Introduction to ML

Data Preprocessing & KNN

Quan Minh Phan & Ngoc Hoang Luong

University of Information Technology
-
Vietnam National University Ho Chi Minh City

June 19, 2023

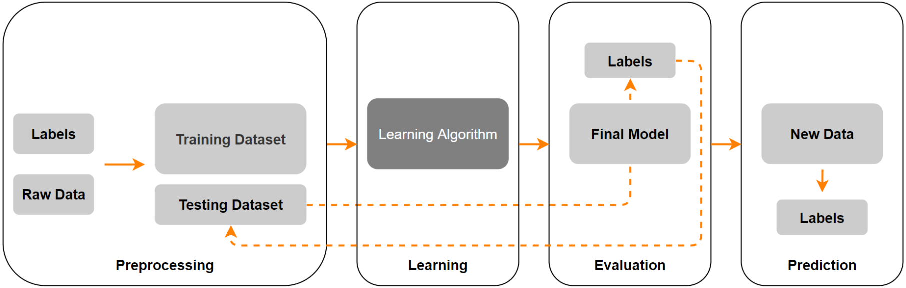
Overview

1. A roadmap for building machine learning system
2. Data Pre-processing
3. K-Nearest Neighbors
4. Model Evaluation

Roadmap

5 major steps:

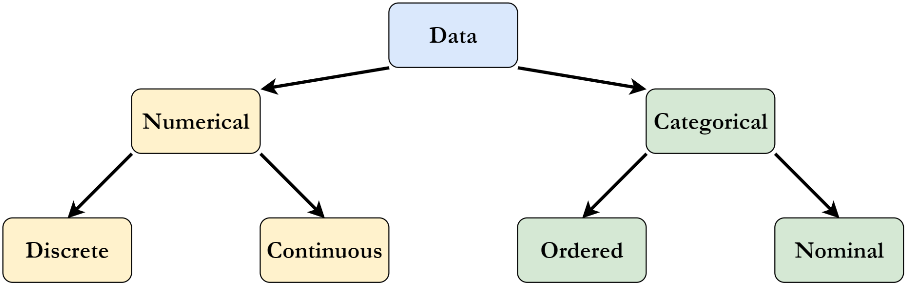
- Data Pre-processing
- Model Learning
- Model Evaluation
- Prediction
- Model Deployment



Overview

1. A roadmap for building machine learning system
2. Data Pre-processing
3. K-Nearest Neighbors
4. Model Evaluation

Types of Data



Numerical: quantitative data

- Discrete: #students in a class, the age of a person, . . .
- Continuous: the height of a person, the wind speed, ...

Categorical: qualitative data

- Ordered: food ratings (excellent, good, bad), feelings (happy, neutral, bad), . . .
- Nominal: the name of students, . . .

How to load data?

Syntax (load)

pandas.read_csv(*filepath*)

Examples

$>>$ **import** pandas **as** pd

$>>$ df $=$ pd.read_csv($'/content/toy\_dataset.csv'$)

Syntax (show n data points)

pandas.DataFrame.head(n)

Examples

$>>$ df.head($n = 5$)

Data Representation

Independent variables

	ID	Age	Sex	Height	Grade	Rich?
0	1	21.0	Male	171.0	Good	Yes
1	2	20.0	Male	170.0	Bad	No
2	3	19.0	Female	NaN	Bad	No
3	4	17.0	Male	165.0	Excellent	Yes
4	5	NaN	Female	166.0	Good	Yes

Target variable

Data Cleaning

- **Independent variables** should NOT contain
  - Missing or NULL values
  - Outliers
  - Data on different scales ([10, 30] vs [1M, 1B])
  - Special characters (*, ?, %, #)
  - ...
- **Data Cleaning:** The processes of detecting and correcting (or removing) missing values or outliers.
  - Ensuring data is correct, consistent and usable.

Missing values

- In .csv files, missing values are usually represented as empty, 'NA', 'N/A', 'null', 'nan', 'NaN'.

	ID	Age	Sex	Height	Grade	Rich?
0	1	21.0	Male	171.0	Good	Yes
1	2	20.0	Male	170.0	Bad	No
2	3	19.0	Female	NaN	Bad	No
3	4	17.0	Male	165.0	Excellent	Yes
4	5	NaN	Female	166.0	Good	Yes

Missing values (cont.)

