Lab 1: Python

Numpy, Sklearn, PyTorch

Gabriele Trivigno & Debora Caldarola

nome.cognome@polito.it

Scikit-learn

https://scikit-learn.org/stable/

General Machine Learning library built on top of numpy

- Machine learning algorithms
  - SVM
  - k-means
  - random forest
  - ...
- Utilities
  - data preprocessing
  - model selection

Brief recap on Logistic Regression

Input: dataset of independent variables

Hypothesis: $Y = WX + B$

Output: estimate of the probability of an event occurring → between 0 and 1. Modeled as: $h(x) = sigmoid(Y)$

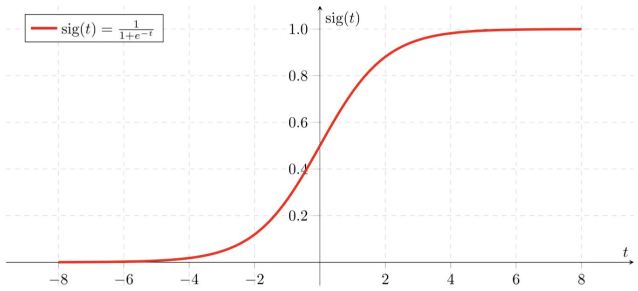
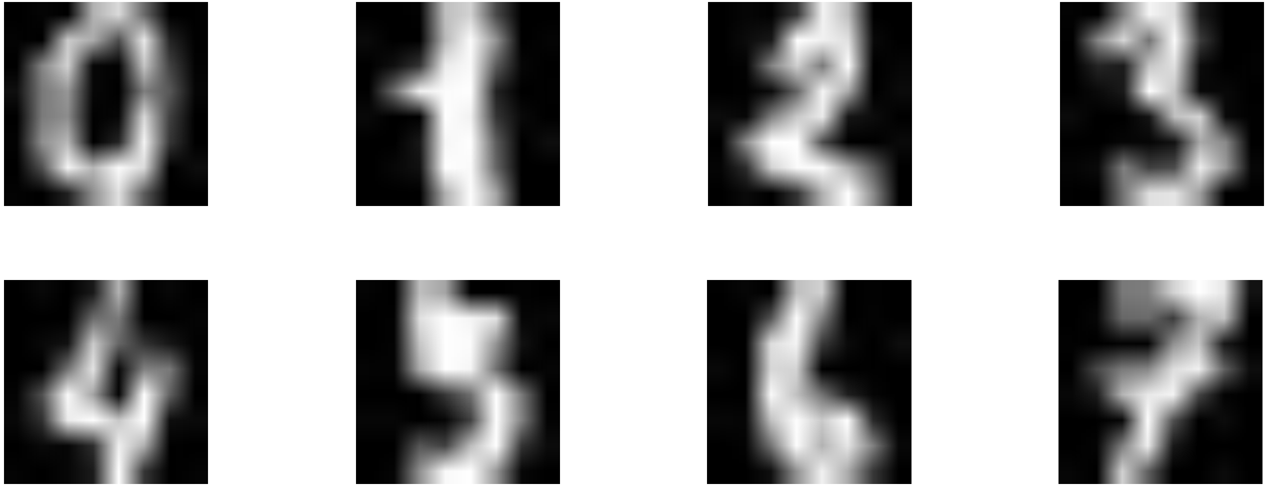


Figure 2: Sigmoid Activation Function

Image from https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Problem: Classification on handwritten Digits



Problem: Classification on handwritten Digits

Load sklearn's digits dataset

- X.shape is (1797 x 64)
  - what do these dimensions represents?

```
import sklearn
from sklearn.datasets import load_digits
X, y = datasets.load_digits(return_X_y=True)
X.shape # Out: (1797, 64)
y.shape # Out: (1797,)
```

Problem: Classification on handwritten Digits

Load the digits dataset (8x8 images)

- X.shape is (1797 x 64)
  - what do these dimensions represents?

```
import sklearn
from sklearn.datasets import load_digits
X, y = datasets.load_digits(return_X_y=True)
X.shape # Out: (1797, 64)
y.shape # Out: (1797,)
```

To show an image, you have to resize the nd-array to 8x8.

