Full name: Iván López Muñoz **2023/24**
Full name: David Suárez Campione **Group: 04**

11752 Machine Learning
Master in Intelligent Systems
Universitat de les Illes Balears

Handout #2: **Supervised learning**

T0. Normalize the dataset samples using *max-min normalization* and consider the following cases:

a. **case A**. All features.

b. **case B**. Best two features according to PCA.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

# Load the dataset (replace gg with your group number)
gg = 4
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Separating features and target (13th column - popularity class)
# Separate features and target
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# PCA for feature reduction (Case B)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Normalize the dataset using Min-Max scaling for CASE B
scaler = MinMaxScaler()
X_B = scaler.fit_transform(X_pca)

# Normalize the dataset using Min-Max scaling for CASE A
scaler = MinMaxScaler()
X_A = scaler.fit_transform(X)
```

T1. **(only for case B)** Assuming that class data follow a 2D Gaussian distribution, consider the **quadratic Bayesian classifier** case, i.e. different covariance matrices for each class, find and report the discrimination function $g_i(x)$ for each class ω_i, i.e. $g_i(x) = a_i x_1^2 + b_i x_2^2 + c_i x_1 x_2 + d_i x_1 + e_i x_2 + f_i$, and evaluate

its performance. HINT: You need to calculate the mean and a covariance matrix for each class.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from openpyxl import load_workbook
from sympy import symbols

# Load the dataset (replace gg with your group number)
gg = 4
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Extracting features and target
X = df.drop(columns=['class'])  # Features
y = df['class']  # Target variable

# Perform PCA to find the two most significant features
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X)

# Normalize all features using MinMaxScaler
scaler = MinMaxScaler()
X_B = scaler.fit_transform(pca_result)

# Splitting the dataset into train and test sets (70-30 split)
X_train, X_test, y_train, y_test = train_test_split(X_B, y, test_size=0.3,
random_state=100)

# Separate the training set into classes
class_1 = X_train[y_train == 0]
class_2 = X_train[y_train == 1]

# Calculate mean and covariance matrix for each class
mean_class_1 = np.mean(class_1, axis=0)
cov_class_1 = np.cov(class_1.T)

mean_class_2 = np.mean(class_2, axis=0)
cov_class_2 = np.cov(class_2.T)

# Initialize and fit the Quadratic Discriminant Analysis model
qda = QuadraticDiscriminantAnalysis()
```

```
qda.fit(X_train, y_train)

# Predict on the test set
y_pred = qda.predict(X_test)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Fill in the results in the Excel file
wb = load_workbook(filename="results.xlsx")
ws = wb.active
ws["J3"] = accuracy
ws["L3"] = precision
ws["N3"] = recall
ws["P3"] = f1
wb.save("results.xlsx")

# Calculate coefficients for quadratic equations representing g_i(x)

# Coefficients for g_1(x)
a_1 = cov_class_1[0, 0]
b_1 = cov_class_1[1, 1]
c_1 = cov_class_1[0, 1] + cov_class_1[1, 0]
d_1 = -2 * cov_class_1[0, 0] * mean_class_1[0] - cov_class_1[0, 1] - cov_class_1[1, 0] *
mean_class_1[1]
e_1 = -2 * cov_class_1[1, 1] * mean_class_1[1] - cov_class_1[0, 1] - cov_class_1[1, 0] *
mean_class_1[0]
f_1 = mean_class_1 @ cov_class_1 @ mean_class_1.T - 2 * mean_class_1 @ cov_class_1 +
np.log(np.linalg.det(cov_class_1))

# Coefficients for g_2(x)
a_2 = cov_class_2[0, 0]
b_2 = cov_class_2[1, 1]
c_2 = cov_class_2[0, 1] + cov_class_2[1, 0]
d_2 = -2 * cov_class_2[0, 0] * mean_class_2[0] - cov_class_2[0, 1] - cov_class_2[1, 0] *
mean_class_2[1]
e_2 = -2 * cov_class_2[1, 1] * mean_class_2[1] - cov_class_2[0, 1] - cov_class_2[1, 0] *
mean_class_2[0]
f_2 = mean_class_2 @ cov_class_2 @ mean_class_2.T - 2 * mean_class_2 @ cov_class_2 +
np.log(np.linalg.det(cov_class_2))

# Define symbols for x1 and x2
x1, x2 = symbols('x1 x2')
```

```
# Define the quadratic equations for g_1(x), g_2(x), and g_12(x) in terms of x1 and x2
def quadratic_eq_in_x(a, b, c, d, e, f):
   return a * x1**2 + b * x2**2 + c * x1 * x2 + d * x1 + e * x2 + f

# Print the equations
print(f"g_1(x): {quadratic_eq_in_x(a_1, b_1, c_1, d_1, e_1, f_1)}")
print(f"g_2(x): {quadratic_eq_in_x(a_2, b_2, c_2, d_2, e_2, f_2)}")
print(f"g_12(x): {quadratic_eq_in_x(a_1 - a_2, b_1 - b_2, c_1 - c_2, d_1 - d_2, e_1 - e_2, f_1 - f_2)}")

'''
Task 1 involved the application of Quadratic Discriminant Analysis (QDA) to a dataset, yielding a model with particular performance metrics. The average accuracy obtained was 0.650, indicating the proportion of correct predictions made by the model.
Precision, measuring the accuracy of positive predictions, was 0.835 on average, while the average recall, reflecting the model's ability to capture positive instances, stood at 0.708. Additionally, the F1 score, harmonizing precision and recall, averaged at 0.614.

The model's decision boundaries were represented by quadratic equations (g_1(x), g_2(x), and g_12(x)), portraying the separation between classes 1 and 2.
Coefficients for these equations were calculated using mean and covariance matrices for each class, providing insights into how the model distinguishes between different classes based on the dataset's features.

This evaluation showcases the QDA model's performance in accurately predicting and distinguishing between classes, demonstrating reasonable accuracy, precision, and recall. The derived quadratic equations offer a visual understanding of the decision boundaries established by the model.
'''
```

