## Seleção de atributos

## Seleção de atributos

Em muitos casos práticos há necessidade/ diversas vantagens em **reduzir o nº de atributos de entrada** de um modelo:

- **complexidade** dos modelos aumenta com o nº atributos de entrada o que pode provocar **sobre-ajustamento**;
- dados e modelos podem ser mais facilmente **analisados** e **compreendidos**;
- eliminação de atributos redundantes ou contraditórios pode melhorar o próprio processo de aprendizagem reduzindo o **ruído**

Processo de escolha do melhor sub-conjunto de atributos de um dado conjunto é um **problema de optimização** que pode tornar-se complexo, dado o espaço alargado de procura

## Seleção de atributos: estratégia

Algoritmos de **filtragem**:

- Seleção de atributos realizada antes do processo de aprendizagem, de forma independente dos classificadores
- Avaliação dos subconjuntos de atributos realizada com medidas estatísticas (e.g. Entropia/ ganho, etc)

Algoritmos **envolventes** (*wrappers*):

- Seleção dos atributos realizada em paralelo com a construção do modelo
- Avaliação dos subconjuntos de atributos realizada treinando um modelo e estimando o seu erro (usando os métodos estudados)

Algoritmos **embebidos** (*embedded*):

- Seleção dos atributos realizada junto com o processo de aprendizagem (e.g. regressão linear com regularização)

## Seleção de atributos: algoritmos de optimização

Heurísticas:

- **Forward selection**: inicia com poucos atributos (e.g. 1) e vai adicionando atributos até atingir um comportamento satisfatório
- **Backward selection**: inicia com todos os atributos e vai removendo

Meta-heurísticas:

- **Algoritmos Evolucionários**
- **Simulated Annealing**

# Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Filtros por variabilidade – remover atributos que variam "pouco" (abaixo de um threshold definido)

```
from sklearn import datasets, svm
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import cross_val_score

digits = datasets.load_digits()
print (digits.data.shape)
sel = VarianceThreshold(threshold=20)
filt = sel.fit_transform(digits.data)
print (filt.shape)
svm_mod = svm.SVC(gamma=0.001, C=100.)
scores= cross_val_score(svm_mod, digits.data, digits.target, cv= 10)
print (scores.mean())
scores_vt= cross_val_score(svm_mod, filt, digits.target, cv= 10)
print (scores_vt.mean())
```

Experimente variar o threshold !

Ver secção 1.13.1 do *User Guide*

# Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Filtros por análise univariada (testes estatísticos) – manter atributos com valores melhores de p-value

```
from sklearn.feature_selection import SelectKBest, chi2, f_classif

filt_kb = SelectKBest(chi2, k=32).fit_transform(digits.data, digits.target)
print (filt_kb.shape)
scores_kb = cross_val_score(svm_model, filt_kb, digits.target, cv = 10)
print (scores_kb.mean())

filt_kb2 = SelectKBest(f_classif, k=32).fit_transform(digits.data, digits.target)
scores_kb2 = cross_val_score(svm_model, filt_kb2, digits.target, cv = 10)
print (scores_kb2.mean())
```

Ver secção 1.13.2 do *User Guide*

# Seleção de atributos

IMPLEMENTAÇÃO EM PYTHON

Wrapper: *recursive feature elimination (RFE)*

```
from sklearn.feature_selection import RFE

svm_model = svm.SVC(kernel = "linear", C=100.)

rfe = RFE(estimator=svm_model, n_features_to_select=32, step=1)

scores_rfe = cross_val_score(rfe, digits.data, digits.target, cv = 10)

print (scores_rfe.mean())
```

Ver secção 1.13.3 do *User Guide*

# Seleção de modelos

IMPLEMENTAÇÃO EM PYTHON

Procura em grelha de parâmetros de SVMs (com validação cruzada na estimação do erro)

```
from sklearn import svm, grid_search, datasets
from sklearn.model_selection import cross_val_score

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 3, 10, 100],
                    'gamma':[0.01, 0.001]}
svm_model_d = svm.SVC( )

opt_model_d = grid_search.GridSearchCV(svm_model_d, parameters)
opt_model_d.fit(digits.data, digits.target)
print (opt_model_d.best_estimator_)
scores_gs = cross_val_score(opt_model_d, digits.data, digits.target, cv = 5)
print (scores_gs.mean())
```

Ver secção 3.2 do *User Guide*

# 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])
```

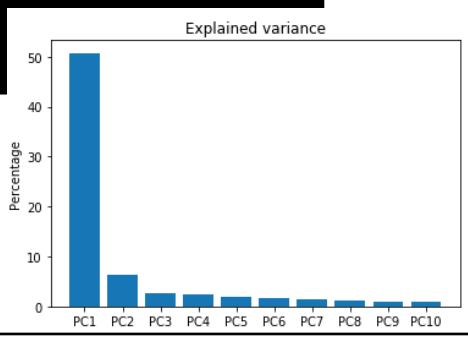
Realizar o PCA

```
from sklearn.decomposition import PCA

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

# PCA - exemplo

Variância explicada

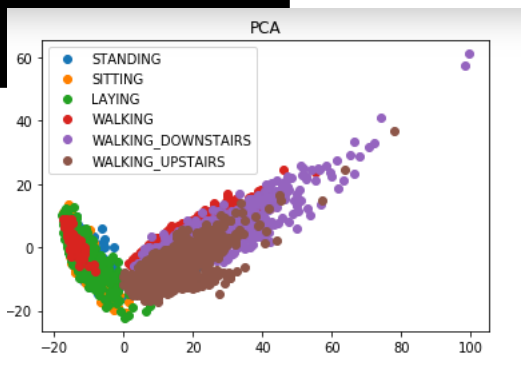
```
import matplotlib.pyplot as plt

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

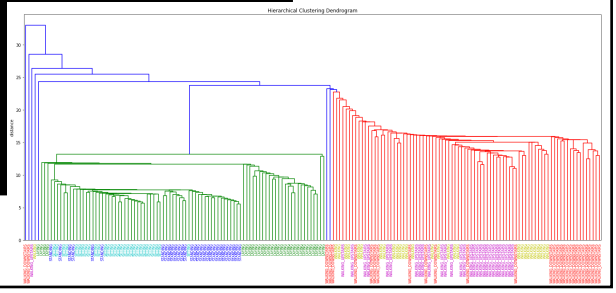
```
for act in allXy['Activity'].unique():
      sp = allXy.index[allXy['Activity']==act]-1
      plt.plot(X_reduced[sp,0],X_reduced[sp,1],'o',label=act)
plt.title("PCA")
plt.legend(loc='best', shadow=False)
plt.show()
```



# Clustering 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))
```