**06. Aprendizaje automático (Machine Learning): conceptos, metodología, algoritmos de aprendizaje para analítica descriptiva, predictiva y prescriptiva, y retos en su implementación (2022-24)**

**Homework 2 - KNN (obligatory)**

First homework that the students will be asked to complete during the descriptive and predictive modules.

**Module 6: Descriptive and Predictive Modeling**

***Exercise 1: K-Nearest Neighbors in depth***

Based on the Python code examples used in class, groups are asked to choose one of the following data sets and describe what their fields represent:

1. "from sklearn.datasets import load\_breast\_cancer" (basic)
2. **"from sklearn.datasets import load\_digits" (advanced)**

Departing from this selection, groups must address the following questions in order:

1. **Describe the dataset in dimensions such as number of features, number of categories, and number of samples per category using Python.**

The 'load\_digits' dataset is loaded and the number of samples, features, and categories in the dataset are obtained and printed.

===================================================================

# MANDATORY KNN HW 2

# load data =================================================================

from sklearn.datasets import load\_digits

digits = load\_digits()

# =============================================================================

# Obtener información sobre el conjunto de datos

# =============================================================================

n\_samples = len(digits.images)

n\_features = digits.data.shape[1]

n\_categories = len(digits.target\_names)

# print(digits)

print("Number of samples: %d" % n\_samples)

print("Number of features: %d" % n\_features)

print("Number of categories: %d" % n\_categories)

Texto

Descripción generada automáticamente

1. **Represent the statistical support of every feature graphically, resorting to Matplotlib’s boxplot function. Are there any outliers that can be detected by simple visual inspection? If so, devise a handcrafted method to detect and isolate such examples.**

A boxplot of the dataset is created to show the statistical support of every feature. Next the outliers in the dataset are identified and printed.

=============================================================

# Statistical support of every feature graphically

=============================================================

import matplotlib.pyplot as plt

X, y = load\_digits(return\_X\_y=True)

plt.boxplot(X)

plt.show()

Gráfico

Descripción generada automáticamente

To perform the outlier detection, we use the interquartile range **(IQR) method**.

This method first uses the **numpy** library to calculate the **first and third quartiles** of the data X along the specified axis (0 in this case).

Then **calculates the median** of the data using numpy too.

The fourth line **calculates the interquartile range (IQR)** by subtracting the first quartile from the third quartile.

Afterwards, it **calculates the upper and lower bounds** for identifying outliers, respectively, using the IQR method. Specifically, any value that is less than the first quartile minus 1.5 times the IQR or greater than the third quartile plus 1.5 times the IQR is considered an outlier.

The code then **initializes an empty list array\_outliers** and **an empty list pos\_outliers** to keep track of the positions of outliers. The **variable n\_outliers is initialized to zero** to keep **count** of the total number of outliers found.

Subsequently it is use a **for loop** to iterates over each row (line) of the data X. For each row, the loop checks each value against the upper and lower bounds calculated earlier. If a value is found to be an outlier, the **flag variable** is set to True and the counter n\_outliers is incremented. The position of the outlier is then added to the pos\_outliers list.

Finally, the **flag variable is** **added to the array\_outliers** list to indicate whether or not each value is an outlier for the current row. The pos\_outliers list is reset to an empty list for the next row.

The resulting array\_outliers list contains True values for positions where outliers were found and False values otherwise, indicating which values are outliers in the original dataset.

# =============================================================================

# there are outliers

# =============================================================================

import numpy as np

q1 = np.quantile(X, 0.25, axis = 0)

q3 = np.quantile(X, 0.75, axis = 0)

med = np.median(X)

iqr = q3 - q1

upper\_bound = q3+(1.5\*iqr)

lower\_bound = q1-(1.5\*iqr)

#Array con un array con la posicion del outlier a true cuando ocurra el oulier o a false cuando no sea outlier

array\_outliers = []

pos\_outliers = []

n\_outliers = 0

for line in X:

count = 0

flag = False

for count, value in enumerate(line):

if value < lower\_bound[count] or value > upper\_bound[count]:

flag = True

n\_outliers +=1

else:

flag = False

pos\_outliers.append(flag)

array\_outliers.append(pos\_outliers)

pos\_outliers = []

# =============================================================================

# true are the outlier and false otherwise

# =============================================================================

print (array\_outliers)

1. **Repeat each of the experiments seen in class with the K-Nearest Neighbors model, providing arguments for each of the steps taken along the process, and commenting on the partial results obtained with the selected dataset. Please use as many performance metrics as needed to illustrate the particularities of the selected dataset (e.g. imbalanced classes).**

The dataset is split into training and testing sets, and a k-NN model is trained and tested. The trained model is then used to make predictions on the testing set with the predict method.