$g_1(x) = 0.027x_1^2+0.034x_2^2-0.020x_1-0.025x_2-0.001x_1x_2-6.994$

$g_2(x) = 0.007x_1^2+0.027x_2^2-0.004x_1-0.023x_2+0.001x_1x_2-8.632$

$g_{12}(x) = 0.021x_1^2+0.007x_2^2-0.0015x_1-0.001x_2-0.002x_1x_2+1.639$

T2. **(only for case B)** Assuming that class data follow a 2D Gaussian distribution, consider the **linear Bayesian classifier** case, i.e. same covariance matrix for each class, find and report the discrimination function $g_i(x)$ for each class ω_i, i.e. $g_i(x) = a_i x_1 + b_i x_2 + c_i$, and evaluate its performance. HINT: You need to calculate the mean for each class and a single covariance matrix for the full dataset, shared by all classes.

```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from openpyxl import load_workbook

# Load the dataset (replace gg with your group number)
gg = 4
df = pd.read_csv('ds%02d_alt.csv' % gg)

# Extracting features and target
X = df.drop(columns=['class'])  # Features
y = df['class']  # Target variable

# Perform PCA to find the two most significant features
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Normalize 2 features using MinMaxScaler
scaler = MinMaxScaler()
X_B = scaler.fit_transform(X_pca)

# Split the dataset into train and test sets (70-30 split)
X_train, X_test, y_train, y_test = train_test_split(X_B, y, test_size=0.3,
random_state=100)

# Initialize and fit the Linear Discriminant Analysis model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Separate the training set into classes
X_train_class_1 = X_train[y_train == 0]
X_train_class_2 = X_train[y_train == 1]

# Predict on the test set
y_pred = lda.predict(X_test)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
# Fill in the results in the Excel file
wb = load_workbook(filename="results.xlsx")
ws = wb.active
ws["J4"] = accuracy
ws["L4"] = precision
ws["N4"] = recall
ws["P4"] = f1
wb.save("results.xlsx")

# Calculate mean for each class
mean_class_1 = np.mean(X_train_class_1, axis=0)
mean_class_2 = np.mean(X_train_class_2, axis=0)

# Calculate covariance matrix (same for both classes)
cov_matrix = np.cov(X_train.T)

# Calculate inverse of the covariance matrix
cov_inv = np.linalg.inv(cov_matrix)

# Calculate coefficients for class 1
a_1 = -2 * cov_matrix[0, 0] * (mean_class_1[0] - mean_class_2[0])
b_1 = -2 * cov_matrix[0, 1] * (mean_class_1[1] - mean_class_2[1])
c_1 = mean_class_1 @ np.linalg.inv(cov_matrix) @ mean_class_1.T - mean_class_2 @
np.linalg.inv(cov_matrix) @ mean_class_2.T

# Coefficients for class 2 (omega_2)
a_2 = -2 * cov_matrix[0, 0] * (mean_class_2[0] - mean_class_1[0])
b_2 = -2 * cov_matrix[0, 1] * (mean_class_2[1] - mean_class_1[1])
c_2 = mean_class_2 @ np.linalg.inv(cov_matrix) @ mean_class_2.T - mean_class_1 @
np.linalg.inv(cov_matrix) @ mean_class_1.T

# Calculate g_12(x) by subtracting coefficients
a_12 = a_1 - a_2
b_12 = b_1 - b_2
c_12 = c_1 - c_2

# Get coefficients for the decision boundary
coefficients = lda.coef_[0]
# Get the intercept
intercept = lda.intercept_

# Print the functions
print(f"g_1(x) = {a_1} * x1 + {b_1} * x2 + {c_1}")
print(f"g_2(x) = {a_2} * x1 + {b_2} * x2 + {c_2}")
print(f"g_1_2(x) = {a_12} * x1 + {b_12} * x2 + {c_12}")
```

```
# Get coefficients for the decision boundary
coefficients = -lda.coef_[0]
# Get the intercept
intercept = -lda.intercept_

# Plotting the decision boundary
x_min, x_max = X_train[:, 0].min(), X_train[:, 0].max()
y_min, y_max = X_train[:, 1].min(), X_train[:, 1].max()
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

Z = lda.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k')
plt.title('Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()


'''
Task 2 introduced Linear Discriminant Analysis (LDA) applied to a dataset,
revealing a model with moderate but noteworthy performance.
The model achieved an average accuracy of 0.665, signifying a reasonably high rate of
correct predictions.
Precision and recall, averaging at 0.652 and 0.728, respectively, along with an F1-
score of 0.688,
highlighted the model's capability in correctly identifying positive instances while
capturing a significant portion of actual positives.

The quadratic equations (g_1(x), g_2(x), and g_1_2(x)) derived from the model's
coefficients
elucidate the decision boundaries between distinct classes.
While the model demonstrated competent discriminative ability between classes,
these metrics suggest that there's potential for further enhancement, especially
in optimizing precision and recall to handle more complex datasets.

The LDA model's performance appears satisfactory,
especially in its ability to classify data and differentiate between classes.
However, fine-tuning the model to strike a better balance between precision
and recall might be beneficial for handling more intricate and nuanced datasets,
ensuring robust performance in varied scenarios.
'''
```