- Use numpy.reshape

```
import matplotlib.pyplot as plt
reshaped_arr = X[0].reshape(8,8)
plt.imshow(reshaped_arr)
```

Problem: Classification on handwritten Digits

To measure performances of a classification algorithm, we need training and test sets

Create train-test splits from the load_digits dataset

```
import sklearn
from sklearn.datasets import load_digits
X, y = load_digits(return_X_y=True)
X.shape # Out: (1797, 64)
y.shape # Out: (1797,)
```

```
X_train, y_train = X[:-200], y[:-200]
X_test, y_test = X[-200:], y[-200:]
```

Problem: Classification on handwritten Digits

To measure performances of a classification algorithm, we need training and test sets

Create train-test splits from the load_digits dataset

```
import sklearn
from sklearn.datasets import load_digits
X, y = load_digits(return_X_y=True)
X.shape # Out: (1797, 64)
y.shape # Out: (1797,)
```

```
X_train, y_train = X[:-200], y[:-200]
X_test, y_test = X[-200:], y[-200:]
```

- **Teaser question**: is this situation realistic?
    - What do we expect on a real scenario?

Problem: Classification on handwritten Digits

Normalize data

Many Machine Learning algorithms work better on standardized data

- 0 mean and unit variance

Apply the standard scaler on train and test

- Why haven't we used the whole dataset?

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Problem: Classification on handwritten Digits

Train a linear regression model on the training set

- We use sklearn LogisticRegression

```
from sklearn.linear_model import
LogisticRegression
regressor = LogisticRegression(solver='lbfgs')
regressor.fit(X_train, y_train)
```

To predict use the `predict` function

```
regressor.predict(X_test[0].reshape(1,-1))
```

Visualize the accuracy on the test set with score

```
regressor.score(X_test, y_test)
# Out: 0.92
```

Problem: Classification on handwritten Digits

Train a linear regression model on the training set

- We use sklearn LogisticRegression

```
from sklearn.linear_model import
LogisticRegression
regressor = LogisticRegression(solver='lbfgs')
regressor.fit(X_train, y_train)
```

To predict use the `predict` function

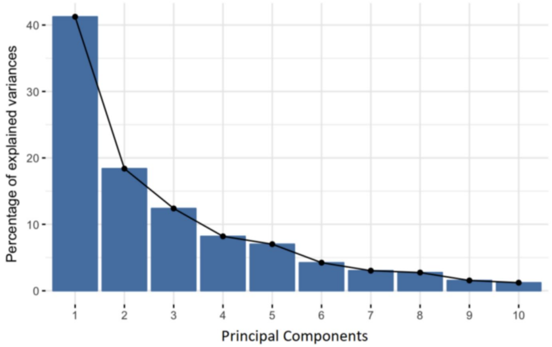
```
regressor.predict(X_test[0].reshape(1,-1))
```

Visualize the accuracy on the test set with score

- What if the data is too large?

```
regressor.score(X_test, y_test)
# Out: 0.92
```

Principal Component Analysis (PCA)



- Method for reducing the dimensionality of large datasets
- The large set of variables is transformed into a set of smaller ones, containing most of the information of the original dataset

Steps:

1. Standardize the data, so that larger values do not dominate over small ones
2. Compute the eigenvalues and eigenvectors of the covariance matrix
   a. The eigenvectors of the Covariance matrix are actually *the directions of the axes where there is the most variance* (most information)
3. The eigenvectors of the Covariance matrix are the **Principal Components**. The first N PCs are associated with the largest N eigenvalues.

Image from https://builtin.com/data-science/step-step-explanation-principal-component-analysis

Problem: PCA for feature selection

We use PCA

- Why don't we select the number of components manually?

```python
from sklearn.decomposition import PCA
pca = PCA(.95) # keep 95% variance
pca.fit(X_train)
```

```python
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

Problem: PCA for feature selection

We use PCA

- Why don't we select the number of components manually?

We apply the logistic regressor on the data with reduced features

- We observe accuracy drop on the test set

```
from sklearn.decomposition import PCA
pca = PCA(.95) # keep 95% variance
pca.fit(X_train)
```

