L1 - Introduction

- Applied Computational Intelligence with focus on machine learning (data-driven artificial intelligence)
- Contents
  - Why Machine Learning?
  - Problems Machine Learning Can Solve
  - Why Python?
  - Essential Libraries and Tools
  - CoLab platform of Google
  - A First Application: Classifying Iris Species

Why Machine Learning?

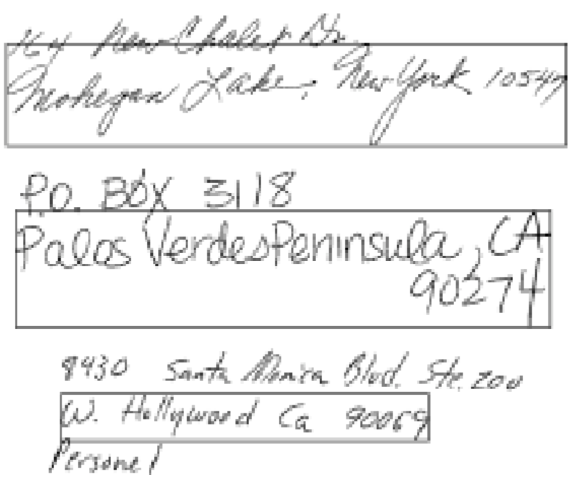
- Have a tremendous influence on the way of data-driven technology
- Data-driven vs. Hand-coded rules
- Disadvantage of system based on hand-coded rules
  - The logic required to make a decision is specific to a single domain and task. Changing the task even slightly might require a rewrite of the whole system.
  - Designing rules requires a deep understanding of how a decision should be made by a human expert, which is however tough.
  - E.g., face detection (unsolved until 2001); Tough as in which way pixels “perceived” by computer is different from how human does

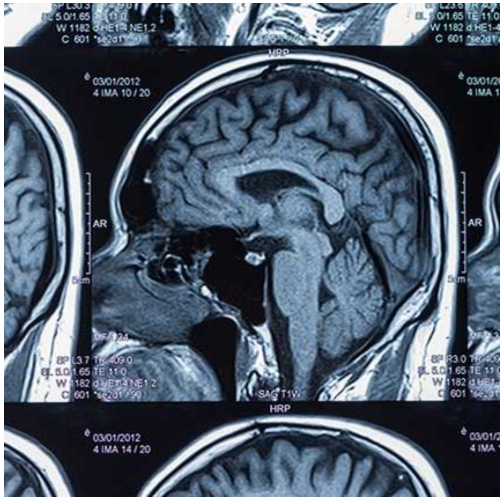
Problems Machine Learning Can Solve: Supervised Learning

- Those as automatic decision making processes
- Supervised Learning
  - User provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired output given an input
  - Algorithm is able to create an output from an input it has never seen before without any help from a human
  - Creating a dataset of inputs and outputs is often a laborious manual process
  - Easy to understand & easy to measure its performance

Examples of Supervised ML Tasks

- Identifying the zip code from handwritten digits on an envelope (data-collection: laborious by easy & cheap)
- Determining whether a tumor is benign based on a medical image (data-collection: expensive expert; ethical & privacy)
- Detecting fraudulent activity in credit card transactions





Problems Machine Learning Can Solve: Unsupervised Learning

- Only the input data is known but not the output data
- The problem is usually harder to understand and evaluate
- Example unsupervised learning examples:
  - Identifying topics in a set of blog posts (might not known beforehand what these topics are or how many topics)
  - Segmenting customers into groups with similar preferences (do not know in advance what these groups might be)
  - Detecting abnormal access patterns to a website (in this example you only observe traffic, and you don’t know what constitutes normal and abnormal behavior, this is an unsupervised problem)

Feature Extraction or Feature Engineering

- Need a representation that input data can be understood by a computer – thinking your data as a table
  - Each data point is a row (named as sample)
  - Each property of data points is a column (named as feature)
- Features need to provide enough information
  - If the only feature that you have for a patient is their last name
  - No algorithm will be able to predict their gender
  - If you add another feature that contains the patient's first name, you will have much better chance as it is often possible to tell the gender by a person's first name
- Important to know your task and your data

Why Python?

