Index

Objectius

L'objectiu d'aquesta prova d'avaluació és l'aplicació de tècniques d'extracció de característiques i de classificació sobre dades de persones classificades segons el seu estat del cor, en un total de 5 categories. Per la qual cosa estarem resolent un problema multi-classe (versus binari o dos classes).

Descripció de la PAC

En aquesta prova ens familiaritzarem amb els mètodes d'extracció i selecció de característiques (mòdul 3) i mètodes d'aprenentatge supervisat (classificadors) d'acord amb el que s'explica al mòdul 4.

L'arxiu de dades conté informació sobre 303 persones. Cada persona està etiquetada amb una classe que pot prendre fins a 5 valors:
- 0: no hi ha cap malaltia del cor
- 1: malaltia simple en els vasos sanguini
- 2: malaltia doble en els vasos sanguinis
- 3: malaltia triple en els vasos sanguinis
- 4: malaltia en l'arteria principal coronaria

La URL per descarregar les dades és http://archive.ics.uci.edu/ml/datasets/heart+disease

De fet, trobareu fins a 4 conjunts de dades, en funció de l'hospital que les ha recollit: Cleveland, Hungary, Switzerland, i VA Long Beach. A més a més, hi ha dos versions de les dades, de 75 variables i de 13 variables. En el darrer cas, el procés de reducció de les variables ha estat realitzat per experts en el domini. En aquesta PAC, per simplicitat, treballarem amb la versió processada de les dades Cleveland: 13 variables. L'arxiu corresponent és "**processedCleveland.csv**".

En la URL trobareu més detalls de les explicacions de les variables, així com diferents articles que les descriuen. Una de les referències citades on expliquen millor les dades és:
Liping Wei and Russ B. Altman. An Automated System for Generating Comparative Disease Profiles and Making Diagnoses. Section on Medical Informatics Stanford University School of Medicine, MSOB X215.

Requeriments:
• Matplotlib
https://matplotlib.org/users/installing.html
python –mpip install –U pip
python –mpip install –U matplotlib
• Scikit-Learn
http://scikit-learn.org/stable/install.html
pip install -U scikit-learn
• Scipy
https://scipy.org/install.html
pip install -U scipy
• Pandas
https://pandas.pydata.org/pandas-docs/version/0.20/install.html
pip install pandas

Draft code:
Es proporciona un codi inicial on teniu el mètode LoadAndPreprocess(...) per llegir l'arxiu, i fer un petit pre-processament, amb la finalitat d'eliminar els valors desconeguts i escalar les dades. Retorna dos matrius de dades: els exemples, i les classes a la qual pertany cada exemple.
El codi també està pensat perquè desenvolupeu el que se us demana en cadascun dels exercicis descrits aquí sota.

Exercici 1 (2 punts)

Apliqueu una anàlisi PCA a les dades de la PAC:
a) Quantes components principals són necessàries per representar un 95% de la variància de les dades originals? Raoneu la resposta tenint en compte els diferents arxius disponibles a la URL.

http://archive.ics.uci.edu/ml/datasets/heart+disease
Cleveland: 13 variables. L'arxiu corresponent és "**processedCleveland.csv**".

Els valors d'aplicar PCA indiquen la quantitat de variància i la suma ha de donar 1 que equival al 100%. Si sumen fins arribar a 0.95 (95%) o més tenim el número de components requerits. En el nostre cas fan falta 12 amb un total de 0.97 (95%) perquè amb 11 obtenim 0.94 (94%).

Activity 1a
[0.23695056 0.36044542 0.45648373 0.54164513 0.61802636 0.68522878
0.75016633 0.8101108 0.86279714 0.9064821 0.94134612 0.97275081
1.]

Tenint en compte que les 13 variable del fitxer Cleveland son les més representatives de les 75 variables inicials i per obtenir el 95% de variància es necessari quasi tots els components.

b) Reconstruïu el conjunt de dades a partir de les 2, 4 i 8 components principals (mitjançant el mètode *inverse_transform*), i calculeu la pèrdua d'informació respecte al conjunt original per a cada cas. Per a fer-ho, podeu calcular la mitjana de les diferències elevades al quadrat entre cada element del conjunt reconstruït i l'original. Quina relació tenen aquests valors respecte a les variàncies acumulades calculades a l'apartat anterior?

