solution

Tema 1 ML - Paunoiu Darius Alexandru¶

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Assuming df is your DataFrame after loading the CSV
statistics_df = pd.read_csv("date_tema_1_iaut_2024.csv")
pd.set_option('display.max_columns', None)

RANDOM_STATES = [42, 10, 15, 21, 13, 30, 35, 37, 45, 53]
RANDOM_STATE = RANDOM_STATES[0]
# List of categorical columns you mentioned
def prelucrate_data(df):
    df['Sedentary_hours_daily'] = df['Sedentary_hours_daily'].str.replace(',', '.').astype(f
    df['Age'] = df['Age'].str.replace(',', '.').astype(float).astype(int)
    df['Est_avg_calorie_intake'] = df['Est_avg_calorie_intake'].astype(int)
    df['Height'] = df['Height'].str.replace(',', '.').astype(float)
    df['Water_daily'] = df['Water_daily'].str.replace(',', '.').astype(float)
    df['Weight'] = df['Weight'].str.replace(',', '.').astype(float)
    df['Physical_activity_level'] = df['Physical_activity_level'].str.replace(',', '.').asty
    df['Technology_time_use'] = df['Technology_time_use'].astype(object)
    df['Main_meals_daily'] = df['Main_meals_daily'].str.replace(',', '.').astype(float).asty
    df['Regular_fiber_diet'] = df['Regular_fiber_diet'].str.replace(',', '.').astype(float)

prelucrate_data(statistics_df)
print(statistics_df.dtypes)

# Splitting the DataFrame into train and test datasets
train_df, test_df = train_test_split(statistics_df, test_size=0.2, random_state=42)

# Printing the shapes of the train and test datasets
print("Train dataset shape:", train_df.shape)
print("Test dataset shape:", test_df.shape)
```

```
statistics_df.tail()
```

```
Transportation                  object
Regular_fiber_diet              object
Diagnostic_in_family_history    object
High_calorie_diet               object
Sedentary_hours_daily          float64
Age                              int64
Alcohol                         object
Est_avg_calorie_intake           int64
Main_meals_daily                object
Snacks                          object
Height                         float64
Smoker                          object
Water_daily                    float64
Calorie_monitoring              object
Weight                         float64
Physical_activity_level        float64
Technology_time_use             object
Gender                          object
Diagnostic                      object
dtype: object
Train dataset shape: (1536, 19)
Test dataset shape: (385, 19)
```

Out[ ]:

	Transportation	Regular_fiber_diet	Diagnostic_in_family_history	High_calorie_diet	Sedentary_h
1916	Public_Transportation	3	yes	yes	
1917	Public_Transportation	3	yes	yes	
1918	Public_Transportation	3	yes	yes	
1919	Public_Transportation	3	yes	yes	
1920	Public_Transportation	3	yes	yes	

In [ ]:

```
# Class distribution overall
class_counts = statistics_df['Diagnostic'].value_counts()
class_counts.plot(kind='bar')
plt.xlabel('Diagnostic')
plt.ylabel('Count')
plt.title('Overall Class Distribution')
plt.show()
```

In [ ]:

```
train_class_counts = train_df['Diagnostic'].value_counts()
test_class_counts = test_df['Diagnostic'].value_counts()

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
train_class_counts.plot(kind='bar')
plt.xlabel('Diagnostic')
plt.ylabel('Count')
plt.title('Train Dataset Class Distribution')

plt.subplot(1, 2, 2)
test_class_counts.plot(kind='bar')
plt.xlabel('Diagnostic')
plt.ylabel('Count')
plt.title('Test Dataset Class Distribution')

plt.tight_layout()
plt.show()
```

In [ ]:

```
import pandas as pd
import numpy as np
from scipy.stats import tmean, tstd, median_abs_deviation, iqr, tmin, tmax

# Identify numerical columns
numerical_columns = statistics_df.select_dtypes(include=['int64', 'float64']).columns

# Initialize a dictionary to store the results
results = {}

# Calculate the required statistics for each numerical column
for col in numerical_columns:
    results[col] = {
        'Mean': tmean(statistics_df[col]),
        'Standard Deviation': tstd(statistics_df[col]),
        'Mean Absolute Deviation': np.mean(np.abs(statistics_df[col] - np.mean(statistics_df
        'Min': tmin(statistics_df[col]),
        'Max': tmax(statistics_df[col]),
        'Difference between Min and Max': tmax(statistics_df[col]) - tmin(statistics_df[col]
        'Median': np.median(statistics_df[col]),  # SciPy does not have a median function
        'Median Absolute Deviation': median_abs_deviation(statistics_df[col]),
        'Interquartile Range': iqr(statistics_df[col]),
    }