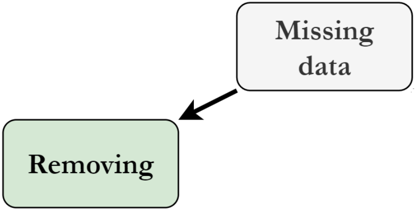
Syntax (count #'NaN' each column)

pandas.DataFrame.isna().sum()

Examples

```
>> df_1 = pd.read_csv('content/toy_dataset.csv')
>> countNULL = df_1.isna().sum()
   countNULL
>> null_cols = df_1[countNULL > 0]
   null_cols
```

How to handle?



Removing

Syntax

pandas.DataFrame.dropna(*inplace*)

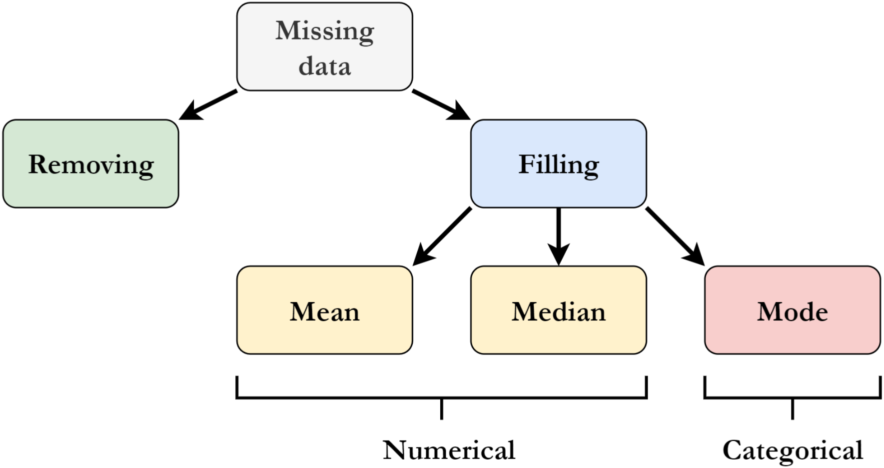
Examples

$>>$ df_1.dropna($inplace = True$)

or

$>>$ df_1 $=$ df_1.dropna($inplace = False$)

How to handle? (cont.)



Filling

Examples

Find the mean, median, and mode for the following list of values:
13, 18, 13, 14, 13, 16, 14, 21, 13

Mean

- $mean = (13 + 18 + 13 + 14 + 13 + 16 + 14 + 21 + 13)/9 = 15$

Median

- Sorting the list: 13, 13, 13, 13, 14, 14, 16, 18, 21
- $median = 14$

Mode

- $mode = 13$

Filling (cont.)

Step 1: Calculating the filling values

Syntax (calculate the mean)

pandas.DataFrame.mean()

Examples

```
>> mean_age = df_1['Age'].mean()
   mean_age
```

Syntax (calculate the median)

pandas.DataFrame.median()

Examples

```
>> median_height = df_1['Height'].median()
   median_height
```

Filling (cont.)

Step 1: Calculating the filling values

Syntax (calculate the mode)

pandas.DataFrame.mode()[0]

Examples

```
>> mode_grade = df_1['Grade'].mode()[0]
   mode_grade
```

Filling (cont.)

Step 2: Replacing 'NaN' by the filling values

Syntax

$$\textbf{pandas.DataFrame.fillna}(value, inplace)$$

Examples

$>>$ df_1['Age'].fillna($value = mean\_age, inplace = True$)

$>>$ df_1['Height'].fillna($value = median\_height, inplace = True$)

$>>$ df_1['Grade'].fillna($value = mode\_grade, inplace = True$)

Outliers (examples)

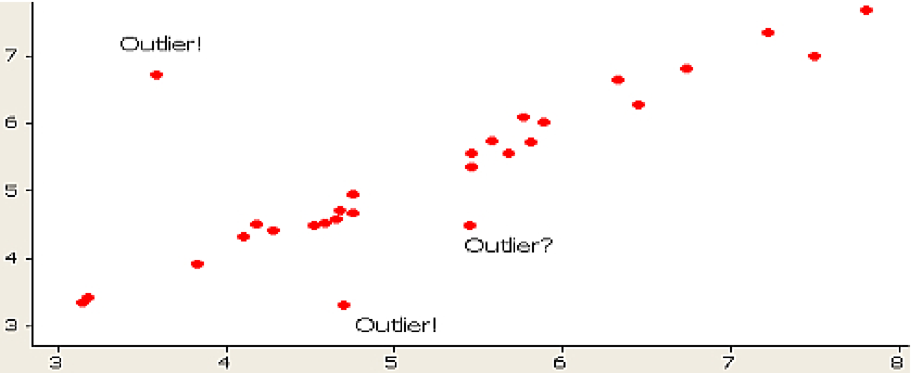


Figure: Examples of outliers

Outliers (cont.)