$g_1(x) = -0.003x_1 - 0.0001x_2 + 1.838$

$g_2(x) = 0.003x_1+0.0001x_2-1.838$

$g_{12}(x) = -0.006x_1-0.0002x_2+3.676$

T3. **(cases A and B)** Implement a **generic Bayesian classifier** (BC) estimating the probability $p(x|\omega_i)$ using the *probability density estimator* (PDE) available in *scikit-learn* for a Gaussian kernel, and evaluate its performance. HINT: The *scikit-learn* implementation of a PDE is available as object *KernelDensity*. For the *bandwidth h* use the recommendation of slide 47 with $h_1 = 1$. Notice that the method *score_samples* of the density estimator provides you with the log-likelihood $\log_e p(x|\omega_i)$ instead of directly $p(x|\omega_i)$.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import MinMaxScaler
from openpyxl import load_workbook
import numpy as np
from sklearn.decomposition import PCA

# Load the dataset
gg = 4  # Replace this with your group number
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Separate features and target
features = df.iloc[:, :-1]
target = df.iloc[:, -1]

'''
CASE A: ALL FEATURES
'''
# Normalize the dataset using Min-Max scaling for both cases
scaler = MinMaxScaler()
X_A = scaler.fit_transform(features)

# Splitting the dataset into train and test sets for Case A (all features)
X_train_A, X_test_A, y_train_A, y_test_A = train_test_split(X_A, target, test_size=0.3, random_state=1)

# Case A: Using all features
# Use Kernel Density Estimator with Gaussian kernel
h1 = 1
N = X_train_A.shape[0]
h = h1 / np.sqrt(N)
```

```
# Fit Kernel Density Estimator
kde_A = KernelDensity(bandwidth=h, kernel='gaussian')
kde_A.fit(X_train_A)

# Calculate log-likelihood for each sample in the test set for each class
log_likelihoods_A = kde_A.score_samples(X_test_A)

# Predict using log-likelihoods (assuming binary classification)
y_pred_A = (log_likelihoods_A >= 0).astype(int)

# Calculate metrics for Case A
accuracy_A = accuracy_score(y_test_A, y_pred_A)
precision_A = precision_score(y_test_A, y_pred_A)
recall_A = recall_score(y_test_A, y_pred_A)
f1_A = f1_score(y_test_A, y_pred_A)

'''
CASE B: BEST 2 FEATURES ACCORDING TO PCA
'''

# Case B: Using PCA for feature reduction and minmax scaler
pca = PCA(n_components=2)
pca_features = pca.fit_transform(features)
scaler = MinMaxScaler()
X_B = scaler.fit_transform(pca_features)

# Splitting the PCA-transformed features into train and test sets for Case B
X_train_B, X_test_B, y_train_B, y_test_B = train_test_split(X_B, target, test_size=0.3,
random_state=1)

# Estimate bandwidth for Gaussian kernel for PCA-transformed features
N_pca = X_train_B.shape[0]
h_pca = h1 / np.sqrt(N_pca)

# Fit Kernel Density Estimator
kde_B = KernelDensity(bandwidth=h_pca, kernel='gaussian')
kde_B.fit(X_train_B)

# Calculate log-likelihood for each sample in the test set for each class
log_likelihoods_B = kde_B.score_samples(X_test_B)

# Predict using log-likelihoods for Case B (assuming binary classification)
y_pred_B = (log_likelihoods_B >= 0).astype(int)

# Calculate metrics for Case B
```

```
accuracy_B = accuracy_score(y_test_B, y_pred_B)
precision_B = precision_score(y_test_B, y_pred_B)
recall_B = recall_score(y_test_B, y_pred_B)
f1_B = f1_score(y_test_B, y_pred_B)

# Fill in the results in the Excel file
wb = load_workbook(filename="results.xlsx")
ws = wb.active
ws["B5"] = accuracy_A
ws["D5"] = precision_A
ws["F5"] = recall_A
ws["H5"] = f1_A

ws["J5"] = accuracy_B
ws["L5"] = precision_B
ws["N5"] = recall_B
ws["P5"] = f1_B

wb.save("results.xlsx")

'''
In Task 3, Case A utilizing all features achieved impressive results across various metrics. It obtained an accuracy of 0.887, indicating strong predictive ability, coupled with a precision of 0.861
and a recall of 0.946, demonstrating the model's reliability in identifying positive instances. The F1-score harmonized these metrics at 0.901, signifying a robust overall performance.

However, Case B, using only the top two features identified by PCA, showed a decrease in performance. Despite maintaining a reasonable accuracy of 0.596, precision, recall, and the F1-score dropped notably to 0.579, 0.955, and 0.721, respectively.
This suggests a trade-off between feature reduction simplicity and predictive power.

The disparity in performance indicates that relying solely on PCA-derived features might sacrifice overall model accuracy and the balance between precision and recall.
It emphasizes the importance of thoughtful feature selection to maintain a well-balanced predictive model.
'''
```

T4. **(cases A and B)** Make use of the implementation of the **naive Bayes** classifier (NB) available in *scikit-learn* and evaluate the resulting classifier. HINT: The *scikit-learn* implements several variations of the NB classifier, use the implementation for Gaussian classes that is available as object *GaussianNB*.

```
import pandas as pd
```

```
from sklearn.model_selection import train_test_split, cross_val_score, RepeatedStratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np
from sklearn.model_selection import cross_val_predict
from openpyxl import load_workbook

# Load the dataset
gg = 4  # Replace with your group number
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Separate features and target
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# PCA for feature reduction (Case B)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Normalize the dataset using Min-Max scaling for CASE B
scaler = MinMaxScaler()
X_B = scaler.fit_transform(X_pca)

# Normalize the dataset using Min-Max scaling for CASE A
scaler = MinMaxScaler()
X_A = scaler.fit_transform(X)

# Specify repetitions and folds for cross-validation
repetitions = 3
folds = 5

# Initialize lists to store metrics across repetitions
accuracy_A_list, precision_A_list, recall_A_list, f1_A_list = [], [], [], []
accuracy_B_list, precision_B_list, recall_B_list, f1_B_list = [], [], [], []