The performance of the model is evaluated using the confusion\_matrix and classification\_report functions, which compute metrics such as accuracy, precision, recall, and F1-score.

The confusion\_matrix function returns a matrix that shows the number of true positives, false positives, true negatives, and false negatives for each class.

The classification\_report function provides a summary of performance metrics for each class, as well as a weighted average of the metrics across all classes.

=============================================================================

# Experiments seen in the class

============================================================================

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, digits.target, test\_size=0.1, random\_state=42)

# Train the model

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

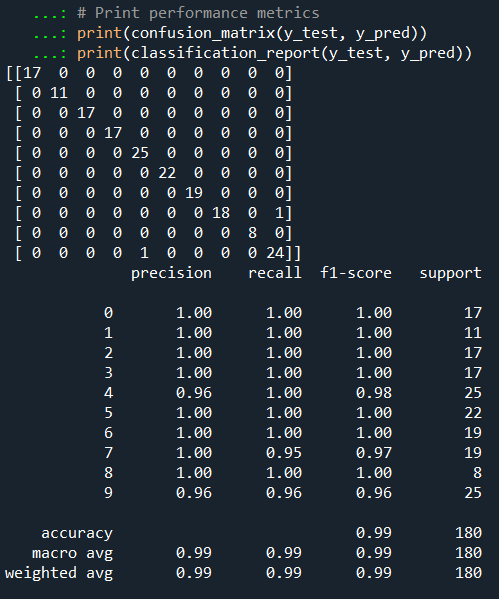
# Test the model

y\_pred = knn.predict(X\_test)

# Print performance metrics

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))



1. **Read the Scikit-learn library documentation and configure the automated validation script so that the GridSearchCV() function uses leave-one-out cross-validation instead of k-fold. Which conclusions can be drawn from the mean cross-validation scores and the test scores using a k-neighbor model with optimized k?**

GridSearchCV is a method in scikit-learn used for hyperparameter tuning of machine learning models. It exhaustively searches over a defined parameter grid for the best combination of hyperparameters that optimize a chosen scoring metric.

In GridSearchCV, you specify the model to be tuned, the hyperparameter space to be explored, and a cross-validation strategy for evaluating the performance of each parameter combination. GridSearchCV then fits the model on every combination of hyperparameters using cross-validation and returns the set of hyperparameters that achieve the best performance on the chosen metric.

In this case we configure the automated validation to do GridSearchCV() function using Leave-One-Out Cross Validation.

After fitting the GridSearchCV object to the training data, the best model is obtained using the "best\_estimator\_" attribute.

The mean cross-validation scores can give an estimate of how well the model is likely to perform on unseen data, based on how well it performed on the training data in the cross-validation procedure. If the mean cross-validation score is high, it suggests that the model is generalizing well to new data and is not overfitting to the training data.

The test scores, on the other hand, provide an estimate of how well the model is performing on a completely independent test set that was not used during the model training or cross-validation.

If the test score is consistent with the mean cross-validation score, it suggests that the model is performing well and is not overfitting to the training data.

# =============================================================================

# 4-GridSearchCV()

# =============================================================================

from sklearn.model\_selection import GridSearchCV, LeaveOneOut

param\_grid = {'n\_neighbors':[1,3]}

miKNNC = KNeighborsClassifier()

miGSCV = GridSearchCV(estimator=miKNNC,

scoring='accuracy',

param\_grid=param\_grid,

cv=LeaveOneOut(),verbose=4)

miGSCV.fit(X\_train, y\_train)

print("el mejor es:",miGSCV.best\_score\_)

miMejorModelo = miGSCV.best\_estimator\_

y\_pred = miMejorModelo.predict(X\_test)

Texto

Descripción generada automáticamente

Finally, it computes the accuracy score of the k-nearest neighbor model by comparing the predicted values (y\_pred) with the true values (y\_test), expressed as a percentage.

