# Exemplo de um workflow de aprendizagem máquina

*Conjunto de dados Human Activity Recognition using Smartphones*

*Exemplos de um workflow com pré-processamento, seleção de atributos, aprendizagem não supervisionada e aprendizagem supervisionada*

## Conjunto de dados

https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

*"The experiments have been carried out with a group of 30 volunteers (…). Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (…). Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity (…). The experiments have been video-recorded to label the data manually. The dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data."*

**Attributes***:*

For each record in the dataset it is provided:
- A 561-feature vector with time and frequency domain variables.
- Its activity label.
- An identifier of the subject who carried out the experiment.

# Conjunto de dados: download; estrutura

*Download dos dados:*

*https://archive.ics.uci.edu/ml/machine-learning-databases/00240/*

Ficheiro*: UCI HAR Dataset.zip* (descompactar o ficheiro – pasta base: UCI HAR Dataset/)

Estrutura (ficheiros principais):

- Códigos das atividades: "*activity_labels.txt*" (2 colunas)
- Atributos: "*features.txt*" (561 linhas, 2 colunas)
- Indivíduos (treino ; teste): "train/subject_train.txt" (7352 linhas), "test/subject_test.txt" (2947 linhas), ambos com 1 coluna
- Atributos de entrada – X (treino; teste): "train/X_train.txt" (7352 linhas), "test/X_test.txt" (2947 linhas), ambos com 561 colunas
- Atributo de saída (atividade) – y (treino; teste): "train/y_train.txt" (7352 linhas), "test/y_test.txt" (2947 linhas), ambos com 1 coluna

# Carregamento dos dados

Ler cada um dos ficheiros para um Data Frame usando o package Pandas

```
import pandas as pd
```

```
activities = pd.read_csv('UCI HAR Dataset/activity_labels.txt', sep=' ',
                         header=None, names=('ID','Activity'))
print(activities)
```

```
features = pd.read_csv("UCI HAR Dataset/features.txt", sep = " ",
                       header = None, names=('ID','Sensor'))
print(features.shape)
features.head()
```

```
subjects_tr = pd.read_csv("UCI HAR Dataset/train/subject_train.txt",
                          header = None, names=['SubjectID'])
subjects_tst = pd.read_csv("UCI HAR Dataset/test/subject_test.txt",
                          header = None, names=['SubjectID'])
print(subjects_tr.shape, subjects_tst.shape)
subjects_tr.head()
```

# Carregamento dos dados

Ler cada um dos ficheiros para um Data Frame usando o package Pandas

```
x_train = pd.read_csv("UCI HAR Dataset/train/X_train.txt",
                      sep = "\s+", header = None)
x_test = pd.read_csv("UCI HAR Dataset/test/X_test.txt",
                      sep = "\s+", header = None)
print(x_train.shape, x_test.shape)
```

```
y_train = pd.read_csv("UCI HAR Dataset/train/y_train.txt",
                            header=None, names=['ActivityID'])
y_test = pd.read_csv("UCI HAR Dataset/test/y_test.txt",
                            header=None, names=['ActivityID'])
print(y_train.shape, y_test.shape)
```

# Preparação dos dados

Juntar os conjuntos de dados de treino e teste

```
subjects_all = pd.concat([subjects_tr, subjects_tst], ignore_index=True)
print(subjects_all.shape)
```

```
x_all = pd.concat([x_train, x_test], ignore_index=True)
print(x_all.shape)
```

```
y_all = y_train.append(y_test, ignore_index=True)
print(y_all.shape)
```

Colocar nomes das colunas de X como nomes das features

```
sensorNames = features['Sensor']
x_all.columns = sensorNames
x_all.head()
```

# Preparação dos dados

Substituir códigos de atividade pela designação (string)

```
for i in activities['ID']:
      activity = activities[activities['ID'] == i]['Activity']
      y_all = y_all.replace({i: activity.iloc[0]})
y_all.columns = ['Activity']
y_all.head()
y_all.tail()
```