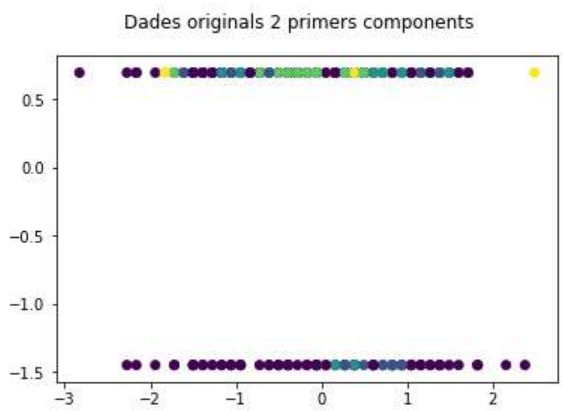
Després transformar i reconstruir el conjunt per a 2, 4 i 8 components, calculem la pèrdua amb l'error quadràtic mitjà (MSE).

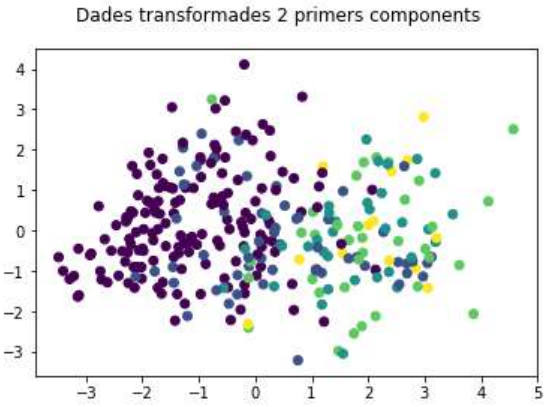
MSE amb 2 Components 0.639554578279
MSE amb 4 Components 0.458354872999
MSE amb 8 Components 0.189889204271

A mida que s'agafen més components la pèrdua de dades respecte a les dades originals es menor i està relacionat amb que quants més components més fidel es la representació de les dades originals, per exemple 8 components que representen el 81% quan es reconstrueix representa un error aproximat del 19% sobre les dades originals. La suma del % que representa i l'error dona el 100% que representen les dades originals.

c) Visualitzar les dades original (2 primers variables) i les dades transformades segons els 2 components principals. Comenteu el que veieu.

He visualitzat les dades originals utilitzant les dues primeres variables i un altre la transformada per als dos primers components. A la gràfica de dades originals es veuen dos línies però a la gràfica després d'aplicar PCA amb dos components i transformar s'aprecia una millora on es veuen agrupacions.





Exercici 2 (2 punts)

En aquest exercici compararem els resultats d'aplicar un classificador simple amb les dades originals i les dades d'acord amb les característiques obtingudes pel PCA.

a) Aplicar KNeighborsClassifier, amb k=3, sobre el conjunt de dades originals

Apliquem KNeighborsClassifier amb 3 veïns (k=3) sobre les dades originals (attributes).

Codi:

```
### Activity 2a ####
k = 3
n_splits = 5
score_source = []

kf = KFold(n_splits)
kf.split(attributes)
knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean')

for X_train, X_test in kf.split(attributes):
  x_train,x_test = numpy.array(attributes)[X_train], numpy.array(attributes)[X_test]
  y_train,y_test = numpy.array(classes)[X_train],numpy.array(classes)[X_test]

  knn.fit(x_train, y_train)
  score_source.append(knn.score(x_test, y_test))

print("2.a: Score Conjunt Original:", numpy.mean(score_source))
```

b) Aplicar KNeighborsClassifier, amb k=3, sobre el conjunt de dades transformades pel PCA amb 8 components

Apliquem KNeighborsClassifier amb 3 veïns (k=3) sobre les dades transformades de PCA que prèviament aplicat classificador PCA amb 8 components i entrenat.

Codi:

```
### Activity 2b ####
score_transformada = []

pca_n8 = PCA(n_components=8)
pca_n8.fit(attributes,classes)
X_pca_n8 = pca_n8.transform(attributes)

for X_train, X_test in kf.split(attributes):
  xt_train,xt_test = numpy.array(X_pca_n8)[X_train],numpy.array(X_pca_n8)[X_test]
  y_train,y_test = numpy.array(classes)[X_train], numpy.array(classes)[X_test]
```

```
    knn.fit(xt_train, y_train)
    score_transformada.append(knn.score(xt_test, y_test))

print("2.b: Score Conjunt Transformat:", numpy.mean(score_transformada))
```

c) Comentar els resultats obtinguts, i la implicació respecte a la reducció de la dimensionalitat amb datasets complexos.

Activity 2
2.a: Score Conjunt Original: 0.565649717514
2.b: Score Conjunt Transformat: 0.575536723164

La reducció de dimensionalitat ajuda a eliminar informació que es pot prescindir, com els components menys representatius i soroll. L'aplicació de PCA amb 8 components mostra una millora respecte a l'original. S'ha comprovat que encara s'obtindríem millors resultats (0.5824) si utilitzéssim 7 components.