Syntax (visualize the outliers)

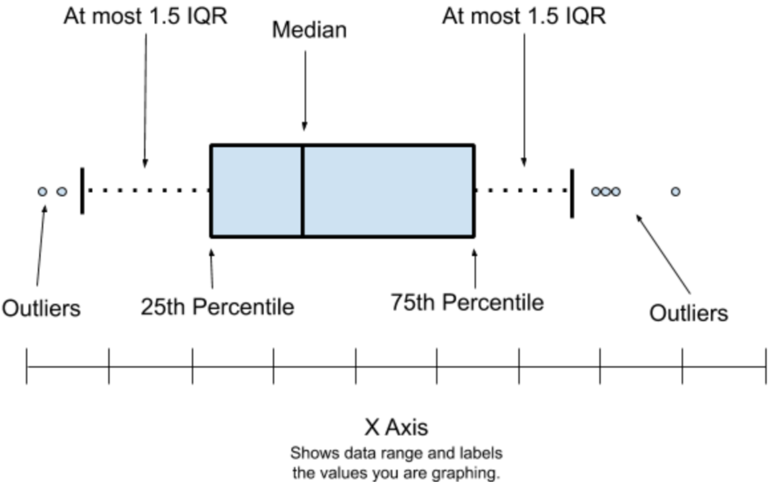
seaborn.boxplot(*data*)

Examples

$>>$ **import** seaborn **as** sbn

sbn.boxplot($df\_1['Height']$)

Outliers (Interquartile Range)



Outliers (cont.)

Examples

Find the outliers on 71, 70, 90, 70, 70, 60, 70, 72, 72, 320, 71, 69

Outliers (cont.)

Examples

Find the outliers on 71, 70, 90, 70, 70, 60, 70, 72, 72, 320, 71, 69

Solution

- Sort the data: 60, 69, 70, 70, 70, 70, 71, 71, 72, 72, 90, 320
- Calculate the median (Q2) $\rightarrow (70 + 71)/2 = 70.5$
- Calculate the lower quartile (Q1) $\rightarrow (70 + 70)/2 = 70.0$
- Calculate the upper quartile (Q3) $\rightarrow (72 + 72)/2 = 72$
- Calculate the interquartile range (IQR) $\rightarrow$ Q3 – Q1 $= 72 - 70 = 2$
- Find the upper and lower fences.
  Lower fence $=$ Q1 – 1.5 * IQR $= 70 - 1.5 * 2 = 67$
  Upper fence $=$ Q3 $+$ 1.5 * IQR $= 71.5 + 1.5 * 2 = 74.5$
- The data points that are lower than the lower fence and greater than the upper fence are outliers $\rightarrow$ outliers: 60; 90; 320.

Outliers (cont.)

Examples

>> Q1 = df_1['Height'].quantile(0.25)

Q3 = df_1['Height'].quantile(0.75)

$>> IQR = Q3 - Q1$

$>> low\_fence = Q1 - (1.5 * IQR)$

$up\_fence = Q3 + (1.5 * IQR)$

>> df_1[((df_1['Height'] < low_fence)|(df_1['Height'] > up_fence))]

>> df_1 = df_1[~((df_1)|(df_1['Height'] > up_fence))]

Data Transformation (Label Encoding)

Label Encoding: replacing each value in a categorical column with numbers from 0 to $N - 1$

Syntax (initialize)

sklearn.preprocessing.LabelEncoder()

Examples

```
>> from sklearn.preprocessing import LabelEncoder
   label_encoder = LabelEncoder()
```

Label Encoding

Syntax (fit & transform)

LabelEncoder().fit_transform(X)

Examples

```
>> df_1['Sex'] = label_encoder.fit_transform(df_1['Sex'])
   df_1.reset_index(drop = True, inplace = True)
```

Data Transformation (One-hot Encoding)

One-hot Encoding: dividing a categorical column into *n* number of columns with *n* is the total number of unique labels in that column.