# Convert the results to a DataFrame
stats_df = pd.DataFrame(results).transpose()
```

```
stats_df
```

Out[ ]:

	Mean	Standard Deviation	Mean Absolute Deviation	Min	Max	Difference between Min and Max
Sedentary_hours_daily	3.693571	21.759835	1.133885	2.21	956.58	
Age	44.454971	633.322337	40.949876	15.00	19685.0	
Est_avg_calorie_intake	2253.687663	434.075794	375.362344	1500.00	3000.00	
Height	3.573488	58.098160	3.738525	1.45	1915.00	
Water_daily	2.010367	0.611034	0.470801	1.00	3.00	
Weight	205.637344	3225.653536	254.647671	-1.00	82628.0	
Physical_activity_level	1.012640	0.855526	0.702160	0.00	3.00	

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Identify discrete, nominal or ordinal columns
categorical_columns = statistics_df.select_dtypes(include=['object', 'category', 'int8']).co
print(categorical_columns)
# Initialize a dictionary to store the results
results = {}

# Calculate the count of unique values for each column
for col in categorical_columns:
    results[col] = statistics_df[col].nunique()

# Convert the results to a DataFrame
unique_counts_df = pd.DataFrame.from_dict(results, orient='index', columns=['Unique Count'])

# Display the DataFrame
print(unique_counts_df)

# Plot a histogram for each column
for col in categorical_columns:
    statistics_df[col].value_counts().plot(kind='bar', title=col)
    plt.show()
```

```
Index(['Transportation', 'Regular_fiber_diet', 'Diagnostic_in_family_history',
       'High_calorie_diet', 'Alcohol', 'Main_meals_daily', 'Snacks', 'Smoker',
       'Calorie_monitoring', 'Technology_time_use', 'Gender', 'Diagnostic'],
      dtype='object')
                            Unique Count
Transportation                         5
Regular_fiber_diet                     4
```

```
Diagnostic_in_family_history            2
High_calorie_diet                       2
Alcohol                                 4
Main_meals_daily                        4
Snacks                                  4
Smoker                                  2
Calorie_monitoring                      2
Technology_time_use                     4
Gender                                  2
Diagnostic                              7
```

In [ ]:

```
import pandas as pd
from IPython.display import display

for column in statistics_df.columns:
    if statistics_df[column].dtype == 'object':
        statistics_df[column] = statistics_df[column].astype('category').cat.codes

cov_attributes = statistics_df.cov()
display(cov_attributes)
```

Transportation	Regular_fiber_diet	Diagnostic_in_family_history	High_calorie_diet	Sedentary_h
Transportation	1.613173	0.088864	-0.048862	-0.
Regular_fiber_diet	0.088864	0.356925	0.003358	-0.
Diagnostic_in_family_history	-0.048862	0.003358	0.148416	0.0
High_calorie_diet	-0.029535	-0.012392	0.024699	0.1
Sedentary_hours_daily	0.422834	-0.137983	0.094913	0.0
Age	8.382093	-4.589061	4.174651	2.4
Alcohol	-0.015630	-0.027903	0.006592	-0.
Est_avg_calorie_intake	8.080411	-7.286559	-6.864424	2.7
Main_meals_daily	-0.001890	0.069033	0.015612	-0.
Snacks	-0.026046	-0.027504	0.031439	0.0
Height	1.200332	-0.415074	0.347992	0.2
Smoker	-0.001645	0.002715	0.000649	-0.
Water_daily	0.040653	0.037126	0.036532	0.0
Calorie_monitoring	0.008656	0.008733	-0.012481	-0.
Weight	82.964571	-25.592608	27.775319	16.
Physical_activity_level	0.013698	-0.003526	-0.017956	-0.
Technology_time_use	0.146976	-0.035743	0.006897	0.0
Gender	-0.087938	-0.096159	0.019265	0.0
Diagnostic	0.031402	0.245788	0.385575	0.1