Exercici 3 (3 punts)

En aquest exercici treballarem amb els següents classificadors de scikit-learn, amb els paràmetres indicats a continuació, per tal d'obtenir l'*score*, el *training time*, i el *prediction time* quan l'apliquem a les dades d'aquesta PAC.

- k Nearest Neighbors (mòdul KneighborsClassifier de sklearn.neighbors): amb 3, 4 i 5 veïns (primer paràmetre).
- Linear SVM (mòdul SVC de sklearn.svm): *kernel="linear"*, *C=0.025*, la resta de paràmetres, valor per defecte.
- Decision Tree (mòdul DecisionTreeClassifier de sklearn.tree): criterion='entropy', *max_depth=5*, la resta de paràmetres, valor per defecte.
- AdaBoost (mòdul AdaBoostClassifier de sklearn.ensemble): paràmetres per defecte.
- Gaussian Naive Bayes (mòdul GaussianNB de sklearn.naive_bayes): paràmetres per defecte.

El classificador amb millor score es Linear SVM seguit de KNN amb 5 veïns. Els més ràpids al entrenar han sigut els KNN, seguit de Gaussian Naive Bayes que ha trigat el doble de temps. En quan a temps de predicció el més ràpid amb diferència es Decission Tree amb 5,7e-05 seguit de linear SVC amb 23.3e-05.

Score Result
0.555555555556 KNN 3 Neigbours
0.552188552189 KNN 4 Neigbours
0.56228956229 KNN 5 Neigbours
0.582491582492 Linear SVM
0.494949494949 Decission Tree
0.498316498316 AdaBoost
0.508417508418 Gaussian Naive Bayes

Training Time
0.00039082613147911616 KNN 3 Neigbours
0.0002785290259150012 KNN 4 Neigbours
0.0002790321474700856 KNN 5 Neigbours
0.0011026448895184633 Linear SVM
0.0010653131442571369 Decission Tree
0.05375157364323968 AdaBoost
0.0006104897365730722 Gaussian Naive Bayes

Prediction Time
0.0007910103546843553 KNN 3 Neigbours
0.000748043630058722 KNN 4 Neigbours
0.000792318476063277 KNN 5 Neigbours
0.00023354980779307274 Linear SVM
5.7154799530205004e-05 Decission Tree
0.0037019807859905995 AdaBoost
0.00021312300426264605 Gaussian Naive Bayes

Exercici 4 (3 punts)

El data set proporcionat està des balancejat. Amb el mètode que hàgiu obtingut millor resultat en l'apartat anterior:

a) Compareu el resultat de la mesura per defecte i la precision_score amb el paràmetre average="macro". Com es veuen afectat els resultats.

Segons sklearn els ratios de les dues mesures son els següents:

Accuracy_score = TP+TN/TP+FP+FN+TN
Precision_score = TP/TP+FP

TP: number of true positives, predicció encerta que es de la classe.
FP: number of false positives, predicció no encerta que es de la classe.
TN: number of true negatives, predicció encerta que no es de la classe.
FN: number of false negatives, predicció no encerta que no es de la classe.

Accuracy_score calcula la mitja dels resultats positius entre el total, en canvi precisions_score calcula la mitja de les observacions positives predites correctament del total d'observacions positives predites tant verdaderes com falses. El paràmetre average macro calcula la mitja de les mètriques binaries, assignant el mateix pes a cadascuna de les classes i per tant en escenaris amb classes no importants, els resultats tendiran a baixar degut a aquestes classes.

Tenint en compte que precision es inferior al average, vol dir que la mitja d'encerts positius es inferior a la mitja d'encerts negatius.

Resultat 4.a (Kfold)
Linear SVM
0.582491582492 Accuracy Score
0.303826568761 Precision Score

b) Ara compareu els resultats depenent de si useu kFold o StratifiedKfold. Quina és l'opció vàlida?

Els resultats amb Kfold i StratifiedKfold son els següents:

4.a (Kfold)
Linear SVM
0.582491582492 Accuracy Score
0.303826568761 Precision Score

4.b (StratifiedKfold)
Linear SVM
0.565671687444 Accuracy Score
0.235362522468 Precision Score

Degut a que StrtifiedKfold al partir el conjunt de dades original de forma que % de classes del conjunt original es mantingui per a cadascun dels trossos, això afecta al entrenar i per tant a la precisió de les prediccions. Recomano utilitzat Kfold degut al millor resultat tant d'accuracy com precision.