# Perform cross-validation for both cases (A and B)
for i in range(repetitions):
    # Case A: Using all features
    nb_classifier_A = GaussianNB()
    cv_A = RepeatedStratifiedKFold(n_splits=folds, n_repeats=1, random_state=i)
    scores_A = cross_val_score(nb_classifier_A, X_A, y, scoring='accuracy', cv=cv_A)
    accuracy_A_list.extend(scores_A)
```

```
    # Get other metrics in addition to accuracy for Case A
    predictions_A = cross_val_predict(nb_classifier_A, X_A, y, cv=cv_A)
    precision_A_list.extend(precision_score(y, predictions_A, average=None))
    recall_A_list.extend(recall_score(y, predictions_A, average=None))
    f1_A_list.extend(f1_score(y, predictions_A, average=None))

    # Case B: Using PCA for feature reduction
    nb_classifier_B = GaussianNB()
    cv_B = RepeatedStratifiedKFold(n_splits=folds, n_repeats=1, random_state=i)
    scores_B = cross_val_score(nb_classifier_B, X_B, y, scoring='accuracy', cv=cv_B)
    accuracy_B_list.extend(scores_B)

    # Get other metrics in addition to accuracy for Case B
    predictions_B = cross_val_predict(nb_classifier_B, X_B, y, cv=cv_B)
    precision_B_list.extend(precision_score(y, predictions_B, average=None))
    recall_B_list.extend(recall_score(y, predictions_B, average=None))
    f1_B_list.extend(f1_score(y, predictions_B, average=None))

# Calculate average and standard metrics for both cases
accuracy_A_avg = np.mean(accuracy_A_list)
accuracy_A_std = np.std(accuracy_A_list)
precision_A_avg = np.mean(precision_A_list)
precision_A_std = np.std(precision_A_list)
recall_A_avg = np.mean(recall_A_list)
recall_A_std = np.std(recall_A_list)
f1_A_avg = np.mean(f1_A_list)
f1_A_std = np.std(f1_A_list)

accuracy_B_avg = np.mean(accuracy_B_list)
accuracy_B_std = np.std(accuracy_B_list)
precision_B_avg = np.mean(precision_B_list)
precision_B_std = np.std(precision_B_list)
recall_B_avg = np.mean(recall_B_list)
recall_B_std = np.std(recall_B_list)
f1_B_avg = np.mean(f1_B_list)
f1_B_std = np.std(f1_B_list)

# Load the existing Excel file
wb = load_workbook(filename="results.xlsx")
ws = wb.active

# Define the cell positions for writing the results
cell_positions = {
    "B": accuracy_A_avg,
    "C": accuracy_A_std,
    "D": precision_A_avg,
```

```
    "E": precision_A_std,
    "F": recall_A_avg,
    "G": recall_A_std,
    "H": f1_A_avg,
    "I": f1_A_std,
    "J": accuracy_B_avg,
    "K": accuracy_B_std,
    "L": precision_B_avg,
    "M": precision_B_std,
    "N": recall_B_avg,
    "O": recall_B_std,
    "P": f1_B_avg,
    "Q": f1_B_std,
}

# Write the results to the specified cells
for col, value in cell_positions.items():
    ws[col + "6"] = value

# Save the updated Excel file
wb.save("results.xlsx")

'''
Case A demonstrated robustness in predicting target classes,
with an average accuracy of 0.770 and a standard deviation of 0.029,
suggesting consistent and reliable predictions across repetitions.
Precision averaged at 0.806 with a standard deviation of 0.103, implying a
high proportion of correctly identified positive instances.
The recall score at 0.768 with a standard deviation of 0.175 showcased the model's
ability to capture most positive instances, while the F1 score,
harmonizing precision and recall, averaged 0.762 with a standard deviation of 0.044,
indicating a balanced performance in classification.

In contrast, Case B utilizing only the top two PCA-derived features displayed
a slightly lower average accuracy of 0.646 with a standard deviation of 0.029.
This suggests a reduction in overall predictive power compared to Case A.
The precision for Case B averaged at 0.672 with a standard deviation of 0.064,
indicating a moderate proportion of correctly identified positive instances.
The recall score, averaging at 0.644 with a standard deviation of 0.201,
showcased the model's ability to capture positive instances, albeit less effectively.
The F1 score, at 0.630 with a standard deviation of 0.077, indicated a trade-off
between precision and recall.

This evaluation highlights the trade-off between model complexity and performance,
showcasing that utilizing all features (Case A) resulted in a more reliable and balanced
model in terms of classification accuracy, precision, recall, and F1 score compared to
```