```
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```
regressor = LogisticRegression(solver='lbfgs')
regressor.fit(X_train, y_train)
regressor.score(X_test, y_test)
# Out: 0.905
```

PCA example: Iris Dataset

Flower dataset of three species of Iris

Four features:

- length and width of sepals
- length and width of petals

What happens if we apply PCA to the data?

IRIS dataset

Iris Versicolor

Iris Setosa

Iris Virginica

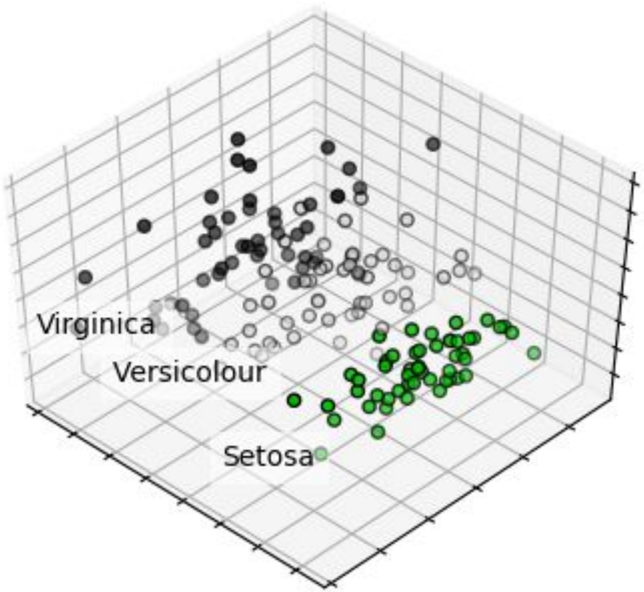
PCA example: Iris Dataset

Iris Versicolor

Iris Virginica

Iris Setosa

sample 1: [length sepals, width sepals, length petals, width petals]
sample 2: [length sepals, width sepals, length petals, width petals]
sample 3: [length sepals, width sepals, length petals, width petals]
...

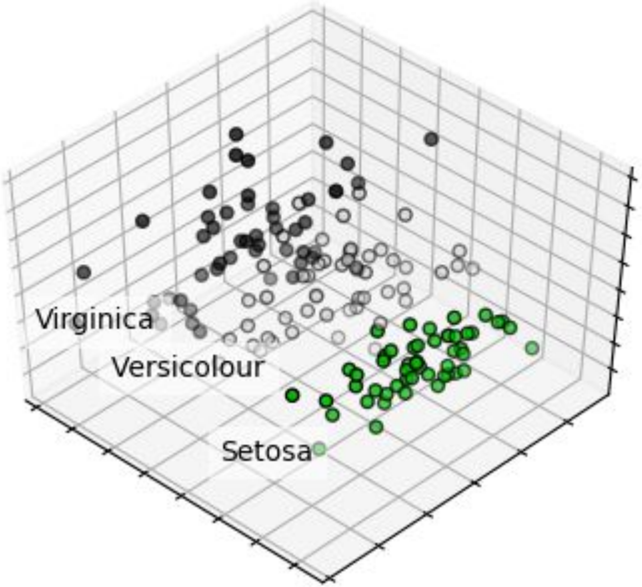


PCA example: Iris Dataset

3d plot of PCA using three components

https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html#sphx-glr-auto-examples-decomposition-plot-pca-iris-py

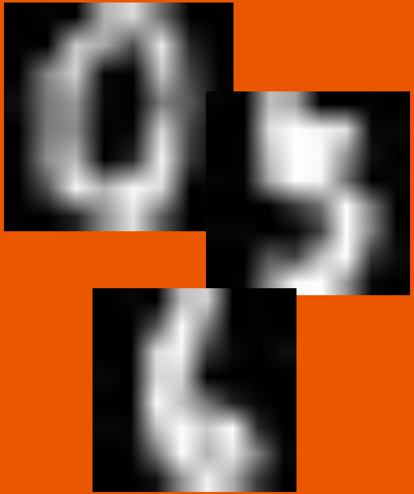
- Follow the example, what happens if we choose different components? Can you visualize results?



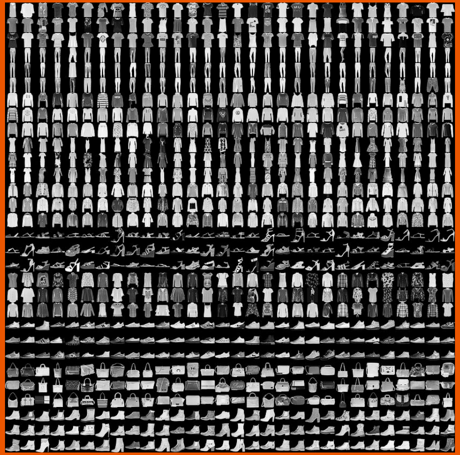
Exercise 1: Logistic Regression with Sklearn

Reproduce the previous example in Colab:

- train a logistic regressor on MNIST
- how to handle the non-binary labels?
- apply PCA and try again. Visualize the principal components
- In what kind of space does the reduced dataset live?



Exercise 2: PCA on Fashion MNIST



1. Clone the Fashion-MNIST github repository
   - git clone https://github.com/zalandoresearch/fashion-mnist.git
2. Load data following instruction on the repository
   - The training dataset has 60k 24x24 images organized in 10 classes
3. Choose one class and visualize the "eigendresses" using PCA
   - Use sklearn's PCA class for decomposition (check the documentation for the class functions)
   - Choose one image and visualize what happens to it when you re-project it using only th first 6 PC *vs* the last 6 PC.

Suggestions

- Remember to Standardize the data
- Project data onto the first `n` components. You can do it in Python by running:
  - `X_t = PCA(n_components=x).fit_transform(X)`
- You have to calculate the principal components of the whole dataset, and then you can transform the single image to the new chosen base
- The transformation with the last 6 components cannot be automatically done by `sklearn`. You have to explicitly extract all components, pick the last 6 ones, and then manually apply the transformation to the new subspace following formulas in the slides
  - Once fitted, the PCA object holds all the components
  - When in doubt, check the documentation

Deep Learning with PyTorch

https://pytorch.org/

Tensors - Recap

- Tensors are the PyTorch **counterpart of Numpy arrays**
- Contain *only numerical values*
- Used to encode:
  - **Signal to process** (e.g. images, strings of text, videos, ...)
  - **Internal states and parameter** of neural networks
- **All of PyTorch computation takes place on Tensors**

