# Practica 4 Clasificación Avanzada

February 9, 2021

### 0.0.1 Ejercicio 1: Seleccione tres algoritmos clasificación de los disponibles scikit-learn

Algoritmos escogidos: - KNN - SVM - DTC

[4]:
```
# -*- coding: utf-8 -*-
from scipy.io import arff
import numpy as np
import pandas as pd
from os import listdir
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingRegressor,␣
↪GradientBoostingClassifier, BaggingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from scipy.stats import wilcoxon,rankdata,f,friedmanchisquare
import warnings
warnings.filterwarnings('ignore')

lista_datasets = listdir('./Datos/P4/')
matrix_ccr   = [['---',0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],['KNN',0.
↪0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],['SVM',0.0,0.0,0.0,0.0,0.0,0.0,0.
↪0,0.0,0.0,0.0,0.0],['DTC',0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]]


def iniciarM():
    c = 1
    for i in lista_datasets:
        matrix_ccr[0][c] = i
        c = c + 1

def imanDavenport():
    print('Test de Iman Davenport')
    nDatasets=10
    kAlgoritms=3
```

```
    chi=friedmanchisquare(matrix_ccr.iloc[1,1:],matrix_ccr.iloc[2,1:
↪],matrix_ccr.iloc[3,1:])
    F=((nDatasets-1)*chi[0])/(nDatasets*(kAlgoritms-1)-chi[0])
    RESULT = f.ppf(q=F, dfn=kAlgoritms-1, dfd=(kAlgoritms-1)*(nDatasets-1))
    print('Valor de F: '+str(F)+' , valor de RESULT: '+str(RESULT))
    if F < RESULT:
        print('No hay diferencias significativas')
    elif F > RESULT:
        print('Si hay diferencias significativas')

def base():
    AUX = 1
    print('Aplicando método base:\n')
    for indice in lista_datasets:
        print('\tDataset: ' + str(indice))
        dataset = arff.loadarff('./Datos/P4/' + str(indice))
        df = pd.DataFrame(dataset[0])

        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]

        # entrenamiento del algoritmo KNN
        print('\tPuntuacion KNN:\n')
        knn = KNeighborsClassifier(n_neighbors=5)
        cv_scores = cross_val_score(knn, input, output, cv=10)
        print('\t\t\t'+str(cv_scores))
        matrix_ccr[1][AUX] = np.mean(cv_scores)*100
        print('\t----------------')
        # entrenamiento algoritmo SVM
        print('\tPuntuacion SVM:\n')
        svm = SVC(gamma='auto')
        cv_scores = cross_val_score(svm, input, output, cv=10)
        print('\t\t\t'+str(cv_scores))
        matrix_ccr[2][AUX] = np.mean(cv_scores)*100
        print('\t-----------------')
        # entrenamiento del arbol de decision
        print('\tPuntuacion TREE:\n')
        arbol = DecisionTreeClassifier()
        cv_scores = cross_val_score(arbol, input, output, cv=10)
        print('\t\t\t'+str(cv_scores))
        matrix_ccr[3][AUX] = np.mean(cv_scores)*100
        print('\t-----------------------------------------------\n')
        AUX = AUX + 1

def bagging():
    print('Aplicando metodo de combinacion BAGGING\n')
    for indice in lista_datasets:
```

```
        print('Dataset: ' + str(indice))
        dataset = arff.loadarff('./Datos/P4/' + str(indice))
        df = pd.DataFrame(dataset[0])

        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]

        for ESTIMADOR_BASE in [KNeighborsClassifier(n_neighbors=5),␣
↪SVC(gamma='auto'), DecisionTreeClassifier()]:
            kfold = model_selection.KFold(n_splits=10)
            model = BaggingClassifier(base_estimator=ESTIMADOR_BASE)
            results = model_selection.cross_val_score(model, input, output,␣
↪cv=kfold)
            print('\n')
            print(results)
            print('\n')

def boosting_r():
    print('Algoritmo de BOOSTING: Regressor\n')
    for indice in lista_datasets:
        print('\tDataset: ' + str(indice))
        dataset = arff.loadarff('./Datos/P4/' + str(indice))
        df = pd.DataFrame(dataset[0])

        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]

        kfold = model_selection.KFold(n_splits=10)

        model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,␣
↪max_depth=1, random_state=0, loss='ls')
        RE = model_selection.cross_val_score(model, input, output, cv=kfold)

        print('\n')
        print(RE)
        print('\n')

def boosting_c():
    print('Algoritmo de BOOSTING: Classifier')
    for indice in lista_datasets:
        print('Dataset: ' + str(indice))
        dataset = arff.loadarff('./Datos/P4/' + str(indice))
        df = pd.DataFrame(dataset[0])

        input = df.iloc[:, df.columns != 'class']
        output = pd.factorize(df['class'])[0]
```

```
        kfold = model_selection.KFold(n_splits=10)

        model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,␣
