Activity-Aware Systems
Data Mining with Python

http://www.cs.unibo.it/difelice/

Context Aware Systems

Prof. Marco Di Felice
Department of Computer Science and Engineering
University of Bologna
Python Language

- Created by Guido van Rossum and first released in 1991
- Actual version: 3.8.0
- Multi-paradigm (object-oriented, procedural, functional)
- Interpreted language (multi-platform)
- Highly Modular, several packages available
Python Language Pills

- Python uses **whitespace indentation** (rather than curly brackets or keywords) to delimit **blocks**.

```python
while n>0:
    n=n-1
print n
```

```python
while n>0:
    n=n-1
print n
```

**THESE BLOCKS ARE NOT EQUIVALENT!!**
Python Language Pills

- Type constraints are **not checked** at compile time
- No need to define types; just use the variables!

```python
message="Hello world"
a=5
x=2.34

>>> type(message)
<type 'str'>
```
Python Language Pills

- **Selection** statement: conditionally executes a block of code

```python
if x==y:
    print("Equal")
else:
    print("Different")
```
Iterative statement: executes a block of code as long as its condition is true.

```python
x=1.0
while x<10.0:
    print (math.log(x))
    x=x + 1.0
```
Python Language Pills

- Function can be declared with the `def` keyword

- Function allows for **partial allocation**

```python
def factorial(n):
    if (n==0):
        return 1
    else:
        return n*factorial(n-1)

>> print factorial(4)
>> 24
```
Python Language Pills

- Function can be declared with the `def` keyword
- Function allows for **partial allocation**

```python
def main()
    print("Hello world!")
main()
```
Python Language Pills

- **List** → array of mutable and ordered elements
- Each element has an index (integer value)
- Elements can be **heterogeneous** (different types)

thislist = ["apple", "banana", "cherry"]
hetlist = [0, 1, “apple”, “cherry”]
print(thislist[1])
print(thislist[-1])
Python Language Pills

- **Lambda function** → single line, anonymous functions
- Lambda function can have multiple input parameters

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```
Python Language Pills

- List → array of mutable and ordered elements
- Each element has an index (integer value)
- Elements can be heterogeneous (different types)

```
thislist = ["apple", "banana", "cherry"]
thislist[0]= [“raspberry” ]
a.append(“orange”)
a.insert(1,”orange”)
```
Python Language Pills

- **List** → array of mutable and ordered elements
- Each element has an index (integer value)
- Elements can be **heterogeneous** (different types)

```python
thislist = ["apple", "banana", "cherry"]
thislist[0]= [“raspberry” ]
for entry in thislist:
    print(entry)
```

**ACTIVITY-AWARE COMPUTING: DATA MINING WITH PYTHON**

**MARCO DI FELICE**
Python Language Pills

- **List Comprehension** → concise way to create a list

- `[ expression for item in list if conditional ]

```python
mylist=[]
For i in range(10):
    mylist[i]=i**2

mylist = [x**2 for x in range(10)]
print mylist
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```
Python Language Pills

- **List Comprehension** → concise way to create a list

```
mylist = []
cont = 0
For i in range(10):
    if (i%2==0):
        mylist[cont]=i**2
        cont+=1
```

```
mylist = [x**2 for x in range(10) if (x%2==0)]
```
Python Language Pills

- **Dictionary** → collection of mutable and unordered elements
- Surrounded by brackets `{ }`
- Set of couples `<key:value>`

```python
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["year"] = 2018
```
- **Dictionary** → collection of mutable and unordered elements
- Surrounded by brackets `{ }`
- Set of couples `<key:value>`

```python
for x in thisdict.values():
    print(x)
for x in thisdict.keys():
    print(thisdict[x])
```
Python for Data Science
NumPy

- Package for **scientific computing** with Python
- Main Features
  - Multidimensional arrays (matrices)
  - Efficient computation (matrix math)
  - Math functions for linear algebra

apt-get install python-numpy

pip install numpy

https://numpy.org/devdocs/user/quickstart.html
NumPy

- The main data structure in Numpy is represented by a homogeneous multidimensional array.
- All elements must be of the same type, indexed by a tuple of non-negative integers.
- Array is managed by class ndarray.
- Main methods: ndim, shape, size, dtype, itemsize, data

https://numpy.org/devdocs/user/quickstart.html
NumPy

- Array **Creation** (several ways)

```python
import numpy as np
a = np.array([[1,2,3],[4,5,6]], dtype=np.int16)
b = np.array([[1,2,3,4,5]])
c = np.array([1.2, 3.5, 5.1])

print a.ndim       # 2 (axes)
print a.shape       # 2x1
```
NumPy

Array **Creation** (several ways)

- `np.ones` → Fill the matrix of 1 values
- `np.zeros` → Fill the matrix of 0 values
- `np.arange` → Fill the matrix of incremental values
- `np.random.random` → Fill the matrix of random values