In [ ]:

```
import seaborn as sns
correlation_matrix = statistics_df.corr()

# Round the values to a maximum of 3 decimals
rounded_corr_matrix = correlation_matrix.round(2)

# Create a heatmap of the sorted and rounded correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(rounded_corr_matrix, annot=True, cmap='coolwarm')

# Set the title of the heatmap
plt.title('Sorted and Rounded Correlation Matrix')

# Display the heatmap
plt.show()
```

In [ ]:

```
import ast
from sklearn.base import ClassifierMixin
from sklearn.discriminant_analysis import StandardScaler
from sklearn.impute import IterativeImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    f1_score,
    make_scorer,
    precision_score,
    recall_score,
)
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import (
    SelectPercentile,
    VarianceThreshold,
    chi2,
    f_classif,
)
from sklearn.model_selection import GridSearchCV

from matplotlib.backends.backend_pdf import PdfPages

def prepare_dataset():
    df = pd.read_csv("date_tema_1_iaut_2024.csv")
    prelucrate_data(df)
    # Replace -1 with NaN in the 'Weight' column
```

```
    df["Weight"] = df["Weight"].replace(-1, np.nan)

    # Initialize the IterativeImputer
    imputer = IterativeImputer()

    # Perform the imputation on the 'Weight' column
    df["Weight"] = imputer.fit_transform(df[["Weight"]])

    # Convert categorical columns to numerical
    le = LabelEncoder()
    for col in df.columns:
        df[col] = le.fit_transform(df[col])

    X = df.drop("Diagnostic", axis=1)
    y = df["Diagnostic"]

    # Create a VarianceThreshold object
    selector = VarianceThreshold(threshold=0.1)

    # Fit and transform the selector to the data
    features_before = X.columns
    X = pd.DataFrame(
        selector.fit_transform(X), columns=X.columns[selector.get_support()]
    )
    print(f"Features removed: {set(features_before) - set(X.columns)}")

    # Create a SelectPercentile object
    selector = SelectPercentile(f_classif, percentile=70)

    # Fit and transform the selector to the data
    features_before = X.columns
    X = pd.DataFrame(
        selector.fit_transform(X, y), columns=X.columns[selector.get_support()]
    )
    print(f"Features removed: {set(features_before) - set(X.columns)}")

    # Standardize the features
    scaler = StandardScaler()
    X = scaler.fit_transform(X, y)
    return X, y


def find_best_params(classifier, param_grid, X, y, random_state=42):

    # Create train test
    X_train, X_test, y_train, y_test = train_test_split(
```

```
        X, y, test_size=0.2, random_state=random_state
    )

    # Convert the custom scorer into a scorer that can be used with GridSearchCV

    scorers = {
        "accuracy": make_scorer(accuracy_score),
        "precision": make_scorer(precision_score, average="weighted"),
        "recall": make_scorer(recall_score, average="weighted"),
        "f1": make_scorer(f1_score, average="weighted"),
    }

    for scr in [accuracy_score, f1_score, precision_score, recall_score]:
        for class_label in np.unique(y):
            scorers[f"{scr.__name__}_D{class_label}"] = make_scorer(
                lambda y_true, y_pred, class_label: scr(
                    y_true == class_label, y_pred == class_label
                ),
                greater_is_better=True,
                class_label=class_label,
            )

    # Initialize a GridSearchCV
    grid_search = GridSearchCV(
        estimator=classifier,
        param_grid=param_grid,
        cv=5,
        scoring=scorers,
        refit="f1",
        n_jobs=4,
    )
    # Fit the GridSearchCV to the training data
    grid_search.fit(X_train, y_train)

    # Print the best parameters
    print("Best parameters found: ", grid_search.best_params_)

    return grid_search, grid_search.best_params_

from IPython.display import display, Latex, HTML


def evaluate_my_model(
    model: ClassifierMixin, grid_search: GridSearchCV, X, y, random_state=42
):
    # Create a DataFrame from cv_results_
```

```
df = pd.DataFrame(grid_search.cv_results_)

columns = [
    "params",
    "mean_test_accuracy",
    "std_test_accuracy",
    "mean_test_precision",
    "std_test_precision",
    "mean_test_recall",
    "std_test_recall",
    "mean_test_f1",
    "std_test_f1",
]
for scr in [accuracy_score, f1_score, precision_score, recall_score]:
    for class_label in np.unique(y):
        columns.append(f"mean_test_{scr.__name__}_D{class_label}")
        columns.append(f"std_test_{scr.__name__}_D{class_label}")
# Select the columns of interest
df = df[columns]

# Rename the columns
df["params"] = df["params"].apply(lambda x: x.values())
rename_params_to = ",".join([x for x in grid_search.best_params_])
df = df.rename(columns={"params": rename_params_to})


# Highlight the row with the best parameters
def highlight_max(s):
    is_max = s == s.max()
    return ['font-weight: bold' if v else '' for v in is_max]

# df.style.apply(highlight_max)
df.style.highlight_max(color = 'pink', axis = 0)
# df.style.apply(
#     lambda x: [
#         "font-weight: bold" if True else "" for _ in x
#     ],
#     axis=1,
# )
# display(df)
dfs = []
for class_label in np.unique(y):
    cols = [rename_params_to]
    for scr in [accuracy_score, f1_score, precision_score, recall_score]:
        cols.append(f"mean_test_{scr.__name__}_D{class_label}")
        cols.append(f"std_test_{scr.__name__}_D{class_label}")
```

```
    dfs.append(df[cols])
    renamed_cols = [rename_params_to]
    for scr in [accuracy_score, f1_score, precision_score, recall_score]:
        renamed_cols.append(f"{scr.__name__}_D{class_label}")
        renamed_cols.append(f"{scr.__name__}_D{class_label}_std")
    dfs[-1].columns = renamed_cols
    # display(dfs[-1])
    # fig, ax =plt.subplots(figsize=(12,4))
    # ax.axis('tight')
    # ax.axis('off')
    # the_table = ax.table(cellText=dfs[-1].values,colLabels=dfs[-1].columns,loc='center

    # #https://stackoverflow.com/questions/4042192/reduce-left-and-right-margins-in-matp
    # pp = PdfPages(f"foo{class_label}.pdf")
    # pp.savefig(fig, bbox_inches='tight')
    # pp.close()
    # print(dfs[-1].to_latex(
    #     index=False,  # To not include the DataFrame index as a column in the table
    #     # caption="Comparison of ML Model Performance Metrics",  # The caption to appe
    #     # label="tab:model_comparison",  # A label used for referencing the table with
    #     # position="htbp",  # The preferred positions where the table should be placed
    #     # column_format="|l|l|l|l|",  # The format of the columns: left-aligned with v
    #     # escape=False,  # Disable escaping LaTeX special characters in the DataFrame
    #     # float_format="{:0.4f}".format  # Formats floats to two decimal places
    #     ))

    if class_label == 6:
        # dfs[-1].style.set_table_styles([
        # {'selector': 'th', 'props': [('font-size', '10pt')]},
        # {'selector': 'td', 'props': [('font-size', '10pt')]}
        # ])
        # latex = dfs[-1].to_latex(index=False, float_format="{:0.4f}".format)
        # ltx = Latex(latex)
        html = dfs[-1].to_html(index=False)
        display(HTML(html))


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=random_state
)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average="weighted")
precision = precision_score(y_test, y_pred, average="weighted")
recall = recall_score(y_test, y_pred, average="weighted")
```

```
    print(
        classification_report(
            y_test, y_pred, target_names=["D0", "D1", "D2", "D3", "D4", "D5", "D6"]
        )
    )

    return accuracy, f1, precision, recall
```