↪max_depth=1, random_state=0)
        RE = model_selection.cross_val_score(model, input, output, cv=kfold)

        print('\n')
        print(RE)
        print('\n')


if __name__ == '__main__':

    iniciarM()
    value = int(input('Metodos disponibles: \n \t\t -> (base) Aplicar metodo␣
↪base \n \t\t -> (bagging) Aplicar Bagging \n \t\t -> (boosting_r) Aplicar␣
↪Boosting Regresion \n \t\t -> (boosting_c) Aplicar Boosting Clasificacion'))

    if value == 1:
        base()
        matrix_ccr = pd.DataFrame(matrix_ccr)
        imanDavenport()
    if value == 2:
        bagging()
    if value == 3:
        boosting_r()
    if value == 4:
        boosting_c()
```

```
Metodos disponibles:
                -> (base) Aplicar metodo base
                -> (bagging) Aplicar Bagging
                -> (boosting_r) Aplicar Boosting Regresion
                -> (boosting_c) Aplicar Boosting Clasificacion2
Aplicando metodo de combinacion BAGGING

Dataset: 0diabetes.arff


[0.63636364 0.75324675 0.66233766 0.64935065 0.75324675 0.72727273
 0.74025974 0.77922078 0.73684211 0.73684211]




[0.58441558 0.71428571 0.55844156 0.61038961 0.64935065 0.61038961
 0.81818182 0.67532468 0.68421053 0.60526316]
```

```
[0.7012987  0.84415584 0.7012987  0.66233766 0.77922078 0.79220779
 0.76623377 0.83116883 0.68421053 0.76315789]


Dataset: 1glass.arff


[0.54545455 0.63636364 0.68181818 0.77272727 0.71428571 0.38095238
 0.76190476 0.66666667 0.76190476 0.80952381]




[0.5        0.54545455 0.54545455 0.5        0.80952381 0.52380952
 0.80952381 0.71428571 0.66666667 0.66666667]




[0.68181818 0.72727273 0.63636364 0.68181818 0.9047619  0.47619048
 0.85714286 0.76190476 0.80952381 0.71428571]


Dataset: 2ionosphere.arff


[0.72222222 0.82857143 0.68571429 0.74285714 0.8        0.82857143
 0.71428571 1.         0.97142857 0.97142857]




[0.86111111 0.91428571 0.82857143 0.82857143 0.85714286 1.
 0.88571429 1.         1.         0.97142857]




[0.75       0.91428571 0.88571429 0.85714286 0.77142857 0.91428571
 0.94285714 1.         0.97142857 0.94285714]


Dataset: 3ris.arff
```

```
[1.         1.         1.         1.         0.8        0.86666667
 1.         0.86666667 0.73333333 1.        ]
```

```
[1.         1.         1.         1.         0.86666667 0.86666667
 1.         0.93333333 0.86666667 0.93333333]
```

```
[1.         1.         1.         1.         0.93333333 0.86666667
 1.         0.86666667 0.8        1.        ]
```

```
Dataset: 4cpu.arff
```

```
[0.03333333 0.         0.         0.03333333 0.         0.
 0.63333333 0.62068966 0.86206897 0.86206897]
```

```
[0.06666667 0.06666667 0.         0.         0.         0.
 0.         0.         0.         0.        ]
```

```
[0.         0.         0.06666667 0.         0.03333333 0.03333333
 0.9        0.93103448 0.86206897 0.93103448]
```

```
Dataset: 5contactLenses.arff
```

```
[1.         1.         0.66666667 0.33333333 1.         0.5
 0.5        0.5        0.5        1.        ]
```

```
[0.66666667 0.33333333 0.66666667 0.33333333 0.5        0.5
 0.5        0.5        0.5        1.        ]
```

```
[1.         1.         0.66666667 1.         1.         0.5
 0.5        0.5        1.         0.5       ]
```

```
Dataset: 6segment-challenge.arff
```

```
[0.96       0.91333333 0.95333333 0.96       0.94       0.91333333
 0.92       0.91333333 0.92       0.96666667]
```

```
[0.59333333 0.46666667 0.48       0.6        0.56       0.52666667
 0.5        0.5        0.53333333 0.54666667]
```

```
[0.95333333 0.96       0.98       0.96       0.98       0.96666667
 0.96       0.98       0.97333333 0.95333333]
```

```
Dataset: 7segment-test.arff
```

```
[0.91358025 0.88888889 0.87654321 0.9382716  0.85185185 0.9382716
 0.83950617 0.90123457 0.95061728 0.91358025]
```

```
[0.35802469 0.33333333 0.39506173 0.38271605 0.34567901 0.41975309
 0.33333333 0.41975309 0.35802469 0.45679012]
```

```
[0.95061728 0.9382716  0.95061728 0.96296296 0.95061728 0.96296296
 0.9382716  0.90123457 0.95061728 0.95061728]
```

```
Dataset: 8weather.arff
```

```
[0.4 1.  1.  1.  1.  1.  1.  1.  1.  1. ]
```

```
[0.6 1.  1.  1.  1.  1.  1.  1.  1.  1. ]
```

```
[0.4 1.  1.  1.  1.  1.  1.  1.  1.  1. ]
```

```
Dataset: 9iris.arff
```

```
[1.         1.         1.         1.         0.86666667 0.86666667
 1.         0.86666667 0.8        1.        ]
```

```
[1.         1.         1.         1.         0.86666667 0.86666667
 1.         0.93333333 0.86666667 0.93333333]
```

```
[1.         1.         1.         1.         0.93333333 0.86666667
 1.         0.86666667 0.8        0.93333333]
```