Juntar tudo num único DataFrame e guardar num CSV

```
x_all = pd.concat([x_all, subjects_all], axis=1)
allXy = pd.concat([x_all, y_all], axis=1)
print(allXy.shape)

allXy.to_csv("HAR_clean.csv")
```

# Agregação dos dados

Criar um conjunto de dados agregado por indivíduo e por atividade

```
import numpy as np
grouped = allXy.groupby (['SubjectID', 'Activity']).aggregate(np.mean)

print(grouped.shape)
grouped.head()

grouped.to_csv("HAR_grouped.csv")
```

# Exploração dos dados

Verificar existência de valores nulos

```
allXy.isnull().sum().sum()
```

Caracterizar distribuição da variável de saída

```
allXy.groupby("Activity").size()
grouped.groupby("Activity").size()
```

Caracterizar distribuição das variáveis de entrada

```
allXy.iloc[:,:-2].mean()
allXy.iloc[:,:-2].max()
allXy.iloc[:,:-2].min()
allXy.iloc[:,:-2].std()
allXy.describe()
```

# Redução de dimensionalidade

Objetivos: dado um conjunto alargado de variáveis, descobrir um **conjunto mais pequeno de variáveis não correlacionadas entre si que explicam a maior parte da variabilidade** dos dados

Em termos de compressão de dados, queremos uma matriz com o menor rank possível, que explique os dados (e os permita reconstruir)

Técnica mais popular: **Análise de componentes principais** (ou **PCA**)

## Análise de componentes principais - PCA

Consta de um procedimento algébrico que **converte as variáveis originais** (tipicamente correlacionadas) num conjunto de **variáveis não correlacionadas** (linearmente) que se designam por **componentes principais (PC) ou variáveis latentes**

Análise baseada na covariância das diversas variáveis

As PCs são ordenadas pela quantidade decrescente de variabilidade (variância) que explicam

## Análise de componentes principais - PCA

Cada PC é gerada de forma a **explicar o máximo de variabilidade** da parte ainda não explicada, tendo que ser **ortogonal** às PCs anteriores

Útil quando há grande número de dados e estes contêm possível redundância

A PCA é sensível à escala dos dados, pelo que se recomenda a sua standardização prévia

## Análise de componentes principais - PCA

PCA fornece mapeamento de um espaço com N dimensões (N – nº variáveis originais) para um espaço com M dimensões (onde M < N)

As coordenadas das observações nas novas variáveis são chamada de ***scores (T)***

As novas dimensões são combinações lineares das variáveis originais, sendo os coeficientes destas no espaço original designados por ***loadings (P)***

Os dados originais (X) são obtidos fazendo $X = T.P^T$

Se considerarmos apenas *k* componentes principais consideramos apenas as primeiras *k* colunas das matrizes T e P e temos uma aproximação dos dados originais

## PCA - exemplo

Standardizar os dados

```
from sklearn import preprocessing
scaled = preprocessing.scale(allXy.iloc[:,:-2])
```

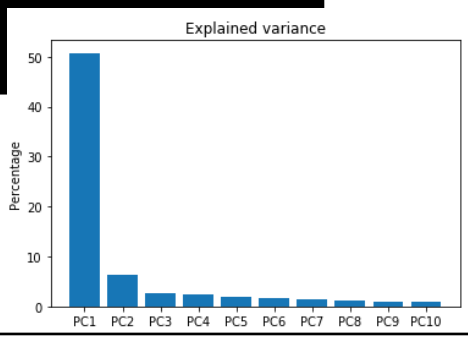
Realizar o PCA

```
from sklearn.decomposition import PCA

n = 10
pca = PCA(n_components=n)
pca.fit(scaled)
X_reduced = pca.transform(scaled)
```

