Lab 1 - Explainable and Trustworthy AI

Teaching Assistant: Eleonora Poeta (eleonora.poeta@polito.it)

Lab 1: Interpretable by design models on structured data

Decision Trees

- Decision trees offer an approach to achieve interpretability-by-design of machine learning models. They give a transparent and intuitive representation of the decision-making process followed by the model. This transparency allows domain experts to easily understand and validate the model's predictions.

When assessing the interpretability of decision trees, several key aspects should be considered depending if assessing *global* or *local* interpretability. In particular, you have to analyze:

1. When assessing **global** interpretability you have to *inspect the entire decision tree*. Then, as mesures for the global interpretability there are:

> - **Depth** of the tree → Shallow trees with fewer levels are easier to interpret, as they represent simpler decision rules. In contrast, deeper trees may become overly complex and difficult to interpret, potentially sacrificing transparency for improved accuracy.
> - **Size** of the tree → This includes the *number of nodes* and the *number of splits*. A larger tree with more nodes and splits may capture intricate patterns in the data but could also lead to overfitting and decreased interpretability.

2. When assessing **local** interpretability you have to *inspect the individual path of a single prediction.* Then, as mesures for the local interpretability there is:

> - **Lenght** of the individual path.

Exercise 1

The *Diabetes prediction dataset* comprises medical and demographic data, alongside diabetes status (positive:1/negative:0) of patients. It includes features like age, gender, body mass index

(BMI), hypertension, heart disease, smoking history, HbA1c level, and blood glucose level. In the following exercise you have to:

- Fit a **Decision tree classifier** model on **Diabetes dataset** and evaluate it calculating model's accuracy.

> - Visualize the decision tree obtained. Are you able to interpret the decision process?
> - Try again with `max_depth=4` and compare the two trees. Which one is the most interpretable?

- Analyze **Global Interpretability**:

> - Continue visualizing the obtained decision tree with `max_depth=4`. Which attributes are the most discriminating? Plot the feature importances and then analyze the values.
> - Calculate the size of the decision tree in terms of the number of nodes, subdivisions, and depth. How these metrics affect the interpretability of the decision tree globally?

- Analyze **Local Interpretability**:

> - Consider the instances 100, 150 and 200 of the train dataset.
> - What are the individual paths? What are the instances allocated in the paths?
> - For each of the previous instances, calculate the length of each path from the root node to the leaf node to which the instance belongs. How the length of these paths contributes to the interpretability of the decision tree locally?

Hint :

> - Before starting do some **preprocessing** of the Diabetes dataset as previously seen in Lab 0.1 (Address null values and preprocess categorical attributes.)
> - As **split ratio** for the dataset use the standard one: train (80%) and test (20%). Account for any class imbalance during the train-test split by making use of the **stratify** argument

Solution:

Imports

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


# If your dataset is stored on Google Drive, mount the drive before reading it
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

Data Preprocessing

```
#Read data from CSV file stored in Google Drive and visualize the first 10 rows
df = pd.read_csv('/content/drive/MyDrive/XAI 2025/Lab 1/diabetes.csv')
# Otherwise, if you are working on GitHub run the following command
# !wget url github
```

```
df.head(10)
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_leve
0	Female	80.0	0	1	never	25.19	6.
1	Female	54.0	0	0	No Info	27.32	6.
2	Male	28.0	0	0	never	27.32	5.
3	Female	36.0	0	0	current	23.45	5.
4	Male	76.0	1	1	current	20.14	4.
5	Female	20.0	0	0	never	27.32	6.
6	Female	44.0	0	0	never	19.31	6.
7	Female	79.0	0	0	No Info	23.86	5.
8	Male	42.0	0	0	never	33.64	4.
9	Female	32.0	0	0	never	27.32	5.

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
# Check if the dataset is balanced
df.diabetes.value_counts()
```

	count
diabetes	
0	91500
1	8500

dtype: int64

Check for duplicate values

```
# check for duplicate rows
duplicates = df.duplicated(keep=False)
print(f"Number of duplicate rows: {duplicates.sum()}")
```

```
Number of duplicate rows: 6939
```

```
df_duplicates = df.loc[duplicates]
df_duplicates
```

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_l
1	Female	54.0	0	0	No Info	27.32	
10	Female	53.0	0	0	never	27.32	
14	Female	76.0	0	0	No Info	27.32	
18	Female	42.0	0	0	No Info	27.32	
41	Male	5.0	0	0	No Info	27.32	
...	...	...	...	...	...	...	
99980	Female	52.0	0	0	never	27.32	
99985	Male	25.0	0	0	No Info	27.32	
99989	Female	26.0	0	0	No Info	27.32	
99990	Male	39.0	0	0	No Info	27.32	
99995	Female	80.0	0	0	No Info	27.32	

6939 rows × 9 columns

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
# Remove duplicates
df.drop_duplicates(inplace=True)