Then we make a scatter plot of the predicted values (y\_pred) against the true values (y\_test), with each data point representing an observation in the test set.

Ultimately it show the coefficient of determination (R-squared) between the predicted values (y\_pred) and the true values (y\_test) using the r2\_score function from sklearn.metrics. R-squared is a statistical measure that indicates how well the predicted values fit the actual values and ranges from 0 to 1, with higher values indicating better fit.

# =============================================================================

# Medimos acierto

# =============================================================================

print(100.\*sum(y\_pred==y\_test)/len(y\_test))

plt.scatter(y\_test,y\_pred,s=40)

plt.plot([min(y\_test),max(y\_test)],[min(y\_test),max(y\_test)],'k',lw=3)

from sklearn.metrics import r2\_score

print(r2\_score(y\_test,y\_pred))

Texto

Descripción generada automáticamente

1. **Elaborate on the need for stratifying the cross-validation process analyzing the distribution of samples by class. If so, please show with empirical evidence what could occur if such a stratification was not performed, especially when decreasing the number of samples of the dataset.**

This code block shows how to perform data scaling and data splitting using stratified shuffle split cross-validation in scikit-learn.

First, it imports the StandardScaler class from the scikit-learn preprocessing module. This class is used to standardize the data by centering and scaling it. The next line creates an instance of the StandardScaler class called miScaler.

Then the StratifiedShuffleSplit class is imported from the scikit-learn model\_selection module. This class provides a way to split the data into training and testing sets while preserving the relative frequencies of each class label in the original data.

An instance of the StratifiedShuffleSplit class is created with the n\_splits parameter set to 3, which means that we split on 3. The test\_size parameter is set to 0.1, which means that 10% of the data is reserved for testing. The random\_state parameter is set to 42 to ensure reproducibility of the results.

Afterwards, a for loop is used to iterate over the splits generated by sss.split(). The loop do 3 iteration since n\_splits was set to 3. Inside the loop, the train\_index and test\_index variables are assigned the indices of the training and testing sets, respectively, for the current split.

Finally, the training and testing sets are created by indexing X\_scaled and y using train\_index and test\_index, and assigning the resulting arrays to X\_train, y\_train, X\_test, and y\_test.

from sklearn.preprocessing import StandardScaler

miScaler = StandardScaler()

X\_scaled = miScaler.fit\_transform(X)

from sklearn.model\_selection import StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n\_splits = 3, test\_size=0.1, random\_state=42)

for i, (train\_index, test\_index) in enumerate(sss.split(X, y)):

X\_train = X\_scaled[train\_index,:]

y\_train = y[train\_index]

X\_test = X\_scaled[test\_index,:]

y\_test = y[test\_index]

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_neighbors':[1,3],

'weights':['uniform', 'distance'],

'p':[2,3,4,5]}

miKNNC = KNeighborsClassifier()

miGSCV = GridSearchCV(estimator=miKNNC,

param\_grid=param\_grid,

scoring='accuracy',

cv=LeaveOneOut(),verbose=4) # 5-fold stratified CV

miGSCV.fit(X\_train, y\_train)

print("el mejor es:",miGSCV.best\_score\_)

miMejorModelo = miGSCV.best\_estimator\_

y\_pred = miMejorModelo.predict(X\_test)

Stratification is important when dealing with imbalanced datasets, where the number of samples in each class is not equal. If the cross-validation process is not stratified, the model may not learn to predict the minority class accurately.

When decreasing the number of samples of the dataset, stratification becomes more important. Without stratification, the training and testing datasets may have significantly different distributions of samples by class, leading to overfitting or underfitting of the model.

For example, suppose a dataset has 90% samples from class A and 10% samples from class B. If we randomly split the data into 5 folds for cross-validation without stratification, there is a high probability that one or more folds will have little or no samples from class B. This can lead to an overly optimistic estimate of model performance, as the model will not have been tested on the class that is underrepresented in that fold.