# PCA - exemplo

Variância explicada

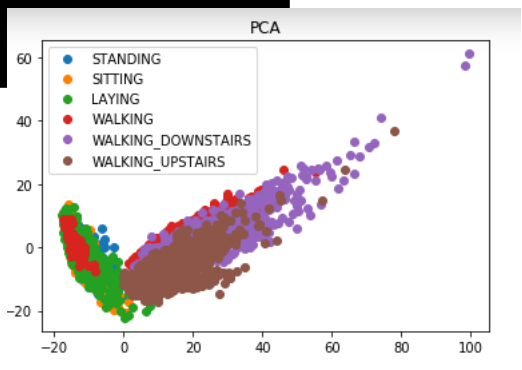
```
import matplotlib.pyplot as plt

print('Var. explained: %s'% str(pca.explained_variance_ratio_))

plt.bar(range(n), pca.explained_variance_ratio_*100)
plt.xticks(range(n), ['PC'+str(i) for i in range(1,n+1)])
plt.title("Explained variance")
plt.ylabel("Percentage")
plt.show()
```




# PCA - exemplo

Scores plot – PC1 + PC2

```
for act in allXy['Activity'].unique():
      sp = allXy.index[allXy['Activity']==act]-1
      plt.plot(X_reduced[sp,0],X_reduced[sp,1],'o',label=act)
plt.title("PCA")
plt.legend(loc='best', shadow=False)
plt.show()
```



# Clustering - definições

Problema na classe de **Aprendizagem Não Supervisionada**

**Objectivo** genérico: **agrupar objetos / entidades / exemplos** (linhas da tabela) com base na sua similaridade

Principais tarefas:

- Como **definir similaridade** ?
- Como **agrupar** elementos baseados nessa similaridade (algoritmos) ?
- Como **visualizar** os agrupamentos ?
- Como **interpretar** os agrupamentos ?

# Clustering - definições

Problema tem diversas variantes/ formulações dependendo de:

- Tipos de **dados** disponíveis (e.g. atributos numéricos vs nominais)
- Formato dos **resultados** desejados:
  - Mapa que define a atribuição de exemplos a clusters – **1 exemplo pertence a 1 cluster**
  - Permitir que um exemplo pertença a mais do que um cluster (**sobreposição**)
  - Atribuir **probabilidades** de pertença: cada par <exemplo, cluster> tem uma probabilidade
  - **Hierarquia** de clusters (com exemplos nas folhas)
- Função **objetivo** para a optimização da partição
- Função de **similaridade** entre os exemplos

# Similaridade (atributos numéricos)

- Por vezes difícil determinar o que é similar ou não !
- Medindo **distâncias**: euclidiana, Manhattan

**Manhattan**

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}|$$

**Euclideana**

$$d(i,j) = \sqrt{(|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + ... + |x_{ip} - x_{jp}|^2)}$$

- Medindo semelhança de formas: coeficientes de **correlação**

# K-Means Clustering

Determina **K** clusters que englobam todos os pontos, de forma a minimizar a média do quadrado das distâncias de cada ponto para o centro do cluster a que pertence d(***V***,***X***):

$$d(\boldsymbol{V},\boldsymbol{X}) = \sum d(v_i, \boldsymbol{X})^2 \ / \ n \qquad 1 \leq i \leq n$$

$d(v_i, \boldsymbol{X})$ refere-se à distância Euclideana entre o ponto $v_i$ e o centro de gravidade do cluster X, a que pertence

# K-Means: Algoritmo de Lloyd

Algoritmo de Lloyd

Gerar aleatoriamente $k$ centros de clusters

**Enquanto** os centros do cluster mudam

- Atribuir a cada ponto um cluster $C_i$ correspondendo ao cluster com centro mais próximo
- Depois de atribuir um cluster a cada um dos $n$ pontos calcular novos centros para os $k$ clusters dados pelo centro de gravidade do cluster

# Características do K-means

Método heurístico eficiente mas **não garante soluções óptimas**

Qualidade da solução final é **dependente da solução inicial** gerada aleatoriamente – optimização **local**