```python
np.ones(2,4)  # Array([[1, 1, 1, 1], [1, 1, 1, 1]])
```
NumPy

Array **Creation** (several ways)

- `np.ones` → Fill the matrix of 1 values
- `np.zeros` → Fill the matrix of 0 values
- `np.arange` → Fill the matrix of incremental values
- `np.random.random` → Fill the matrix of random values

```python
np.arange(10, 30, 5)  # array([10, 15, 20, 25])
```
NumPy

- Array **Loading** (from file)

```python
data = numpy.genfromtxt('data.txt', delimiter = ',')
```

- `dtype`: data type of the array
- `missing_values`: strings corresponding to missing data
- `skip_header`: skip initial lines
- `usecols`: which columns to read
- `names`: {None, True} reads the first line as header
NumPy

Array Manipulation

- Array can be manipulated through arithmetic operations.
- A new array is created and filled with the result.
- Example of binary operations: add(+), minus(-), mul(*)

```
A = np.array([[1,0], [0,3]])
B = np.array([[2,0], [0,4]])
C = A * B
array([[2, 0], [0, 12]])
```
NumPy

Array Manipulation

- Array can be manipulated through arithmetic operations.
- **Universal functions** applying elementwise on the array
- Example of *universal functions*: sin, cos, exp

```python
B=np.array([0, 1, 2])
np.exp(B)
array([ 1.0 ,  2.71828183,  7.3890561 ])
```
The `shape` of an array is the number of elements on each axis. The shape of an array can be changed with the `reshape` operator (input $\rightarrow$ new shape) or with the `resize` operator.

```python
b = np.arange(15).reshape(3,5)
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```
Array Merging

Merge two arrays on different axes

```
a = np.array([[[1, 2], [3, 4]]])
b = np.array([[[5, 6, 7], [8]]])
# Merge on rows
c = np.concatenate((a, b))
print(c)
# Merge on columns
d = np.concatenate((a, b), axis=1)
print(d)
```
Array **Slicing/Indexing**

- Extension of Python slicing (on multiple axes)
- Syntax: `[start; stop; step]`

```python
a = list(range(20))
a[1:3]  #Result?
a[:3]   #Result?
a[-3:]  #Result?
a[3:8:2] #Result?
a[4:1:-1] #Result?
```

**Extension of Python slicing (on multiple axes)**

**Syntax**: `[start; stop; step]`
NumPy

- Array **Slicing/Indexing**
  - Extension of Python slicing (on multiple axes)
  - Syntax: [start; stop; step]

```python
a = array([[ 0,  1,  2,  3],
           [10, 11, 12, 13],
           [20, 21, 22, 23],
           [30, 31, 32, 33],
           [40, 41, 42, 43]])
print (a[0:5, 1]) #Result?
print (a[0:5, :]) #Result?
```
NumPy

Array 

- Array can be passed by reference or by value

```python
a=np.array([[ 0,  1,  2,  3],
            [10, 11, 12, 13])
b=a  # REFERENCE
b[0,0]=100
print(a)
>> array([[100,  1,  2,  3],
          [10, 11, 12, 13]])
```
NumPy

Array Copy vs View

Array can be passed by reference or by value

```python
a=np.array([[ 0, 1, 2, 3],
            [10, 11, 12, 13])
b=a.view()  # REFERENCE
b[0,0]=100
print(a)
>> array([[100, 1, 2, 3],
          [10, 11, 12, 13]])
```
NumPy

- Array **Copy vs View**
  - Array can be passed by **reference** or by **value**

```python
a=np.array([[ 0,  1,  2,  3],
            [10, 11, 12, 13]])
b=a.copy()  # VALUE
b[0,0]=100
print(a)   
>>> array([[ 0,  1,  2,  3],
         [10, 11, 12, 13]])
```
MatPlotLib

- Package for data visualization and plot creation

```python
import matplotlib.pyplot as plt
```

[Anatomy of a figure diagram]
MatPlotLib

Package for data visualization and plot creation

```python
import matplotlib.pyplot as plt
values=[10.3, 13.23, 15.12, 16.12, 17.13, 18.67]
plt.title("Plot test")
plt.ylabel("Current values")
plt.xlabel("Years")
plt.plot(years, values, color='green', marker='o')
plt.show()
```
MatPlotLib