In [ ]:

```
# Initialize a RandomForestClassifier
clf = RandomForestClassifier(random_state=42)


# Define the parameter grid
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [15, 20, 25],
    'max_samples': [0.5, 0.7, 1.0],
}
X, y = prepare_dataset()
grid_search, best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
```

```
Features removed: {'Calorie_monitoring', 'Smoker'}
Features removed: {'Est_avg_calorie_intake', 'Physical_activity_level', 'Technology_time_use
Best parameters found:  {'max_depth': 20, 'max_samples': 1.0, 'n_estimators': 150}
```

In [ ]:

```
best_clf = RandomForestClassifier(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, grid_search, X, y, RANDOM_STAT
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(grid_search.best_index_)
```

max_depth,max_samples,n_estimators	accuracy_score_D6	accuracy_score_D6_std	f1_score_D6	f
(15, 0.5, 100)	0.995918	0.008163	0.995918	0
(15, 0.5, 150)	0.995918	0.008163	0.995918	0
(15, 0.5, 200)	0.995918	0.008163	0.995918	0
(15, 0.7, 100)	0.995918	0.008163	0.995918	0
(15, 0.7, 150)	0.995918	0.008163	0.995918	0
(15, 0.7, 200)	0.995918	0.008163	0.995918	0
(15, 1.0, 100)	0.995918	0.008163	0.995918	0
(15, 1.0, 150)	0.995918	0.008163	0.995918	0
(15, 1.0, 200)	0.995918	0.008163	0.995918	0