# check for duplicate rows
duplicates = df.duplicated(keep=False)
```

```
print(f"Number of duplicate rows: {duplicates.sum()}")
print(f"New number of samples after removing duplicates: {len(df)}")
```

```
Number of duplicate rows: 0
New number of samples after removing duplicates: 96146
```

```
# Split into training and test set
df_train, df_test = train_test_split(df, test_size=0.2, shuffle=True, random_stat
```

```
# Print the number of samples in training and test set
print(f"Number of training examples: {len(df_train)}")
print(f"Number of test examples: {len(df_test)}")
```

```
Number of training examples: 76916
Number of test examples: 19230
```

Check for missing values

```
print(f'Are there any null values? Training: {df_train.isnull().values.any()}, Te
```

```
Are there any null values? Training: False, Test: False
```

```
nan_count_train = df_train.isna().sum()
nan_count_test = df_test.isna().sum()
```

```
print("Train")
print(nan_count_train)
```

```
Train
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
diabetes               0
dtype: int64
```

```
print("Test")
print(nan_count_test)
```

```
Test
gender                 0
age                    0
hypertension           0
heart_disease          0
smoking_history        0
bmi                    0
HbA1c_level            0
blood_glucose_level    0
```

```
diabetes                0
dtype: int64
```

Discretize age column.

```
age_category = ['Child (0-14]', 'Young (14-24]', 'Adults (24-50]', 'Senior (50-10

df_train['age_disc']=pd.cut(x=df_train['age'], bins=[0,14,24,50,100],labels=age_c
df_train = df_train.drop(columns=['age']) # Remove the old age column

df_test['age_disc']=pd.cut(x=df_test['age'], bins=[0,14,24,50,100],labels=age_cat
df_test = df_test.drop(columns=['age']) # Remove the old age column
```

```
print(list(set(df_train.smoking_history.tolist())))
```

```
['never', 'No Info', 'former', 'ever', 'not current', 'current']
```

```
print(list(set(df_test.smoking_history.tolist())))
```

```
['No Info', 'current', 'former', 'ever', 'not current', 'never']
```

```
print(df_train.smoking_history.value_counts())
```

```
smoking_history
never          27509
No Info        26307
former          7476
current         7349
not current     5108
ever            3167
Name: count, dtype: int64
```

```
print(df_test.smoking_history.value_counts())
```

```
smoking_history
never          6889
No Info        6580
current        1848
former         1823
not current    1259
ever            831
Name: count, dtype: int64
```

Combine not current and former

```
df_train.loc[df_train['smoking_history'] == 'former', 'smoking_history'] = 'not c
df_test.loc[df_test['smoking_history'] == 'former', 'smoking_history'] = 'not cur
```

```
print(df_train.smoking_history.value_counts())
```

```
smoking_history
never          27509
No Info        26307
not current    12584
current         7349
ever            3167
Name: count, dtype: int64
```

```
print(df_test.smoking_history.value_counts())
```

```
smoking_history
never          6889
No Info        6580
not current    3082
current        1848
ever            831
Name: count, dtype: int64
```

```
df_train_encoded = df_train.copy()
df_test_encoded = df_test.copy()
```

```
df_train_encoded.head()
```

	gender	hypertension	heart_disease	smoking_history	bmi	HbA1c_level
79000	Male	0	0	No Info	23.87	5.7
32011	Female	0	0	not current	33.03	4.0
95559	Female	0	0	No Info	27.32	6.6

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
smoking_history_order = ["never", "not current", "No Info", "current", "ever"]
```

```
from sklearn.preprocessing import OrdinalEncoder

# Instantiate the OrdinalEncoder specifying the list of the categories
ord_enc = OrdinalEncoder(categories=[smoking_history_order, age_category])

# Fit the OrdinalEncoder on training data
ord_enc.fit(df_train_encoded[['smoking_history', 'age_disc']])

ord_enc
```

```
▾                        OrdinalEncoder
OrdinalEncoder(categories=[['never', 'not current', 'No Info', 'current',
                            'ever'],
                           ['Child (0-14]', 'Young (14-24]', 'Adults (24-50]'
                            'Senior (50-100]']])
```

```
df_train_encoded[["smoking_history", "age_disc"]] = ord_enc.transform(df_train_en
df_test_encoded[["smoking_history", "age_disc"]] = ord_enc.transform(df_test_enco
```

```
df_train_encoded.head()
```

	gender	hypertension	heart_disease	smoking_history	bmi	HbA1c_level
79000	Male	0	0	2.0	23.87	5.7
32011	Female	0	0	1.0	33.03	4.0
95559	Female	0	0	2.0	27.32	6.6
32057	Male	1	0	2.0	28.86	4.8
97797	Female	0	0	1.0	26.48	6.5