```
#                         R    G    B
img = tensor([[[ 48,  80,  79],
               [175, 104, 207],
               [162,  24, 224],
               [ 97,  27,  28],
               [ 51, 137,  60],
               [124, 214, 249]],
               ...
            ]])
t.size() ==> [256, 256, 3]
t.device ==> gpu:0
t.dtype ==> torch.float32
```

Tensors

- **Create a Tensor 't' of size (2,3) from scratch:**
  - Zeros init:
    t = torch.zeros(size=(2,3), dtype=torch.float32)
  - Ones init:
    t = torch.ones(size=(2,3), dtype=torch.float32)
  - Random init:
    t = torch.rand(size=(2,3), dtype=torch.float32)
- **Properties:**
  - **t.size()** - returns its shape - OSS. size() is a method of the Tensor class! (not a property)
  - **t.device** - whether it is store on CPU or GPU (+index)
  - **t.dtype** - values type (e.g. torch.int8, torch.float32, torch.bool, ...)

1. Create a tensor from scratch in PyTorch

```
In [10]: torch.zeros(size=(2,3), dtype=torch.float32)
Out[10]:
tensor([[0., 0., 0.],
        [0., 0., 0.]])

In [11]: torch.ones(size=(2,3), dtype=torch.float32)
Out[11]:
tensor([[1., 1., 1.],
        [1., 1., 1.]])

In [12]: torch.rand(size=(2,3), dtype=torch.float32)
Out[12]:
tensor([[0.5818, 0.3530, 0.5550],
        [0.4265, 0.1131, 0.5708]])
```

2. Check tensor 't' properties

```
In [21]: print(t)
tensor([[0.2330, 0.1985, 0.6867],
        [0.3123, 0.6324, 0.1508]])

In [22]: t.size()
Out[22]: torch.Size([2, 3])

In [23]: t.device
Out[23]: device(type='cpu')

In [24]: t.dtype
Out[24]: torch.float32
```

Tensors

- **From NumPy array to Tensor:**
  - use **torch.from_numpy()** static method
  - line 36
- **From Tensor to NumPy array:**
  - use tensor.**numpy()** method
  - line 38
- **Move tensor between CPU and GPU**:
  - Tensor are initialized on CPU by default
  - **tensor.device** to check where it is stored
  - **tensor.cuda()** to move it on GPU:0
  - **tensor.cpu()** to move it on CPU

3. Numpy bridge

```
In [33]: import torch, numpy as np

In [34]: np_arr = np.random.rand(2, 3)

In [35]: np_arr.shape
Out[35]: (2, 3)

In [36]: t = torch.from_numpy(np_arr)

In [37]: t.size()
Out[37]: torch.Size([2, 3])

In [38]: np_arr_theReturn = t.numpy()

In [39]: np_arr_theReturn.shape
Out[39]: (2, 3)

In [40]: np_arr_theReturn
Out[40]:
array([[0.20823187, 0.55286813, 0.37927967],
       [0.31437054, 0.21745502, 0.22156234]])
```

4. Move tensor to GPU, you need a CUDA capable device

```
In [18]: print(t,", device: ", t.device)
tensor([[0.6417, 0.4857, 0.9736],
        [0.1095, 0.5623, 0.1343]]) , device:  cpu

In [19]: t = t.cuda() # move to gpu!