max_depth,max_samples,n_estimators	accuracy_score_D6	accuracy_score_D6_std	f1_score_D6	f
(20, 0.5, 100)	0.995918	0.008163	0.995918	0
(20, 0.5, 150)	0.995918	0.008163	0.995918	0
(20, 0.5, 200)	0.995918	0.008163	0.995918	0
(20, 0.7, 100)	0.995918	0.008163	0.995918	0
(20, 0.7, 150)	0.995918	0.008163	0.995918	0
(20, 0.7, 200)	0.995918	0.008163	0.995918	0
(20, 1.0, 100)	0.995918	0.008163	0.995918	0
(20, 1.0, 150)	0.995918	0.008163	0.995918	0
(20, 1.0, 200)	0.995918	0.008163	0.995918	0
(25, 0.5, 100)	0.995918	0.008163	0.995918	0
(25, 0.5, 150)	0.995918	0.008163	0.995918	0
(25, 0.5, 200)	0.995918	0.008163	0.995918	0
(25, 0.7, 100)	0.995918	0.008163	0.995918	0
(25, 0.7, 150)	0.995918	0.008163	0.995918	0
(25, 0.7, 200)	0.995918	0.008163	0.995918	0
(25, 1.0, 100)	0.995918	0.008163	0.995918	0
(25, 1.0, 150)	0.995918	0.008163	0.995918	0
(25, 1.0, 200)	0.995918	0.008163	0.995918	0

```
              precision    recall  f1-score   support

          D0       0.96      0.80      0.87        60
          D1       0.75      0.88      0.81        52
          D2       0.86      0.86      0.86        42
          D3       0.88      0.90      0.89        49
          D4       0.94      0.86      0.89        69
          D5       0.91      0.98      0.94        60
          D6       0.96      0.98      0.97        53

    accuracy                           0.89       385
   macro avg       0.89      0.89      0.89       385
weighted avg       0.90      0.89      0.89       385

Accuracy: 0.8935064935064935
F1 Score: 0.8937436674764665
Precision: 0.8988330853810947
Recall: 0.8935064935064935
16
```

In [ ]:

```
# Initialize a RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
```

```
clf = ExtraTreesClassifier(random_state=RANDOM_STATE)


# Define the parameter grid
param_grid = {
    'n_estimators': [100, 150, 200, 250],
    'max_depth': [10, 15, 20, 25, 30],
    'max_samples': [0.5, 0.7, 0.8, 1.0],
    'bootstrap': [True]
}
X, y = prepare_dataset()
grid_search, best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
best_clf = ExtraTreesClassifier(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y, RANDOM_STATE)
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

```
Features removed: {'Calorie_monitoring', 'Smoker'}
Features removed: {'Est_avg_calorie_intake', 'Physical_activity_level', 'Technology_time_use
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[177], line 16
      9 param_grid = {
     10     'n_estimators': [100, 150, 200, 250],
     11     'max_depth': [10, 15, 20, 25, 30],
     12     'max_samples': [0.5, 0.7, 0.8, 1.0],
     13     'bootstrap': [True]
     14 }
     15 X, y = prepare_dataset()
---> 16 grid_search, best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
     17 best_clf = ExtraTreesClassifier(**best_params, random_state=RANDOM_STATE)
     18 accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y, RANDOM_STATE)

Cell In[174], line 108, in find_best_params(classifier, param_grid, X, y, random_state)
     99 grid_search = GridSearchCV(
    100     estimator=classifier,
    101     param_grid=param_grid,
   (...)
    105     n_jobs=4,
    106 )
    107 # Fit the GridSearchCV to the training data
--> 108 grid_search.fit(X_train, y_train)
    110 # Print the best parameters
    111 print("Best parameters found: ", grid_search.best_params_)
```

```
File ~/.local/lib/python3.10/site-packages/sklearn/base.py:1474, in _fit_context.<locals>.de
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File ~/.local/lib/python3.10/site-packages/sklearn/model_selection/_search.py:970, in BaseSe
    964     results = self._format_results(
    965         all_candidate_params, n_splits, all_out, all_more_results
    966     )
    968     return results
--> 970 self._run_search(evaluate_candidates)
    972 # multimetric is determined here because in the case of a callable
    973 # self.scoring the return type is only known after calling
    974 first_test_score = all_out[0]["test_scores"]