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
df_test_encoded.head()
```

	gender	hypertension	heart_disease	smoking_history	bmi	HbA1c_level
82004	Female	0	0	3.0	36.77	6.6
10542	Male	0	0	2.0	22.29	4.5
31572	Female	1	0	1.0	34.24	6.2
98055	Male	0	0	1.0	24.39	4.0
49107	Male	0	1	2.0	35.00	4.5

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
print(df_train_encoded.gender.value_counts())
```

```
gender
Female    44817
Male      32085
Other        14
Name: count, dtype: int64
```

```
print(df_test_encoded.gender.value_counts())
```

```
gender
Female    11344
Male       7882
Other         4
Name: count, dtype: int64
```

```
# Remove all the rows where gender = 'Other'

df_train_encoded = df_train_encoded[df_train_encoded['gender'] != 'Other']
df_test_encoded = df_test_encoded[df_test_encoded['gender'] != 'Other']
```

```
print(df_train_encoded.gender.value_counts())
```

```
gender
Female    44817
Male      32085
Name: count, dtype: int64
```

```
print(df_test_encoded.gender.value_counts())
```

```
gender
Female    11344
Male       7882
Name: count, dtype: int64
```

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(handle_unknown='ignore')

ohe_categorical_columns = ['gender']

# Fit the one-hot encoder on training data
ohe.fit(df_train_encoded[ohe_categorical_columns])

# Create a new DataFrame with only the one-hot encoded columns
temp_df_train = pd.DataFrame(data=ohe.transform(df_train_encoded[ohe_categorical_
                             columns=ohe.get_feature_names_out())

# Remove the old categorical columns from the original data
df_train_encoded.drop(columns=ohe_categorical_columns, axis=1, inplace=True)
df_train_encoded = pd.concat([df_train_encoded.reset_index(drop=True), temp_df_tr

# Perform the same procedure on the test set
temp_df_test = pd.DataFrame(data=ohe.transform(df_test_encoded[ohe_categorical_co
                       columns=ohe.get_feature_names_out())

df_test_encoded.drop(columns=ohe_categorical_columns, axis=1, inplace=True)
df_test_encoded = pd.concat([df_test_encoded.reset_index(drop=True), temp_df_test
```

```
<ipython-input-35-8526baaa8557>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
```

```
  df_train_encoded.drop(columns=ohe_categorical_columns, axis=1, inplace=True
<ipython-input-35-8526baaa8557>:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_test_encoded.drop(columns=ohe_categorical_columns, axis=1, inplace=True)
```

```
df_train_encoded.head()
```

	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_gluc
0	0	0	2.0	23.87	5.7	
1	0	0	1.0	33.03	4.0	
2	0	0	2.0	27.32	6.6	
3	1	0	2.0	28.86	4.8	
4	0	0	1.0	26.48	6.5	

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
df_test_encoded.head()
```

	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_gluc
0	0	0	3.0	36.77	6.6	
1	0	0	2.0	22.29	4.5	
2	1	0	1.0	34.24	6.2	
3	0	0	1.0	24.39	4.0	
4	0	1	2.0	35.00	4.5	

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
from sklearn.preprocessing import MinMaxScaler

features_to_normalize = ['bmi', 'HbA1c_level', 'blood_glucose_level', 'age_disc',

minmax_s = MinMaxScaler()

minmax_s.fit(df_train_encoded[features_to_normalize])

df_train_encoded[features_to_normalize] = minmax_s.transform(df_train_encoded[fea
df_test_encoded[features_to_normalize] = minmax_s.transform(df_test_encoded[featu
```

```
df_train_encoded.head()
```

	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_gl
0	0	0	0.50	0.162657	0.400000	
1	0	0	0.25	0.270156	0.090909	
2	0	0	0.50	0.203145	0.563636	
3	1	0	0.50	0.221218	0.236364	
4	0	0	0.25	0.193287	0.545455	

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
df_test_encoded.head()
```

	hypertension	heart_disease	smoking_history	bmi	HbA1c_level	blood_gl
0	0	0	0.75	0.314048	0.563636	
1	0	0	0.50	0.144115	0.181818	
2	1	0	0.25	0.284356	0.490909	
3	0	0	0.25	0.168760	0.090909	
4	0	1	0.50	0.293275	0.181818	

Passaggi successivi: Visualizza grafici consigliati New interactive sheet

```
# Extract target variable and input features for the training data
y_train = df_train_encoded['diabetes']
X_train = df_train_encoded.drop('diabetes', axis=1)


# Extract target variable and input features for  the testing data
y_test = df_test_encoded['diabetes']
X_test = df_test_encoded.drop('diabetes', axis=1)
```

Fit the decision tree

```
# define the model
model = tree.DecisionTreeClassifier()
# fit the model
model.fit(X_train, y_train)
# extract the feature names
feature_names = X_train.columns
tree.plot_tree(model, feature_names=feature_names)
```