Collège de Bois-de-Boulogne

Cahier thématique

Analyse de dataset et extraction, sélection de prédicteurs avec la méthode PCA

Manipulation 1 : Utilisation de PCA dans une approche de classification

Préparé par : Hafed Benteftifa

Date	Version	Changement
20 juin 2016	1.0	Version initiale

Approche

Dans le cas de l'apprentissage supervisé, un ensemble de prédicteurs doit être choisi. Dans ce cadre, on va procéder à l'exploration du dataset pour établir la pertinence d'utiliser certaines des variables disponibles dans le dataset.

Un pré-traitement pourra aussi être appliqué sur le dataset afin de corriger des valeurs manquantes ou aberrantes.

Jeux de données : Iris (Fisher, 1936)



Iris est un ensemble (jeu) de données introduit en 1936 par Ronald Aylmer Fisher comme un exemple d'analyse discriminante. Cet ensemble contient 150 exemples de critères observées sur 3 espèces différentes d'iris de Gaspésie (Setosa, Versicolor, Verginica).

Chaque exemple est composé de quatre attributs :

1. Longueur des sépales = sepal length en cm
2. Largeur des sépales = sepal width en cm
3. Longueur pétales = petal length en cm
4. Largeur des pétales = petal width en cm
5. Une classe (l'espèce).
    - Iris Setosa
    - Iris Versicolour
    - Iris Virginica

Manipulation 1 : Utilisation de PCA dans une approche de classification

Objectif

Évaluer la pertinence d'utiliser PCA dans la réduction des nombres de caractéristiques (features) lors d'une classification

Démarche

1. On commence par les imports nécessaires pour la manipulation, soit :

Réduction de dimensions et Principal Component Analysis (PCA)

In [1]:
```
# Imports de base
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl
import pandas as pd
from bokeh.plotting import figure,show,output_notebook

output_notebook()
%matplotlib inline
#import pour la classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report

# import pour PCA
from sklearn.decomposition import PCA
```

2. Procéder au chargement du dataset d'interet. Dans notre cas, on prend celui portant sur les fleurs Iris, soit :

In [2]:
```
#Import du dataset Iris
from sklearn.datasets import load_iris
iris = load_iris()
```

3. Afficher le dataset

```
In [27]: #Afficher le dataset
         print (iris)
```

4. On recupère les parties features et target comme suit :

```
In [31]: #Features et target
         X = iris.data
         y = iris.target
         noms_target = iris.target_names
```

5. On choisit maintenant d'effectuer un PCA avec le nombre de composantes limité à 2. On devra aussi valider ce choix par la suite en vérifiant la variance.

```
In [33]: # creer le modele et effectuer un fit du data
         pca = PCA(n_components=2)
         X_resultat = pca.fit(X).transform(X)
```

6. On affiche le resultat

```
In [34]: X_resultat
```

7. On cherche maintenant le pourcentage de variance pris en charge par les 2 composantes PCA. On utilise l'attribut `explained_variance_ratio_` :

```
In [36]: # Pourcentage de variance explained  :
         print ("Composant 1: " + str(pca.explained_variance_ratio_[0]))
         print ("Composant 2: " + str(pca.explained_variance_ratio_[1]))

         Composant 1: 0.924616207174
         Composant 2: 0.0530155678505
```

8. On voit que les 2 composantes prennent en charge presque la totalité de la variance. On procède maintenant au tracé des points du dataset dans le système de coordonnées ou bases PCA. Les labels utilisés sont ceux des targets. On prépare d'abord les points en couleurs.

In [37]:
```
echelle_couleur = {0:'red',1:'blue',2:'orange'}

couleurs = list()

for valeur in y:
    couleur = echelle_couleur[valeur]
    couleurs.append(couleur)
```