In [20]: print(t,", device: ", t.device)
tensor([[0.6417, 0.4857, 0.9736],
        [0.1095, 0.5623, 0.1343]], device='cuda:0') , device:  cuda:0
```

Exercise 3: basic operations with PyTorch

1. Initialize two random 4D-tensors and move them to GPU
2. Basic math operations:
    a. Sum the two tensors
        - What's the difference between `torch.add_()` and `torch.sum()`?
    b. Multiply the two tensors
        - What's the difference between using `*`, `@`, `torch.mul()` and `torch.matmul()`?
    c. Concatenate the two tensors along different axis

PyTorch - Basic Components

- **Network Architecture**
  - Define a subclass of `torch.nn.Module`
- **Dataset**
  - Define a subclass of `torch.utils.data.Dataset`
- **Loss Function + Optimizer**
  - Network Prediction penalization + Update Network parameters
- **Training Loop**
  - Simple python script: sort of main function interconnecting all components

PyTorch - Basic Components

- **Network Architecture**
  - Define a subclass of `torch.nn.Module`
- **Dataset**
  - Define a subclass of `torch.utils.data.Dataset`
- **Loss Function + Optimizer**
  - Network Prediction penalization + Update Network parameters
- **Training Loop**
  - Simple python script: sort of main function interconnecting all components

Today, we only care about this...

Data Reading

- Define Preprocessing
  - https://pytorch.org/docs/stable/torchvision/transforms.html
- Create a Dataset class
  - https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset
- Choose a Sampling Strategy
  - https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

Dataset Class

- `torch.data.Dataset`: base class for all datasets
- **Basic Methods to override:**
    - `init()`: initial processing, loading data in memory, ...
    - `len()`: returns the total number of samples in the dataset
    - `getitem(idx)`: given and index return a data sample from the dataset
- **Transforms:** pre-processing or data-augmentation to be applied on each sample

```
In [9]: transform = transforms.Compose([
            transforms.ToPILImage(), # because the input dtype is numpy.ndarray
            transforms.RandomHorizontalFlip(0.5), # because this method is used for PIL Image dtype
            transforms.ToTensor(), # because inpus dtype is PIL Image
        ])

        train_dataset = DatasetMNIST(file_path='../input/train.csv', transform=transform)
```

Example: Instantiating an object Dataset

Dataset Class

```
In [8]: import torch
        import pandas as pd
        from torch.utils.data import DataLoader, Dataset
        from torchvision import transforms

        class DatasetMNIST(Dataset):

            def __init__(self, file_path, transform=None):
                self.data = pd.read_csv(file_path)
                self.transform = transform

            def __len__(self):
                return len(self.data)

            def __getitem__(self, index):
                # load image as ndarray type (Height * Width * Channels)
                # be carefull for converting dtype to np.uint8 [Unsigned integer (0 to 255)]
                # in this example, i don't use ToTensor() method of torchvision.transforms
                # so you can convert numpy ndarray shape to tensor in PyTorch (H, W, C) --> (C, H, W)
                image = self.data.iloc[index, 1:].values.astype(np.uint8).reshape((1, 28, 28))
                label = self.data.iloc[index, 0]

                if self.transform is not None:
                    image = self.transform(image)

                return image, label
```

DataLoader

- PyTorch provides an efficient way to iterate a Dataset through **DataLoader** (`torch.utils.data.DataLoader`)
- Its **constructor takes as input an object of type Dataset**
- Provides:
  - **Data Batching**: we want to forward to our network batches of data (*e.g.* 32 images at a time)
  - **Data Shuffling**
  - **Parallel Data Loading**: multiple threads/workers loading data in parallel

DataLoader

- Constructor takes as input an object of type Dataset

```
train_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=True, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_train, shuffle=True)
```

Example: Instantiating a DataLoader object

Exercise 4: Datasets in PyTorch

1. Write the Dataset class for loading **CIFAR10**
2. Write the associated Dataloaders
   - Distinguish between `trainloader` and `testloader`
3. Iterate over the dataset and visualize one image for each class
   - Labels: `{0: "airplane", 1: "automobile", 2: "bird", 3: "cat", 4: "deer", 5: "dog", 6: "frog", 7: "horse", 8: "ship", 9: "truck"}`

Exercise 5 - Classification on CIFAR

- Use the Dataset and Dataloader just implemented
- Copy the rest of the script from here (**NOT** the Dataset class) - tutorial
- Treat the network and the optimizer as a **black box**, focus on the dataset class implementation

Bonus Exercise: Logistic Regression on MNIST in Pytorch

- Take the Dataset and Dataloader just implemented and make it work with MNIST
- You can take the loss and optimizer from the previous example
- Figure out how to fit your X weight matrix!