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

# artificially decrease number of samples in class 1

X = X[y != 1]

y = y[y != 1]

y[y == 2] = 1

print('Number of samples in each class:', np.bincount(y))

# KNN model

knn = KNeighborsClassifier()

# cross-validation without stratification

scores = cross\_val\_score(knn, X, y, cv=5)

print('Cross-validation scores without stratification:', scores)

print('Mean score without stratification:', np.mean(scores))

# cross-validation with stratification

skf = StratifiedKFold(n\_splits=5)

scores = cross\_val\_score(knn, X, y, cv=skf)

print('Cross-validation scores with stratification:', scores)

print('Mean score with stratification:', np.mean(scores))

To demostrate the importance of stratifying the cross-validation process, first, the number of samples in class 1 is artificially decreased by removing them from X and y, and the label of class 2 is changed to 1. Then, the number of samples in each class is printed using np.bincount(y).

Next, a KNN model is created, and cross-validation is performed with and without stratification. The cross\_val\_score function from sklearn.model\_selection is used for this purpose. When cross-validation is performed without stratification, the mean score is printed. Then, cross-validation with stratification is performed using the StratifiedKFold function from sklearn.model\_selection, and the mean score is printed.

The results demonstrate that when the number of samples in a class is significantly decreased, cross-validation without stratification can result in a biased estimation of model performance, leading to an overestimation of model accuracy.

**Texto

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

1. **Include in the set of hyper-parameters adjusted via cross-validation process the weights of the distance metric between samples according to the “weights” parameter of the model in Scikit-learn. Compute the model’s performance when distance metric weights are fine-tuned within cross-validation with respect to only tuning the number of neighbors (K).**

The weights parameter in K-Nearest Neighbors (KNN) model controls the contribution of the neighbors to the prediction of the target variable.

Tuning the weights parameter is important because it allows the model to assign different weights to the neighbors based on their distance from the query point.

This can significantly improve the accuracy of the model.

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_neighbors':[1,3],

**'weights':['uniform', 'distance'],**

'p':[2,3,4,5]}

miKNNC = KNeighborsClassifier()

miGSCV = GridSearchCV(estimator=miKNNC,

param\_grid=param\_grid,

scoring='accuracy',

cv=LeaveOneOut(),verbose=4) # 5-fold stratified CV

miGSCV.fit(X\_train, y\_train)

print("el mejor es:",miGSCV.best\_score\_)

miMejorModelo = miGSCV.best\_estimator\_

y\_pred = miMejorModelo.predict(X\_test)

**Texto

Descripción generada automáticamente**

1. **Following the same approach as in the last section, enter the type of distance metric (“metric” parameter) within the cross-validation process. Evaluates the results and gains / losses of generalizability of the model.**

Different distance metrics, such as Euclidean, Manhattan, and Minkowski, have different sensitivities to various feature scales and distributions. Manhattan distance metric may perform better than Euclidean distance in some cases where the features have different units and scales.

By including the distance metric as a hyper-parameter and tuning it within the cross-validation process, we can evaluate the impact of different distance metrics on the model's performance and select the best one.

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_neighbors':[1,3],

'weights':['uniform', 'distance'],

'p':[2,3,4,5],

**'metric': ['manhattan', 'chebyshev']}**

miKNNC = KNeighborsClassifier()

miGSCV = GridSearchCV(estimator=miKNNC,

param\_grid=param\_grid,

scoring='accuracy',

cv=LeaveOneOut(),verbose=4)

miGSCV.fit(X\_train, y\_train)

print("el mejor es:",miGSCV.best\_score\_)

miMejorModelo = miGSCV.best\_estimator\_

y\_pred = miMejorModelo.predict(X\_test)

Imagen que contiene Texto

Descripción generada automáticamente

1) (ADVANCED) Repeat questions 1 to 7 with a dataset of your choice from the [UCI repository,](https://archive.ics.uci.edu/ml/datasets.html) Choose a classification dataset with a “moderate” number of samples not to overload the computing resources)

Reports can be a DOC document, a PDF document (along with the Python scripts that generate the reported figures and results) or a Jupyter Notebook (with saved checkpoint). Other formats (e.g. link to Google Colab) must be agreed with the professor. When uploading the report, please indicate name, surname and ID (DNI number) of all members of the team.

**Delivery deadline:** March 7th, 2021