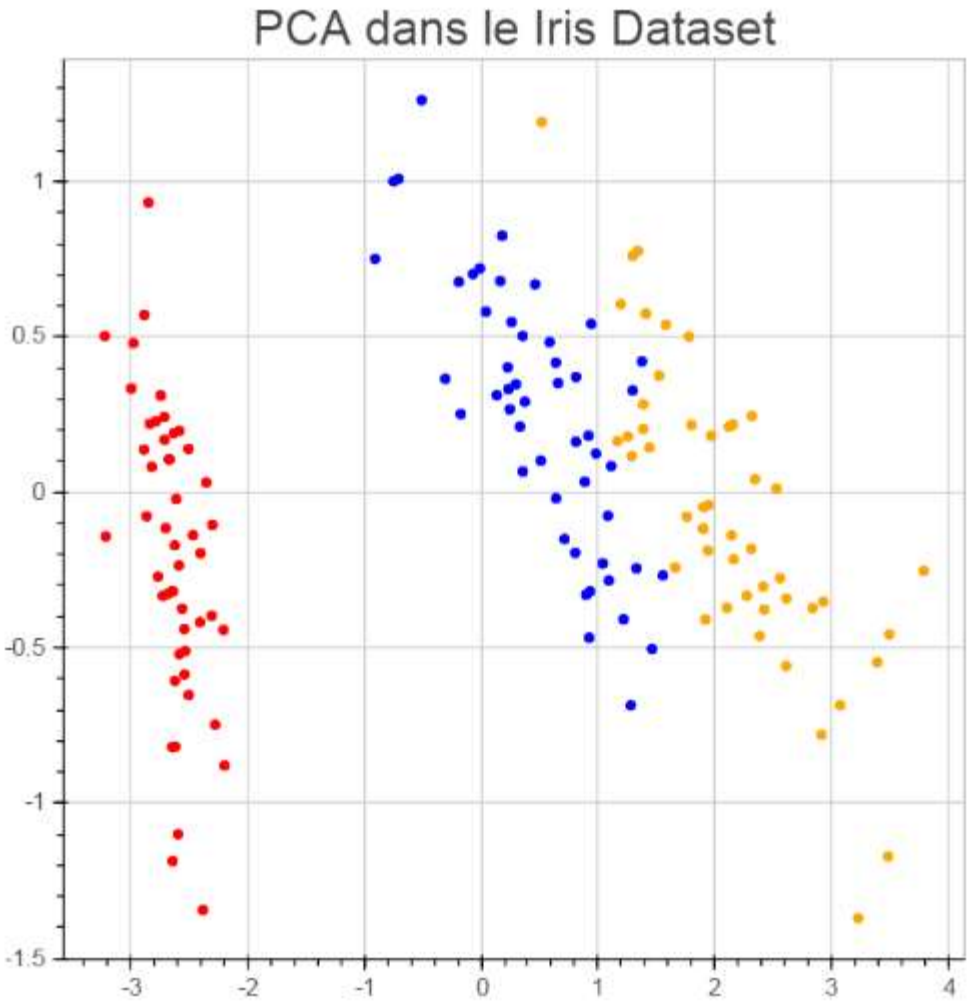
9. On a maintenant les points que l'on peut afficher

In [26]:
```
#Plot comme scatter
p_fig = figure(title="PCA dans le Iris Dataset",tools='')

p1_values =  X_resultat[:,0]
p2_values =  X_resultat[:,1]
p_fig.circle(x = p1_values,y=p2_values,size = 3,color=couleurs)

show(p_fig)
```

10. Cela nous donne :



11. On utilise maintenant une technique simple pour evaluer si le choix de n=2 est adéquat. On refait la même procédure mais avec toutes les composantes:

```
In [40]: # créer le modele mais sans spécifier n_components:
         pca = PCA()
         X_resultat = pca.fit(X).transform(X)
```

12. Afficher les variances expliquées

```
In [49]: variance_explique = pca.explained_variance_ratio_
         print (variance_explique)

[ 0.92461621  0.05301557  0.01718514  0.00518309]
```