### 0.0.2 Ejercicio 2: Para cada uno de los tres métodos realize los siguientes pasos usando validación cruzada de 10 particiones:

**2.1 Aplique el método base a cada uno de los tres conjuntos y anote los resultados obtenidos.**

**2.2 Aplique el método de combinación de clasificadores Bagging a cada unos de los clasificadores y anote los resultados.**

**2.3 Seleecione dos algoritmos de Boosting y aplique estos algoritmos a cada uno de los conjuntos y anote los resultados obtenidos.**

**2.4 Compare si hay diferencias significativas entre ellos usando el test de Iman-Davenport. Si es así, aplique el procedimiento de Wilcoxon para comparar cada método de agrupación con el clasificador base.** El metodo base es encargado de hacer

la cross-validation, la forma más sencilla de utilizar la validación cruzada es una llamada a la función auxiliar cross_val_score en el estimador y el conjunto de datos.

El metodo bagging es un metaestimador de conjunto que se ajusta a los clasificadores de base en subconjuntos aleatorios del conjunto de datos original y luego agrega sus predicciones individuales (ya sea votando o promediando) para formar una predicción final. Tal metaestimulador puede usarse típicamente como una forma de reducir la varianza de un estimador de caja negra (por ejemplo, un árbol de decisión), al introducir la aleatorización en su procedimiento de construcción y luego hacer un conjunto de él.

El método de boosting o aumento de gradiente para regresión,construye un modelo aditivo de manera progresiva por etapas; Permite la optimización de funciones arbitrarias de pérdida diferenciable. En cada etapa se ajusta un árbol de regresión en el gradiente negativo de la función de pérdida dada.

El método de boosting para clasificación construye un modelo aditivo de manera progresiva por etapas; Permite la optimización de funciones arbitrarias de pérdida diferenciable.En cada etapa se ajusta un árbol de regresión en el gradiente negativo de la función de pérdida dada.

La gran diferencia entre las técnicas de bagging y validación es que los modelos promedios de bagging es que son usados para reducir la varianza a la que está sujeta la predicción, mientras que el muestreo de validación como la validación cruzada y la validación fuera de la rutina evalúan un número de modelos sustitutos suponiendo que son equivalentes (es decir, un buen sustituto) para el modelo real en cuestión que está entrenado en todo el conjunto de datos.

### 0.0.3 Ejercicio 3: Enuncie las conclusiones del estudio indicando la influencia del clasificador base en el rendimiento de las agrupaciones de clasificadores.

Se han aplicado métodos de ensemble, en este caso han sido Bagging y GradientBoosting. Estos métodos utilizan múltiples algoritmos de aprendizaje, y el objetivo es mejorar el rendimiento que dichos algoritmos obtienen de manera individual. Lo que ya de entrada nos da a suponer que el tiempo de computo será mucho mayor, así que no en todos los casos sería recomendable realizar un ensemble.

En la metodología de Bagging este tomara un clasificador base especificado por el usuario programador, y aparte de este clasificador tambien es necesario un conjunto de parametros. Es muy importante que al usar este método se estimen bien los parametros a usar y sobre todo el clasificador base, pues de esto dependera un buen clasificador en el futuro.

Por otro lado, el própio clasificador Bagging tiene sus própios parámetros que también deben estudiarse para optimizarlos según el problema. Con lo cual, podría ser una buena opción para mejorar el rendimiento (debido sobre todo a que este método tiende a reducir el sobreentrenamiento), aunque requiriendo bastante estudio por parte del programador.

En los otros métodos hemos utilizado GradientBoostingRegressor y GradientBoostingClasifier, es muy importante saber cual de los dos utilizar según el problema ya que la puntuación obtenida cambiará según nos enfrentemos a clasifiación o regresión.

Comparato con los métodos de Bagging, aquí el clasificador base se construye de forma lineal, con lo que el sesgo del ensemble se va reduciend y por ello esta metodología funciona mejor combinando modelos menos complejos como resultado se obtiene un modelo muy fuerte.

Los algoritmos de Boosting se dedican a construir un gran árbol en cada una de las iteraciones tomando por referencía los errores encontrados en cada una de las iteraciones realizadas, para estos metodos hay que tener clara la identificación de los valores perdidos pues suponen un gran problema para estas metodologias pues se fijan mucho en estos valores tornando así unos valores inadecuados, para que esto no ocurra se debe de realizar un buen preprocesamiento e identificación de estos valores.