```
reducing features through PCA (Case B). It emphasizes the importance of feature selection
strategies and their direct impact on model performance and predictive power.
'''
```

T5. **(cases A and B)** Make use of the implementation of **logistic regression** (LR) available in *scikit-learn* and evaluate the resulting classifier. HINT: The *scikit-learn* implementation of LR is available as object *LogisticRegression*. Do not incorporate any regularization term (`penalty = None`).

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, RepeatedStratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np
from sklearn.model_selection import cross_val_predict
from openpyxl import load_workbook

# Load the dataset
gg = 4  # Replace with your group number
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Separate features and target
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# PCA for feature reduction (Case B)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Normalize the dataset using Min-Max scaling for CASE B
scaler = MinMaxScaler()
X_B = scaler.fit_transform(X_pca)

# Normalize the dataset using Min-Max scaling for CASE A
scaler = MinMaxScaler()
X_A = scaler.fit_transform(X)

# Specify repetitions and folds for cross-validation
repetitions = 3
folds = 5
```

```
# Initialize lists to store metrics across repetitions
accuracy_A_list, precision_A_list, recall_A_list, f1_A_list = [], [], [], []
accuracy_B_list, precision_B_list, recall_B_list, f1_B_list = [], [], [], []

# Perform cross-validation for both cases (A and B)
for i in range(repetitions):
    # Case A: Using all features
    lr_classifier_A = LogisticRegression(penalty=None)
    cv_A = RepeatedStratifiedKFold(n_splits=folds, n_repeats=1, random_state=i)
    scores_A = cross_val_score(lr_classifier_A, X_A, y, scoring='accuracy', cv=cv_A)
    accuracy_A_list.extend(scores_A)

    # Get other metrics in addition to accuracy for Case A
    predictions_A = cross_val_predict(lr_classifier_A, X_A, y, cv=cv_A)
    precision_A_list.extend(precision_score(y, predictions_A, average=None))
    recall_A_list.extend(recall_score(y, predictions_A, average=None))
    f1_A_list.extend(f1_score(y, predictions_A, average=None))

    # Case B: Using PCA for feature reduction
    lr_classifier_B = LogisticRegression(penalty=None)
    cv_B = RepeatedStratifiedKFold(n_splits=folds, n_repeats=1, random_state=i)
    scores_B = cross_val_score(lr_classifier_B, X_B, y, scoring='accuracy', cv=cv_B)
    accuracy_B_list.extend(scores_B)

    # Get other metrics in addition to accuracy for Case B
    predictions_B = cross_val_predict(lr_classifier_B, X_B, y, cv=cv_B)
    precision_B_list.extend(precision_score(y, predictions_B, average=None))
    recall_B_list.extend(recall_score(y, predictions_B, average=None))
    f1_B_list.extend(f1_score(y, predictions_B, average=None))

# Calculate average and standard metrics for both cases
accuracy_A_avg = np.mean(accuracy_A_list)
accuracy_A_std = np.std(accuracy_A_list)
precision_A_avg = np.mean(precision_A_list)
precision_A_std = np.std(precision_A_list)
recall_A_avg = np.mean(recall_A_list)
recall_A_std = np.std(recall_A_list)
f1_A_avg = np.mean(f1_A_list)
f1_A_std = np.std(f1_A_list)

accuracy_B_avg = np.mean(accuracy_B_list)
accuracy_B_std = np.std(accuracy_B_list)
precision_B_avg = np.mean(precision_B_list)
precision_B_std = np.std(precision_B_list)
recall_B_avg = np.mean(recall_B_list)
recall_B_std = np.std(recall_B_list)
```

```
f1_B_avg = np.mean(f1_B_list)
f1_B_std = np.std(f1_B_list)

# Load the existing Excel file
wb = load_workbook(filename="results.xlsx")
ws = wb.active

# Define the cell positions for writing the results
cell_positions = {
   "B": accuracy_A_avg,
   "C": accuracy_A_std,
   "D": precision_A_avg,
   "E": precision_A_std,
   "F": recall_A_avg,
   "G": recall_A_std,
   "H": f1_A_avg,
   "I": f1_A_std,
   "J": accuracy_B_avg,
   "K": accuracy_B_std,
   "L": precision_B_avg,
   "M": precision_B_std,
   "N": recall_B_avg,
   "O": recall_B_std,
   "P": f1_B_avg,
   "Q": f1_B_std,
}

# Write the results to the specified cells
for col, value in cell_positions.items():
   ws[col + "7"] = value

# Save the updated Excel file
wb.save("results.xlsx")

'''
```

Case A displayed an improvement in all metrics compared to Task 4. With all features utilized, Task 5's Case A achieved higher values in accuracy (0.815 vs. 0.770), precision (0.817 vs. 0.806), recall (0.814 vs. 0.768), and F1 score (0.814 vs. 0.762). This upturn showcases the efficacy of logistic regression with the full feature set in this task, potentially owing to optimized model tuning or dataset specifics.

Conversely, Task 5's Case B exhibited similar metrics as Task 4's Case B.
Both cases utilized PCA for feature reduction and logistic regression for classification.
Despite similar methodologies, Task 5's Case B showed only slight fluctuations in accuracy (0.642 vs. 0.646), precision (0.643 vs. 0.672), recall (0.641 vs. 0.644), and F1

score (0.641 vs. 0.630) compared to Task 4. This outcome suggests consistency in the predictive capability of a reduced-feature logistic regression model between the tasks.

The variance observed across Task 5's cases echoes the importance of exploring different feature selection methods and algorithms. While Case A's improvement in Task 5 signifies potential enhancements in the model or data optimization, the similarity in Case B between tasks underlines the consistency of feature reduction and logistic regression within this specific scope.
These findings emphasize the iterative nature of model building, where diverse approaches impact model performance, and incremental enhancements can refine predictive capabilities
'''

T6. **(cases A and B)** Adopt a classifier ensemble approach using the **stacking** approach, comprising NB, LR and decision tree (DT) classifiers. HINT: The *scikit-learn* implementation of the DT classifier is available as object *DecisionTreeClassifier*. Use also a DT for the second layer. For the first layer, regularize the DT imposing a minimum of 10 samples per leaf, while, for the second layer (the blender), impose a maximum depth of 3.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, RepeatedStratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import StackingClassifier
import numpy as np
from openpyxl import load_workbook
from sklearn.model_selection import cross_val_predict

# Load the dataset
gg = 4  # Replace with your group number
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Separate features and target
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target