- It combines the power of general-purpose programming languages with the ease of use of domain-specific scripting languages such as MATLAB
- Rich of libraries: data loading, visualization, statistics, natural language processing, image processing, etc.
- scikit-learn: an open source project (http://scikit-learn.org/)
- Cloud Platform CoLab of Google (free)
  - Start from creating an account in google
  - https://colab.research.google.com/

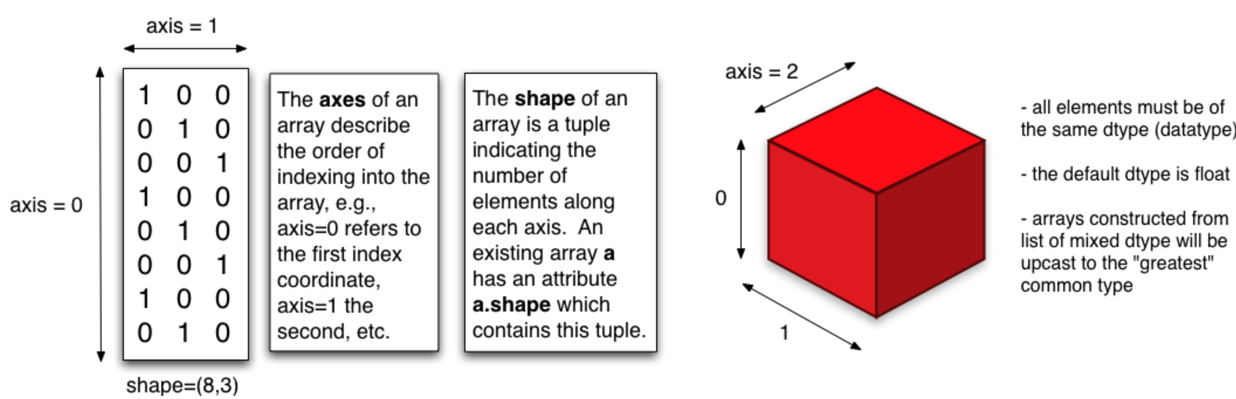
```
# Hello World
print("Hello World! \nThis is my first program of Python running on CoLab.")
```

Essential Libraries and Tools of Python

- NumPy
  - contains functionality for multidimensional arrays, high-level mathematical functions such as linear algebra operations and the Fourier transform, and pseudorandom number generators.
  - A NumPy array looks like

```
import numpy as np
x = np.array([[1, 2, 3], [4, 5, 6]])
print("x:\n{}".format(x))
print(x[0, 1])  # 4
```

Anatomy of an array



- SciPy: collection of functions for scientific computing
  - Advanced linear algebra routines, mathematical function optimization, signal processing, special mathematical functions, and statistical distributions.
  - Most important for us is scipy.sparse: provides sparse matrices

```
from scipy import sparse
# Create a 2D NumPy array with a diagonal of ones, and zeros everywhere else
eye = np.eye(4)
print("NumPy array:\n{}".format(eye))
# https://en.wikipedia.org/wiki/Sparse_matrix

# Convert the NumPy array to a SciPy sparse matrix in CSR format; only nonzero entries are stored
sparse_matrix = sparse.csr_matrix(eye)
print("\nSciPy sparse CSR matrix:\n{}".format(sparse_matrix))
```

  - Usually it is not possible to create dense representations of sparse data (as not fit into memory), so we need to create sparse representations directly (http://www.scipy-lectures.org/).

- matplotlib: the primary scientific plotting library in Python
  - It provides functions for making publication-quality visualizations such as line charts, histograms, scatter plots, and so on.

```
%matplotlib inline
import matplotlib.pyplot as plt
# Generate a sequence of numbers from -10 to 10 with 100 steps in between
x = np.linspace(-10, 10, 100)
# Create a second array using sine
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```

- pandas: a Python library for data wrangling and analysis
  - Simply put, a pandas data structure as DataFrame is a table, similar to an Excel spreadsheet.
  - provides a great range of methods to modify and operate on this table (i.e., SQL-like queries and joins of tables)

- pandas (continue):
  - In contrast to NumPy, which requires that all entries in an array be of the same type, pandas allows each column to have a separate type (for example, integers, dates, floating-point numbers, and strings).
  - It can import from a great variety of file formats and databases, like SQL, Excel files, and comma-separated values (CSV) files.