**Repetir algoritmo** com diversas soluções iniciais distintas melhora a sua eficiência

Solução inicial pode ser melhorada criando-se uma distribuição dos centróides que os distribua melhor (i.e. quando se cria cada centróide reduz-se probabilidade de pontos mais próximos)

## Kmeans - exemplo

Correr o clustering

```
from sklearn.cluster import Kmeans

k=6
kmeans_har = KMeans(n_clusters=k, max_iter=1000)
kmeans_har.fit(scaled)
labels = kmeans_har.labels_
centroids = kmeans_har.cluster_centers_
```

Comparar com os grupos “naturais”

```
pd.crosstab(labels, allXy["Activity"], rownames=['clusters'] )
```

## Clustering hierárquico

- Abordagem **aglomerativa**: vai agrupando os objetos, iteração a iteração, criando uma árvore que representa uma hierarquia de clusters
- Estratégia ***bottom up***, construindo árvore das folhas para a raiz
- Folhas da árvore: exemplos/ objetos; nós da árvore: representam possíveis clusters
- Critério de junção baseado na distância entre clusters, i.e. são juntos os clusters mais próximos em cada iteração
- Baseado numa **matriz de distâncias**: onde estão guardadas as distâncias entre todos os pares de objetos; esta matriz é construída aplicando uma métrica de similaridade

## Clustering Hierárquico: Distâncias entre clusters

$d_{min}(C, C^*) = \min d(x,y)$ para todos os elementos $x$ em $C^*$ e $y$ em $C$ – ***SINGLE LINKAGE / NEAREST NEIGHBOUR***

Distância entre 2 clusters é a menor distância entre qualquer par de elementos dos 2 clusters

$d_{min}(C, C^*) = \max d(x,y)$ para todos os elementos $x$ em $C^*$ e $y$ em $C$ – ***COMPLETE LINKAGE***

Distância entre 2 clusters é a maior distância entre qualquer par de elementos dos 2 clusters

$d_{avg}(C, C^*) = (1 / |C^*||C|) \sum d(x,y)$ para todos os elementos $x$ em $C^*$ e $y$ em $C$ – ***AVERAGE LINKAGE***

Distância entre 2 clusters é a média das distâncias entre os pares de elementos dos 2 clusters

## Clustering hierárquico - exemplo

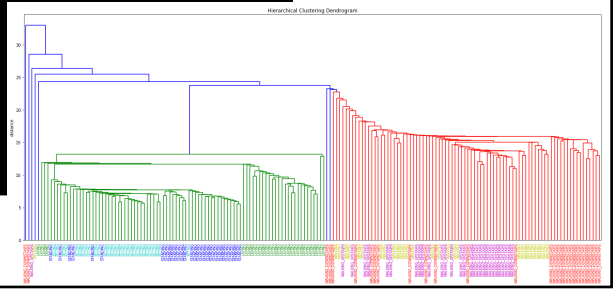
Correr o clustering (para os dados agrupados standardizados)

```
from scipy.cluster.hierarchy import dendrogram, linkage

grouped_sc = preprocessing.scale(grouped.iloc[:,2:])

Z = linkage(grouped_sc, method='single', metric='euclidean')
```

# Clustering hierárquico - exemplo

Visualizar árvore

```
plt.figure(figsize=(25, 10))
dendrogram(Z,
      labels=list(grouped.index.get_level_values(1)),
           leaf_rotation=90., leaf_font_size=8.)
plt.title('Hierarchical Clustering Dendrogram')
plt.ylabel('distance')
lcolors = {'STANDING':'b', "WALKING_UPSTAIRS":"m",
"LAYING":'g', 'SITTING':'c', "WALKING" :"y",
"WALKING_DOWNSTAIRS":"r"}
ax = plt.gca()
xlbls = ax.get_xmajorticklabels()
for lbl in xlbls:
  lbl.set_color(lcolors[lbl.get_text()])
plt.show()
```