File ~/.local/lib/python3.10/site-packages/sklearn/model_selection/_search.py:1527, in GridS
   1525 def _run_search(self, evaluate_candidates):
   1526     """Search all candidates in param_grid"""
-> 1527     evaluate_candidates(ParameterGrid(self.param_grid))

File ~/.local/lib/python3.10/site-packages/sklearn/model_selection/_search.py:916, in BaseSe
    908 if self.verbose > 0:
    909     print(
    910         "Fitting {0} folds for each of {1} candidates,"
    911         " totalling {2} fits".format(
    912             n_splits, n_candidates, n_candidates * n_splits
    913         )
    914     )
--> 916 out = parallel(
    917     delayed(_fit_and_score)(
    918         clone(base_estimator),
    919         X,
    920         y,
    921         train=train,
    922         test=test,
    923         parameters=parameters,
    924         split_progress=(split_idx, n_splits),
    925         candidate_progress=(cand_idx, n_candidates),
    926         **fit_and_score_kwargs,
    927     )
    928     for (cand_idx, parameters), (split_idx, (train, test)) in product(
```

```
    929         enumerate(candidate_params),
    930         enumerate(cv.split(X, y, **routed_params.splitter.split)),
    931     )
    932 )
    934 if len(out) < 1:
    935     raise ValueError(
    936         "No fits were performed. "
    937         "Was the CV iterator empty? "
    938         "Were there no candidates?"
    939     )

File ~/.local/lib/python3.10/site-packages/sklearn/utils/parallel.py:67, in Parallel.__call_
     62 config = get_config()
     63 iterable_with_config = (
     64     (_with_config(delayed_func, config), args, kwargs)
     65     for delayed_func, args, kwargs in iterable
     66 )
---> 67 return super().__call__(iterable_with_config)

File ~/.local/lib/python3.10/site-packages/joblib/parallel.py:1952, in Parallel.__call__(sel
   1946 # The first item from the output is blank, but it makes the interpreter
   1947 # progress until it enters the Try/Except block of the generator and
   1948 # reach the first `yield` statement. This starts the aynchronous
   1949 # dispatch of the tasks to the workers.
   1950 next(output)
-> 1952 return output if self.return_generator else list(output)

File ~/.local/lib/python3.10/site-packages/joblib/parallel.py:1595, in Parallel._get_outputs
   1592     yield
   1594     with self._backend.retrieval_context():
-> 1595         yield from self._retrieve()
   1597 except GeneratorExit:
   1598     # The generator has been garbage collected before being fully
   1599     # consumed. This aborts the remaining tasks if possible and warn
   1600     # the user if necessary.
   1601     self._exception = True

File ~/.local/lib/python3.10/site-packages/joblib/parallel.py:1707, in Parallel._retrieve(se
   1702 # If the next job is not ready for retrieval yet, we just wait for
   1703 # async callbacks to progress.
   1704 if ((len(self._jobs) == 0) or
   1705     (self._jobs[0].get_status(
   1706         timeout=self.timeout) == TASK_PENDING)):
-> 1707     time.sleep(0.01)
   1708     continue
   1710 # We need to be careful: the job list can be filling up as
```

```
   1711 # we empty it and Python list are not thread-safe by
   1712 # default hence the use of the lock

KeyboardInterrupt:
```

In [ ]:

```
from xgboost import XGBClassifier

# Initialize a XGBClassifier
clf = XGBClassifier(random_state=RANDOM_STATE)

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [10, 15, 20, 25],
    'learning_rate': [0.01, 0.1, 0.2],
}

X, y = prepare_dataset()
best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
best_clf = XGBClassifier(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y, RANDOM_STATE)
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
```

In [ ]:

```
from sklearn.svm import SVC

# Initialize a SVC
clf = SVC(random_state=RANDOM_STATE)

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid']
}

X, y = prepare_dataset()
best_params = find_best_params(clf, param_grid, X, y, RANDOM_STATE)
best_clf = SVC(**best_params, random_state=RANDOM_STATE)
accuracy, f1, precision, recall = evaluate_my_model(best_clf, X, y, RANDOM_STATE)
print(f"Accuracy: {accuracy}")
print(f"F1 Score: {f1}")
print(f"Precision: {precision}")
```

```
print(f"Recall: {recall}")
```