Package for **data visualization and plot creation**

```python
import matplotlib.pyplot as plt
values1=[10.3, 13.23, 15.12, 16.12, 17.13, 18.67]
values2=[20.3, 18.23, 15.12, 14.12, 12.13, 10.67]
plt.title("Plot test")
plt.ylabel("Current values")
plt.xlabel("Years")
plt.plot(years, values1,'g-', label="serie1")
plt.plot(years, values2,'r:', label="serie2")
plt.show()
```
MatPlotLib

Package for data visualization and plot creation

```python
import matplotlib.pyplot as plt
valuesX=[0,1,2,3,4,5]
valuesY=[10.3, 13.23, 15.12, 16.12, 17.13, 18.67]
plt.axis([-1,6,10,19])
plt.xticks(np.arange(6), label)
plt.bar(valuesX, valuesY, 0.5)
plt.ylabel("Current values")
plt.xlabel("Years")
plt.show()
```
Scikit-Learn

- Package for **Machine Learning** in Python
  - Based on **NumPy** and SciPy
  - Supports both **supervised** learning (classification, prediction) and **unsupervised** learning (clustering)
  - Integrated with other Python libraries (e.g. matplotlib)

```
pip install scikit-learn
```
Scikit-Learn

- We will implement this **data pipeline**:

1. **Load the dataset**
2. Build training/test sets
3. Preprocess the training set
4. Set up the classifier
5. Train the classifier
6. Run the classifier
7. Get and show the evaluation performance
Scikit-Learn

- **Load** the dataset (using NumPy library)

Create two NUMPY arrays from the dataset:
- One array with the data instances (without the classes’ info)
- Another array containing the classes’ info only

```python
data = numpy.genfromtxt('file.txt', delimiter = ',',
                        usecols={1,2,3})

classes = numpy.genfromtxt('file.txt', delimiter = ',',
                          usecols={4})
```
We will implement this data pipeline:

1. Load the dataset
2. Build training/test sets
3. Preprocess the training set
4. Set up the classifier
5. Train the classifier
6. Run the classifier
7. Get and show the evaluation performance
Scikit-Learn

- **EVALUATION TECHNIQUES**

  - **HOLDOUT MODEL**
  - **ORIGINAL DATASET**
  - **TRAINING**
  - **TESTING**
  - Data-mining algorithm
  - Predictive model
  - e.g. **CLASSIFICATION ACCURACY**

  Scikit-Learn: **ACTIVITY-AWARE COMPUTING: DATA MINING WITH PYTHON**
  MARCO DI FELICE
Scikit-Learn

- **EVALUATION TECHNIQUES**

  - ORIGINAL DATASET
  - TRAINING
  - CALIBRATION
  - TESTING

  - INITIAL MODEL
  - PREDICTIVE MODEL
  - PARAMETER OPTIMIZATION

  - HOLDOUT MODEL (extended)

  - Data-mining algorithm
  - e.g. CLASSIFICATION ACCURACY
### Scikit-Learn

#### EVALUATION TECHNIQUES

<table>
<thead>
<tr>
<th>K-FOLD MODEL</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ORIGINAL DATASET</th>
</tr>
</thead>
</table>

**K=10**

<table>
<thead>
<tr>
<th>TRAIN</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TEST</th>
</tr>
</thead>
</table>

---

**ACTIVITY-AWARE COMPUTING: DATA MINING WITH PYTHON**

MARCO DI FELICE
Split the dataset into test/train set

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=10)
```
We will implement this **data pipeline**:

1. Load the dataset
2. Build training/test sets
3. **Preprocess the training set**
4. Set up the classifier
5. Train the classifier
6. Run the classifier
7. Get and show the evaluation performance
Scikit-Learn

- Manage the missing values

```python
from sklearn.impute import SimpleImputer
filter = SimpleImputer(missing_values=np.nan, strategy='mean')
Xtest = filter.fit_transform(Xtest)
```

Parameters:
- strategy → mean, median, most_frequent, constant
- missing_values → placeholder for missing values
- fill_value → replacing value, used only by the strategy constant
Scikit-Learn

- Remove the missing values (NumPy)

```python
mask = np.any(np.isnan(train_data), axis=1)
train_data=train_data[~mask]
```
Scikit-Learn

- Scale the values of the dataset

```python
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
```

Parameters:
- `feature_range` → range of scaled data, default 0

Scikit-Learn

- Feature selection (tree-based selection)

```python
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
clf = ExtraTreesClassifier(n_estimators=50)
clf = clf.fit(X, y)
clf = clf.fit(X, y)
print(clf.feature_importances_)
model = SelectFromModel(clf, prefit=True)
X_new = model.transform(X)
```
We will implement this **data pipeline**:

1. Load the dataset
2. Build training/test sets
3. Preprocess the training set
4. **Set up the classifier**
5. Train the classifier
6. Run the classifier
7. Get and show the evaluation performance
Scikit-Learn

- **Dummy Classifier** (only for benchmarking)