# Seleção de atributos

Por variabilidade

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=0.05)
filt_var = sel.fit_transform(allXy.iloc[:,:-2])
print(filt_var .shape)
```

Testes estatísticos univariados

```
from sklearn.feature_selection import SelectPercentile, f_classif
selector = SelectPercentile(f_classif, percentile=30)
filt_univ = selector.fit_transform(allXy.iloc[:,:-2], allXy.iloc[:,-1])
print(filt_univ .shape)
```

# Seleção de atributos

Remover atributos altamente correlacionados

```
corr_matrix = allXy.iloc[:,:-2].corr()
drop_cols = []
threshold_cor = 0.9
for i in range(len(corr_matrix.columns) - 1):
    for j in range(i+1,len(corr_matrix.columns)):
        if j not in drop_cols:
            item = corr_matrix.iloc[i:(i+1), j:(j+1)]
            val = item.values
            if abs(val) >= threshold_cor:
                drop_cols.append(j)

print(drop_cols)
```

```
keep = []
for i in range(len(corr_matrix.columns)):
    if i not in drop_cols:
        keep.append(i)
orig = allXy.iloc[:,:-2]
filt_cor = orig.iloc[:,keep]
print(filt_cor.shape)
```

# Divisão da amostra para aprendizagem máquina

```
from sklearn.model_selection import train_test_split

X_tr, X_ts, y_tr, y_ts = train_test_split(allXy.iloc[:,:-2],
                                allXy.iloc[:,-1], test_size= 0.3)

print(X_tr.shape, y_tr.shape)
print(X_ts.shape, y_ts.shape)

print(y_tr.value_counts()/len(y_tr))
print(y_ts.value_counts()/len(y_ts))
```

# Modelos baseline de aprendizagem máquina

K-vizinhos mais próximos

```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier()
knn_model.fit(X_tr, y_tr)
knn_model.score(X_ts, y_ts)
```

SVM

```
from sklearn import svm

svm_model = svm.SVC(gamma=0.001, C=10.)
svm_model.fit(X_tr, y_tr)
svm_model.score(X_ts, y_ts)
```

# Ensembles

Random forest

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_tr, y_tr)
rf_model.score(X_ts, y_ts)
```

Bagging com árvores de decisão

```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree

bag_model = BaggingClassifier(tree.DecisionTreeClassifier())
bag_model.fit(X_tr, y_tr)
bag_model.score(X_ts, y_ts)
```

# Testar dados com feature selection

Variabilidade

```
X_tr_f, X_ts_f, y_tr_f, y_ts_f = train_test_split(filt_var,
                                    allXy.iloc[:,-1], test_size= 0.3)

svm_model.fit(X_tr_f, y_tr_f)
svm_model.score(X_ts_f, y_ts_f)
```

Correlações

```
X_tr_f2, X_ts_f2, y_tr_f2, y_ts_f2 = train_test_split(filt_cor,
                                       allXy.iloc[:,-1], test_size= 0.3)

svm_model.fit(X_tr_f2, y_tr_f2)
svm_model.score(X_ts_f2, y_ts_f2)
```

# Otimização de hiperparâmetros

Grid search; SVM com kernel Gaussiano

```
from sklearn.model_selection import GridSearchCV

parameters = {'C':[1, 10, 100], 'gamma':[0.01, 0.001]}

svm_model_d = svm.SVC()
opt_model_d = GridSearchCV(svm_model_d, parameters)

opt_model_d.fit(X_tr, y_tr)
print (opt_model_d.best_estimator_)

opt_model_d.score(X_ts, y_ts)
```

Para ser mais rápido, podem realizar com o dataset filtrado por variabilidade !

Modelo final !!

```
opt_model_d.fit(allXy.iloc[:,:-2], allXy.iloc[:,-1])
```