, they are by far the most widely used.
  - Regression methods are widely used for forecasting, as they *quantify* in exact terms the association between the inputs and the target, including both the magnitude and uncertainty of the relationship.



Descriptive Models

- A *descriptive model* is used for tasks that would benefit from the insight gained from summarizing data in new and interesting ways.
- In fact, because there is no target to learn, the process of training a descriptive model is called *unsupervised learning*.
- The descriptive modeling task of dividing a dataset into homogeneous groups is called *clustering*.

Pattern Discovery

- The descriptive modeling task called *pattern discovery* is used to identify frequent associations within data.
  - Detect patterns of fraudulent behavior, screen for genetic defects, or prevent criminal activity.

Transaction 1	
Transaction 2	
Transaction 3	
Transaction 4	
Transaction 5	
Transaction 6	
Transaction 7	
Transaction 8	

Thank You for Your Time