# PCA for feature reduction (Case B)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

```
# Normalize the dataset using Min-Max scaling for CASE B
scaler = MinMaxScaler()
X_B = scaler.fit_transform(X_pca)

# Normalize the dataset using Min-Max scaling for CASE A
scaler = MinMaxScaler()
X_A = scaler.fit_transform(X)

# Define classifiers for the first layer
nb_classifier = GaussianNB()
lr_classifier = LogisticRegression()
dt_classifier = DecisionTreeClassifier(min_samples_leaf=10)

# Define blender (meta-classifier)
blender = DecisionTreeClassifier(max_depth=3)

# Stacking ensemble model
estimators = [('nb', nb_classifier), ('lr', lr_classifier), ('dt', dt_classifier)]
stacking_classifier = StackingClassifier(estimators=estimators,
final_estimator=blender)

# Specify repetitions and folds for cross-validation
repetitions = 3
folds = 5

# Initialize lists to store metrics across repetitions
accuracy_A_list, precision_A_list, recall_A_list, f1_A_list = [], [], [], []
accuracy_B_list, precision_B_list, recall_B_list, f1_B_list = [], [], [], []

# Perform cross-validation for both cases (A and B)
for i in range(repetitions):
    # Case A: Using all features
    cv_A = RepeatedStratifiedKFold(n_splits=folds, n_repeats=1, random_state=i)
    scores_A = cross_val_score(stacking_classifier, X_A, y, scoring='accuracy', cv=cv_A)
    accuracy_A_list.extend(scores_A)

    # Get other metrics in addition to accuracy for Case A
    predictions_A = cross_val_predict(stacking_classifier, X_A, y, cv=cv_A)
    precision_A_list.extend(precision_score(y, predictions_A, average=None))
    recall_A_list.extend(recall_score(y, predictions_A, average=None))
    f1_A_list.extend(f1_score(y, predictions_A, average=None))

    # Case B: Using PCA for feature reduction
    cv_B = RepeatedStratifiedKFold(n_splits=folds, n_repeats=1, random_state=i)
    scores_B = cross_val_score(stacking_classifier, X_B, y, scoring='accuracy', cv=cv_B)
```

```
    accuracy_B_list.extend(scores_B)

    # Get other metrics in addition to accuracy for Case B
    predictions_B = cross_val_predict(stacking_classifier, X_B, y, cv=cv_B)
    precision_B_list.extend(precision_score(y, predictions_B, average=None))
    recall_B_list.extend(recall_score(y, predictions_B, average=None))
    f1_B_list.extend(f1_score(y, predictions_B, average=None))

# Calculate average and standard metrics for both cases
accuracy_A_avg = np.mean(accuracy_A_list)
accuracy_A_std = np.std(accuracy_A_list)
precision_A_avg = np.mean(precision_A_list)
precision_A_std = np.std(precision_A_list)
recall_A_avg = np.mean(recall_A_list)
recall_A_std = np.std(recall_A_list)
f1_A_avg = np.mean(f1_A_list)
f1_A_std = np.std(f1_A_list)

accuracy_B_avg = np.mean(accuracy_B_list)
accuracy_B_std = np.std(accuracy_B_list)
precision_B_avg = np.mean(precision_B_list)
precision_B_std = np.std(precision_B_list)
recall_B_avg = np.mean(recall_B_list)
recall_B_std = np.std(recall_B_list)
f1_B_avg = np.mean(f1_B_list)
f1_B_std = np.std(f1_B_list)

# Load the existing Excel file
wb = load_workbook(filename="results.xlsx")
ws = wb.active

# Define the cell positions for writing the results
cell_positions = {
    "B": accuracy_A_avg,
    "C": accuracy_A_std,
    "D": precision_A_avg,
    "E": precision_A_std,
    "F": recall_A_avg,
    "G": recall_A_std,
    "H": f1_A_avg,
    "I": f1_A_std,
    "J": accuracy_B_avg,
    "K": accuracy_B_std,
    "L": precision_B_avg,
    "M": precision_B_std,
    "N": recall_B_avg,
```

```
    "O": recall_B_std,
    "P": f1_B_avg,
    "Q": f1_B_std,
}

# Write the results to the specified cells
for col, value in cell_positions.items():
    ws[col + "8"] = value

# Save the updated Excel file
wb.save("results.xlsx")


'''
```

In Task 6, the stacked ensemble model incorporating Naive Bayes, Logistic Regression, and Decision Tree classifiers performed consistently well across both cases A and B. This ensemble strategy outperformed individual classifiers in terms of accuracy, precision, recall, and F1 score.

Case A, leveraging all features, showcased substantial predictive capability with an accuracy of 0.839 and an F1 score of 0.840. This demonstrates an improvement compared to Task 4 and Task 5 for Case A, reflecting the efficacy of the ensemble method in harnessing the full feature set. The precision and recall metrics, averaging 0.842 and 0.837, respectively, highlight its balanced performance in correctly identifying positive instances while minimizing false positives.

Conversely, Case B's use of reduced features via PCA led to a decline in all metrics compared to Task 5's Case A, indicating that the ensemble's predictive capacity diminished with the reduced feature set. Despite this reduction, Case B in Task 6 still outperformed Task 4 in terms of precision, recall, and F1 score, showcasing the inherent strength of the ensemble approach even with fewer features.

This demonstrates the consistency of the stacked ensemble model across diverse datasets and its ability to adapt to different feature subsets. The relationship across tasks showcases that while Case A continues to perform well across tasks, Case B's performance fluctuates based on the dimensionality reduction method employed, reiterating the importance of feature selection techniques in ensemble learning

```
'''
```

T7. **(cases A and B)**

a. Adopt a classifier ensemble approach using the implementation of **random forests** (RF) available in *scikit-learn*. HINT: The *scikit-learn* implementation of RF is available as object *RandomForestClassifier*.

b. Tune the RF model by means of *grid search* (with `cv = 3`) as follows: (1) number of trees among 20, 40 and 60, (2) minimum samples per leaf equal to 5 or 10, and (3) consider the two impurity criteria *gini* and *entropy*. HINT: The *scikit-learn* implementation of grid search is available as object *GridSearchCV*.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score, RepeatedKFold, cross_val_predict
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from openpyxl import load_workbook

# Load the dataset
gg = 4  # Replace with your group number
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Select features and target variable
features = df.iloc[:, :-1]
target = df.iloc[:, -1]

# Normalize the dataset using Min-Max scaling for both cases
scaler = MinMaxScaler()
X_A = scaler.fit_transform(features)