```
import pandas as pd
from IPython.display import display
# create a simple dataset of people
data = {'Name': ["John", "Anna", "Peter", "Linda"],
'Location' : ["New York", "Paris", "Berlin", "London"],
'Age' : [24, 13, 53, 33]
}
data_pandas = pd.DataFrame(data)
# IPython.display allows "pretty printing" of dataframes
# in the Jupyter notebook
display(data_pandas)
```

```
# Select all rows that have an age column
greater than 30
display(data_pandas[data_pandas.Age > 30])
```

Python 2 vs. Python 3

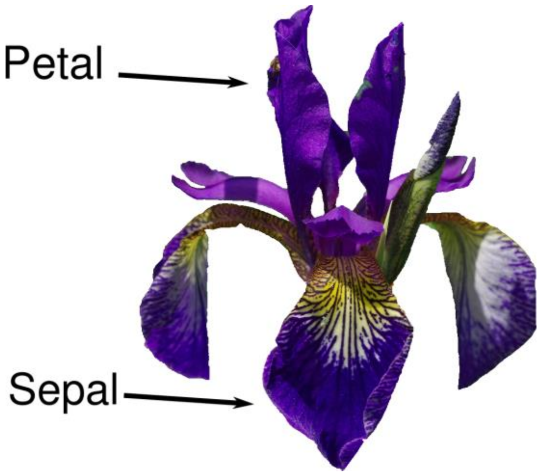
- Python 2 is no longer actively developed
- Migrate to Python 3 if you have old code in Python 2

```
import sys
print("Python version: {}".format(sys.version))
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import matplotlib
print("matplotlib version: {}".format(matplotlib.__version__))
import numpy as np
print("NumPy version: {}".format(np.__version__))
import scipy as sp
print("SciPy version: {}".format(sp.__version__))
import IPython
print("IPython version: {}".format(IPython.__version__))
import sklearn
print("scikit-learn version: {}".format(sklearn.__version__))
```

First Application: Classifying Iris Species

- Data contains the measurements of some irises that have been previously identified by an expert botanist
  - Belonging to the species *setosa*, *versicolor*, or *virginica*
  - For these measurements, one can be certain of which species each iris belongs to
  - Our goal: to build a machine learning model that can predict the species for a new iris.

  - An example of classification problem
  - For a particular data point, the species it belongs to is called its label.



Meet the Data

- What we used is the Iris dataset

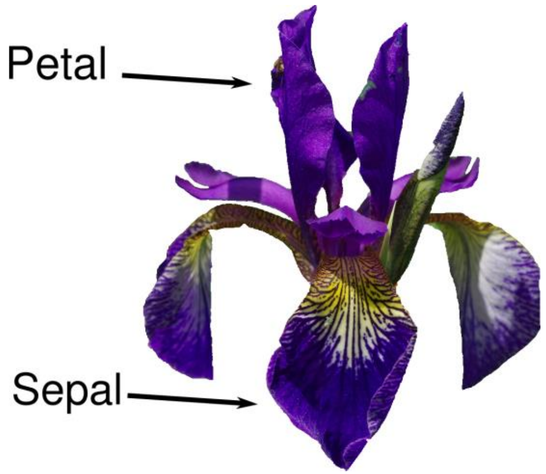
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
print("Keys of iris_dataset: \n{}".format(iris_dataset.keys()))

print(iris_dataset['DESCR'][:193] + "\n...")

print("Target names: {}".format(iris_dataset['target_names']))
print("Feature names: \n{}\n".format(iris_dataset['feature_names']))

print("Type of data: {}".format(type(iris_dataset['data'])))
print("Shape of data: {}\n".format(iris_dataset['data'].shape))
print("First five rows of data:\n{}".format(iris_dataset['data'][:10]))

print("Type of target: {}".format(type(iris_dataset['target'])))
print("Shape of target: {}\n".format(iris_dataset['target'].shape))
print("Target:\n{}".format(iris_dataset['target']))
```