```python
from sklearn.dummy import DummyClassifier
neigh = DummyClassifier(strategy="uniform", random_seed=10 )
```

**Parameters:**
- `strategy` ➔ most_frequent, uniform, constant
- `random_seed` ➔ seed of the random value generator
Scikit-Learn

- **K-Nearest Neighbour** (see prev slides)

```python
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
```

**Parameters:**
- `n_neighbours` → Number of neighbours
- `weights` → weight function used by the prediction phase
- `metric` → distance function

Scikit-Learn

- **Naive Bayesian** Classifier (see prev slides)

```python
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

**Parameters:**

- `prior` → Prior probabilities of the classes

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)

Parameters:
✧ n_estimators → number of trees
✧ criterion → metric of the split (e.g. “entropy”)
✧ max_depth → max depth of each tree
Scikit-Learn

- **Decision Tree** Classifier (see prev slides)

```python
from sklearn import tree
clf = tree.DecisionTreeClassifier(random_state=0)
```

Parameters:
- criterion → metric of the split (e.g. “entropy”)
- max_depth → max depth of each tree
Scikit-Learn

- **Perceptron** Classifier (see prev slides)

```python
from sklearn.linear_model import Perceptron
clf = Perceptron(max_iter=500, random_state=0)
```

**Parameters:**
- `tol` → stop threshold
- `max_iter` → maximum number of iterations
- `eta0` → learning rate (multiplicative coefficient)
Scikit-Learn

- We will implement this **data pipeline**:

  1. Load the dataset
  2. Build training/test sets
  3. Preprocess the training set
  4. Set up the classifier
  5. **Train the classifier**
  6. Run the classifier
  7. Get and show the evaluation performance
Train the model over the training set

```python
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(Xtrain, Ytrain)
```
Scikit-Learn

We will implement this data pipeline:

1. Load the dataset
2. Build training/test sets
3. Preprocess the training set
4. Set up the classifier
5. Train the classifier
6. Run the classifier
7. Get and show the evaluation performance
Scikit-Learn

- Test the model over the testing set

```python
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(Xtrain, Ytrain)
result = clf.predict(Xtest)
print(result)
print(Ytest)
```
Scikit-Learn

- We will implement this **data pipeline:**
  1. Load the dataset
  2. Build training/test sets
  3. Preprocess the training set
  4. Set up the classifier
  5. Train the classifier
  6. Run the classifier
  7. **Get and show the evaluation performance**
Scikit-Learn

- Compute the classification accuracy (%)

```python
metrics.accuracy_score(REAL_CLASS, PREDICTED)
```

```python
from sklearn import metrics
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(Xtrain,Ytrain)
result=clf.predict(Xtest)
accuracy=metrics.accuracy_score(Ytest,result)
print(accuracy)
```
Scikit-Learn

Compute the confusion matrix

confusion_matrix(REAL_CLASS, PREDICTED)

from sklearn import metrics
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(Xtrain,Ytrain)
result=clf.predict(Xtest)
cmatrix=metrics.confusion_matrix(Ytest,result)
print(cmatrix)
Scikit-Learn

DEMO/EXERCISE
Scikit-Learn

Download the dataset har_dataset.csv

EX0)
✧ Load the dataset
✧ Remove the first 5 columns
✧ Remove the last column
✧ Merge into a new dataset
EX1)

✧ Replace all missing values with the mean value of each column
✧ Split the dataset into train-test (splitting ratio=30%)
✧ Test four classification algorithms:
   Dummy(uniform), Naive Bayes, Perceptron,
   Random Forest (number of trees: 10)
✧ Repeat the test 10 times: at each run, use a different random seed (random_state) from this set:
   [101,4543,565,124,12,7676,766,90,12,1211]
✧ Plot an histogram (x-axis: classification algorithms, y-axis: mean accuracy over the 10 runs for each algorithm)
Scikit-Learn

EX2)

- Select the algorithm providing the highest accuracy in EX1
- Repeat the analysis by considering four different pre-processing solutions to manage the NULL values: remove all NULL values, replace with mean, replace with most_frequent value, replace with media

- Plot an histogram (x-axis: preprocessing solution, y-axis: mean accuracy over the 10 runs for each solution)
EX3)

✧ Select the best pre-processing solution in EX2
✧ Consider the four algorithms of EX1
✧ Compute the (mean) total number of false positive and false negative classification errors for each algorithm
✧ Plot a stacked histogram (x-axis: classification algorithm, y-axis: number of classification errors, each bar is splitted into two stacked sub-bars, one for the false positive, the other one for the false negative)