# Case B: Best two features according to PCA
pca = PCA(n_components=2)
features_B = pca.fit_transform(features)
scaler_B = MinMaxScaler()
X_B = scaler_B.fit_transform(features_B)

# Parameters for grid search
param_grid = {
   'n_estimators': [20, 40, 60],
   'min_samples_leaf': [5, 10],
   'criterion': ['gini', 'entropy']
}

# Number of repetitions and folds
num_repetitions = 3
num_folds = 5
```

```
# Initialize lists to store metric values
accuracy_values_A = []
precision_values_A = []
recall_values_A = []
f1_values_A = []

accuracy_values_B = []
precision_values_B = []
recall_values_B = []
f1_values_B = []

# Loop for repetitions
for i in range(num_repetitions):
    # Case A: All features
    rf_classifier_A = RandomForestClassifier()
    kf_A = RepeatedKFold(n_splits=num_folds, n_repeats=1)

    # Perform cross-validation for Case A
    grid_search_A = GridSearchCV(estimator=rf_classifier_A, param_grid=param_grid, cv=kf_A)
    grid_search_A.fit(X_A, target)
    best_rf_A = grid_search_A.best_estimator_

    accuracy_values_A.extend(cross_val_score(best_rf_A, X_A, target, cv=kf_A, scoring='accuracy'))
    predictions_A = cross_val_predict(best_rf_A, X_A, target, cv=kf_A)
    precision_values_A.extend(precision_score(target, predictions_A, average=None))
    recall_values_A.extend(recall_score(target, predictions_A, average=None))
    f1_values_A.extend(f1_score(target, predictions_A, average=None))

    # Case B: Best two features according to PCA
    rf_classifier_B = RandomForestClassifier()
    kf_B = RepeatedKFold(n_splits=num_folds, n_repeats=1)

    # Perform cross-validation for Case B
    grid_search_B = GridSearchCV(estimator=rf_classifier_B, param_grid=param_grid, cv=kf_B)
    grid_search_B.fit(X_B, target)
    best_rf_B = grid_search_B.best_estimator_

    accuracy_values_B.extend(cross_val_score(best_rf_B, X_B, target, cv=kf_B, scoring='accuracy'))
    predictions_B = cross_val_predict(best_rf_B, X_B, target, cv=kf_B)
    precision_values_B.extend(precision_score(target, predictions_B, average=None))
    recall_values_B.extend(recall_score(target, predictions_B, average=None))
    f1_values_B.extend(f1_score(target, predictions_B, average=None))
```

```
# Calculate mean and standard deviation for each metric
mean_accuracy_A = np.mean(accuracy_values_A)
std_accuracy_A = np.std(accuracy_values_A)
mean_precision_A = np.mean(precision_values_A)
std_precision_A = np.std(precision_values_A)
mean_recall_A = np.mean(recall_values_A)
std_recall_A = np.std(recall_values_A)
mean_f1_A = np.mean(f1_values_A)
std_f1_A = np.std(f1_values_A)

mean_accuracy_B = np.mean(accuracy_values_B)
std_accuracy_B = np.std(accuracy_values_B)
mean_precision_B = np.mean(precision_values_B)
std_precision_B = np.std(precision_values_B)
mean_recall_B = np.mean(recall_values_B)
std_recall_B = np.std(recall_values_B)
mean_f1_B = np.mean(f1_values_B)
std_f1_B = np.std(f1_values_B)

# Update the Excel file with mean and standard deviation values
wb = load_workbook(filename="results.xlsx")
ws = wb.active

ws["B9"] = mean_accuracy_A
ws["C9"] = std_accuracy_A
ws["D9"] = mean_precision_A
ws["E9"] = std_precision_A
ws["F9"] = mean_recall_A
ws["G9"] = std_recall_A
ws["H9"] = mean_f1_A
ws["I9"] = std_f1_A

ws["J9"] = mean_accuracy_B
ws["K9"] = std_accuracy_B
ws["L9"] = mean_precision_B
ws["M9"] = std_precision_B
ws["N9"] = mean_recall_B
ws["O9"] = std_recall_B
ws["P9"] = mean_f1_B
ws["Q9"] = std_f1_B

# Save the updated Excel file
wb.save("results.xlsx")

'''
```

Task 7 introduces Random Forest classifiers with grid search for hyperparameter tuning.

In Case A, leveraging all features, the Random Forest model demonstrates remarkable performance across metrics. With an average accuracy of 0.912 and an F1 score of 0.909, it achieves robust predictive capabilities. The precision, recall, and F1 scores' tight standard deviations (0.013, 0.018, and 0.003, respectively) signify consistent and reliable performance across cross-validation folds. Compared to previous tasks,
Task 7's Case A outperforms all previous cases, showcasing the efficacy of Random Forests in handling multiple features

However, Case B, utilizing only the two best PCA-derived features,
experiences a notable drop in performance compared to Case A. While maintaining acceptable values, with an average accuracy of 0.761 and an F1 score of 0.745, there's a clear decrease in predictive power and consistency. The larger standard deviations in precision, recall, and F1 score (0.027, 0.057, and 0.017, respectively) indicate greater variability in performance across folds. This parallels the findings from Task 6, emphasizing the importance of feature selection and its impact on model performance.

The stark contrast in performance between Case A and Case B underscores the significance of feature selection in model accuracy. It reaffirms that a larger feature set can provide more information for models to generalize better, while reduced feature sets might sacrifice predictive performance. These observations align with the findings across previous tasks, emphasizing the critical role of feature engineering
in enhancing model accuracy and reliability.
'''

T8. For the best model (according to the F_1-score) that you have obtained for your dataset:

   a. Generate the classification map for **case B** using the provided source code given the *model* object, and the test data (X_{te}, y_{te}).

   b. Load file `ds`*gg*`_samples.csv` (*gg* is the group number) and classify the 4 samples x contained therein, indicating also the probability *a posteriori* for each class $p(\omega_i|x)$. Notice that, in this way, we can say, for the involved track, the probability of becoming a *highly popular* song (according to the training data available). HINT: The `predict_proba` method of *scikit-learn* classification models of tasks T4 – T7 provide **unnormalized** probabilities per class, i.e. the probabilities do not sum to 1, and so you have to normalize them. Notice that for the cases of tasks T1 – T3 you already get unnormalized probabilities.