Target:

0 means *setosa*

1 means *versicolor*

2 means *virginica*

Measuring Success: Training & Testing

- Cannot use the training data to evaluate the performance
- To assess the model's performance, we show it new data (data that it hasn't seen before) for which we have labels
- Splitting the labeled data we have collected into two parts:
  - One part of the data is used to build our machine learning model, and is called the *training data* or *training set*. (around 75%)
  - The rest of the data will be used to assess how well the model works; this is called the *test data*, *test set*, or *hold-out set*.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'], iris_dataset['target'], random_state=0)

print("X_train shape: {}".format(X_train.shape))    print("y_train shape: {}".format(y_train.shape))
print("X_test shape: {}".format(X_test.shape))      print("y_test shape: {}".format(y_test.shape))
```

First Things First: Look at Your Data

- Before building a machine learning model it is often a good idea to inspect the data
  - Check if the desired information contained in the data
  - A good way to find abnormalities and peculiarities
- One of the best ways to inspect data is to visualize it
  - by using a scatter plot (not work for high-dim., need *pair plot*)

```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o', hist_kwds={'bins': 20},
s=60, alpha=.8, cmap=mglearn.cm3)
```

  - Note that, diagonal is filled with histograms of each feature.

Building First Model: k-Nearest Neighbors

- Here we use a *k-nearest neighbors* (*knn*) classifier
  - The most important parameter is # of neighbors

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

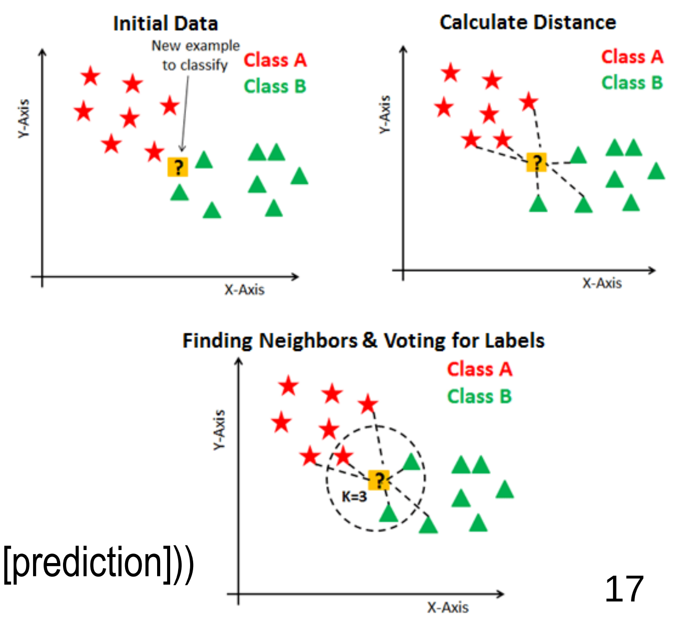
  - The knn object encapsulates the algorithm that will be used to build the model from the training data

```
knn.fit(X_train, y_train)
```

  - Making predictions

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))

prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))
```



Evaluating the Model

- This is where the test set that we created earlier comes in.
  - This data was not used to build the model, but we do know what the correct species is for each iris in the test set.
  - Therefore, we can make a prediction for each iris in the test data and compare it against its label (the known species).
  - We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted:

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))

print("Test set score (by mean): {:.2f}".format(np.mean(y_pred == y_test)))
print("Test set score (by knn.score): {:.2f}".format(knn.score(X_test, y_test)))
```

Summary and Outlook

- Introduction of machine learning (data-driven artificial intelligence)
- Contents
  - Why Machine Learning?
  - Problems Machine Learning Can Solve
  - Why Python?
  - Essential Libraries and Tools
  - CoLab platform of Google
  - A First Application: Classifying Iris Species
  - Splitting labeled set into training (75%) and test (25%) datasets

Course Assessment Scheme

- Four Assignments (70% in total)
  - Assignment 1: Data preparation (10%)
  - Assignment 2: Supervised learning (20%)
  - Assignment 3: Unsupervised learning (20%)
  - Assignment 4: Algorithm chain (20%)
- Final Examination (30%)

Course Project Description