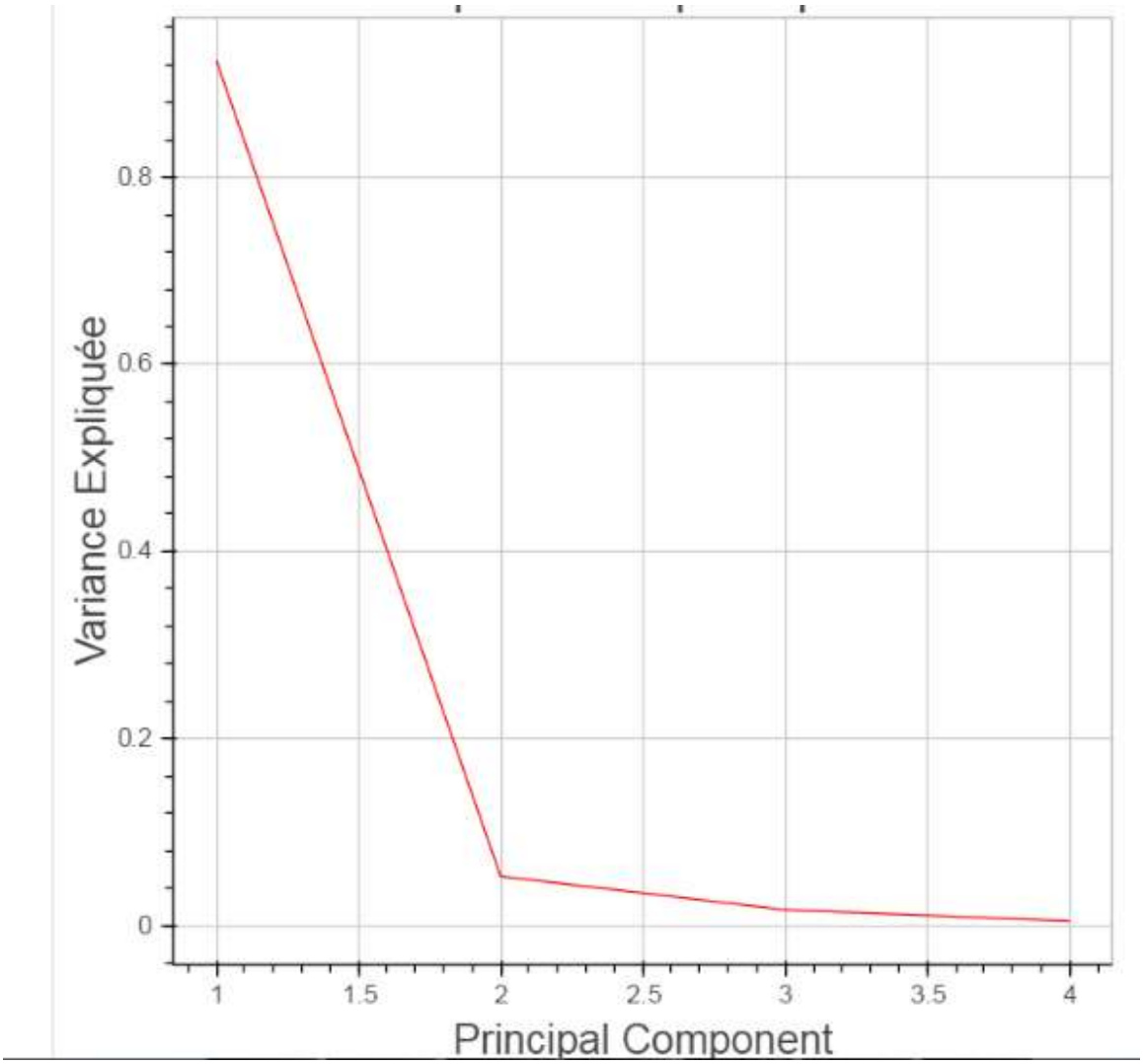
13. Afficher les composantes PCA

In [47]:
```
print (pca.components_)
```

```
[[ 0.36158968 -0.08226889  0.85657211  0.35884393]
 [-0.65653988 -0.72971237  0.1757674   0.07470647]
 [ 0.58099728 -0.59641809 -0.07252408 -0.54906091]
 [ 0.31725455 -0.32409435 -0.47971899  0.75112056]]
```

14. Afficher le graphe qui nous permet de voir les variances expliquées par rapport aux nombre de composantes

In [51]:
```
comp_id = [1, 2, 3, 4]              # id du composant
p_pca = figure(title="Composantes principales",tools='',
           x_axis_label="Principal Component",
          y_axis_label='Variance Expliquée')
p_pca.line(comp_id,variance_explique,color='red')
show(p_pca)
```



On voit qu'avec 2 composantes, on est capable de capturer l'essentiel des features.

15. On évalue maintenant si la vitesse de traitement est améliorée en prenant moins de feature ou plus de features

```
In [52]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
```

16. Pour simplifier, on utilise la fonction `%%timeit` pour effectuer une classification selon la sélection de features. L'algorithme de classification utilisé est le KNN

```
In [53]: %%timeit
         modele = KNeighborsClassifier(2)
         modele.fit(X_train, y_train)

         1000 loops, best of 3: 279 µs per loop
```

17. On peut afficher un rapport des mesures de classification après un test comme suit :

```
In [55]: modele = KNeighborsClassifier(2)
         modele.fit(X_train, y_train)
         print (classification_report(y_test,model.predict(X_test)) )
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
avg / total	1.00	1.00	1.00	30

18. Il reste maintenant à effectuer la même approche mais avec un nombre de features réduit selon PCA. On commence par préparer notre data pour le training et le test selon la transformation PCA.

```
In [27]: # refaire maintenant avec 2 composantes principales
         # creer le modele et effectuer un fit du data
         pca_2 = PCA(n_components=2)
         X_resultat_2 = pca.fit(X).transform(X)
         X_r_train2, X_r_test2, y_r_train2, y_r_test2 = train_test_split(X_resultat_2,y,
                                                                         test_size=0.2, random_state=42)
```

19. Développer le modèle selon le data de training, soit :

```
In [25]: # Effectuer le training avec KNN
         %%timeit
         modele = KNeighborsClassifier(2)
         modele.fit(X_r_train2, y_r_train2)
```

20. On peut ensuite afficher le rapport de sortie des métriques, soit :

In [26]:
```
modele2 = KNeighborsClassifier(2)
modele2.fit(X_r_train2, y_r_train2)
print (classification_report(y_r_test2,modele2.predict(X_r_test2)) )
```