```
import pandas as pd
```

```
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score, RepeatedKFold, cross_val_predict
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from openpyxl import load_workbook
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Load the dataset
gg = 4  # Replace with your group number
df = pd.read_csv('ds%02d_alt.csv' % (gg))

# Select features and target variable
features = df.iloc[:, :-1]
target = df.iloc[:, -1]

# Case B: Best two features according to PCA
pca = PCA(n_components=2)
features_B = pca.fit_transform(features)
scaler_B = MinMaxScaler()  # Create a separate scaler for Case B
X_B = scaler_B.fit_transform(features_B)

# Parameters for grid search
param_grid = {
   'n_estimators': [20, 40, 60],
   'min_samples_leaf': [5, 10],
   'criterion': ['gini', 'entropy']
}

# Number of repetitions and folds
num_repetitions = 3
num_folds = 5

accuracy_values_B = []
precision_values_B = []
recall_values_B = []
f1_values_B = []

# Loop for repetitions
for i in range(num_repetitions):
   # Case A: All features
   # Case B: Best two features according to PCA
   rf_classifier_B = RandomForestClassifier()
```

```
   kf_B = RepeatedKFold(n_splits=num_folds, n_repeats=1)

   # Perform cross-validation for Case B
   grid_search_B = GridSearchCV(estimator=rf_classifier_B, param_grid=param_grid, cv=kf_B)
   grid_search_B.fit(X_B, target)
   best_rf_B = grid_search_B.best_estimator_

   accuracy_values_B.extend(cross_val_score(best_rf_B, X_B, target, cv=kf_B, scoring='accuracy'))
   predictions_B = cross_val_predict(best_rf_B, X_B, target, cv=kf_B)
   precision_values_B.extend(precision_score(target, predictions_B, average=None))
   recall_values_B.extend(recall_score(target, predictions_B, average=None))
   f1_values_B.extend(f1_score(target, predictions_B, average=None))

X_train_B, X_test_B, y_train_B, y_test_B = train_test_split(X_B, target, test_size=0.3, random_state=42)
Xte = X_test_B
yte = y_test_B

def plot_class(c, X, y):
   m1 = ['k', 'w']
   m2 = ['x', 'o']
   i = np.where(y == c)[0]
   plt.scatter(X[i, 0], X[i, 1], c=m1[c], marker=m2[c], label='class %d' % (c))

x1lim = [Xte[:, 0].min(), Xte[:, 0].max()]
x2lim = [Xte[:, 1].min(), Xte[:, 1].max()]

npts = 100
x1s = np.linspace(x1lim[0], x1lim[1], npts)
x2s = np.linspace(x2lim[0], x2lim[1], npts)

m = np.zeros((npts, npts))
for k1, x1 in enumerate(x1s):
   for k2, x2 in enumerate(x2s):
      x = np.array([x1, x2])
      m[k1, k2] = best_rf_B.predict([x])

plt.figure()
plt.imshow(m.T, cmap='RdYlGn', origin='lower', extent=(x1lim[0], x1lim[1], x2lim[0], x2lim[1]))
for c in range(len(np.unique(yte))):
   plot_class(c, Xte, yte)
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
```

```
plt.legend()
plt.show()

# Load the dataset and perform PCA transformation for X_samples
gg_samples = 4  # Replace with your group number
df_samples = pd.read_csv('ds%02d_samples_alt.csv' % (gg_samples))

# Select features for X_samples
features_samples = df_samples.iloc[:, :-1]

# Apply PCA transformation on features_samples using the same PCA object used for the training data
features_B_samples = pca.transform(features_samples)
X_samples = scaler_B.transform(features_B_samples)

# Classify the samples and get the posterior probability for each class
predictions_samples = best_rf_B.predict(X_samples)
probabilities_samples = best_rf_B.predict_proba(X_samples)

# Normalize probabilities
normalized_probabilities_samples = probabilities_samples / np.sum(probabilities_samples, axis=1)[:, np.newaxis]

# Display the predictions and normalized probabilities for X_samples
for i, prediction_sample in enumerate(predictions_samples):
  print(f"Sample {i+1}: Predicted class - {prediction_sample}, Normalized probabilities - {normalized_probabilities_samples[i]}")

# Update the Excel file with predictions and probabilities for each sample
wb = load_workbook(filename="results.xlsx")
ws = wb.active

# Write the probabilities and predicted classes for each sample
for i, (prediction, probabilities) in enumerate(zip(predictions_samples, normalized_probabilities_samples), start=14):
  ws[f'D{i}'] = prediction  # Write the predicted class in column D

  # Write the normalized probabilities for class 1 (w1) in column B
  ws[f'B{i}'] = probabilities[0]  # Probability for class w1

  # Write the normalized probabilities for class 2 (w2) in column C
  ws[f'C{i}'] = probabilities[1]  # Probability for class w2

# Save the updated Excel file
wb.save("results.xlsx")
```

Full name: Iván López Muñoz **2023/24**
Full name: David Suárez Campione **Group: 04**

(*1) In all cases, you have to evaluate the model performance:

a. For tasks T1 – T3, use a train set/test set strategy (*holdout cross validation*), with 70% of the dataset for training and 30% for testing.

b. For tasks T4 – T7, use *3-repetitions, 5-fold cross validation*. Report on the average values and the standard deviations.