Syntax (initialize)

sklearn.preprocessing.OneHotEncoder()

Examples

```
>> from sklearn.preprocessing import OneHotEncoder
   onehot_encoder = OneHotEncoder(sparse_output = False)
```

One-hot Encoding

Syntax (fit & transform)

OneHotEncoder().fit_transform(X)

Examples

>> variable = 'Grade'

>> encoded_data = onehot_encoder.fit_transform(df_1[[variable]])

>> encoded_col = pd.DataFrame(data=encoded_data, columns=onehot_encoder.get_feature_names_out([variable]))

>> df_1 = pd.concat([df_1.drop(columns=[variable, 'Rich?']), encoded_col, df_1['Rich?']], axis=1)

Data Scaling

Normalization: involves to the rescaling of the features to a range of $[0, 1]$

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

where:

- x_{max}: the largest value of column x
- x_{min}: the smallest value of column x

Standardization: centers the columns at the mean 0 with the standard deviation 1

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

where:

- μ_x: the mean of column x
- σ_x: the standard deviation of column x

Normalization

Syntax

sklearn.preprocessing.MinMaxScaler()

Examples

```
>> from sklearn.preprocessing import MinMaxScaler
   norm_scaler = MinMaxScaler()
>> df_1[['Age']] = norm_scaler.fit_transform(df_1[['Age']])
   df_1[['Age']]
```

Standardization

Syntax

sklearn.preprocessing.StandardScaler()

Examples

```
>> from sklearn.preprocessing import StandardScaler
   std_scaler = StandardScaler()
>> df_1[['Height']] = std_scaler.fit_transform(df_1[['Height']])
   df_1[['Height']]
```

Data Splitting (Train-Test Split)

Syntax

sklearn.model_selection.train_test_split$(X, y, test\_size, random\_state)$

- X: independent variables
- y: target variable

Examples

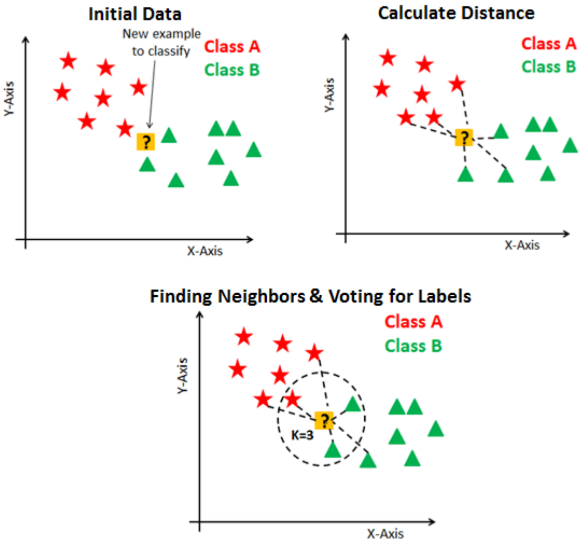
```
>> from sklearn.model_selection import train_test_split
>> X = df_1.drop(columns = ['Rich?', 'ID'])
   y = df_1['Rich?']
>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

Exercises

DataPreprocessing_exercise.pdf

Overview

Recall


Initial Data
New example to classify
Class A
Class B
?
Y-Axis
X-Axis
Calculate Distance
Class A
Class B
?
Y-Axis
X-Axis
Finding Neighbors & Voting for Labels
Class A
Class B
?
K=3
Y-Axis
X-Axis

Load 'Iris' dataset and Train-test Split (7:3)

Examples

```
>> iris_dataset = pd.read_csv('content/iris_dataset.csv')
   iris_dataset
>> X = iris_dataset.drop(columns=['species'])
   y = iris_dataset['species']
>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

How to implement KNN?

Syntax (initialize)

sklearn.neighbors.KNeighborsClassifier($n\_neighbors$, p)

where:

- n_neighbors: the number of neighbors (K)
- p: power parameter for the Minkowski metric.
  - $p = 1$: Manhattan distance
  - $p = 2$: Euclidean distance
  - $p > 2$: Minkowski distance

Examples

```
>> from sklearn.neighbors import KNeighborsClassifier
>> clf = KNeighborsClassifier(n_neighbors = 3, p = 2)
```

How to implement? (cont.)

Syntax (fit)

sklearn.neighbors.KNeighborsClassifier().fit(X, y)

Examples

$>>$ clf.fit(X_train, y_train)

Syntax (predict)

sklearn.neighbors.KNeighborsClassifier().predict(X)

Examples

$>>$ y_pred $=$ clf.predict(X_test)

Overview

Performance Metrics

Classification

- Accuracy
- Confusion matrix
- Precision and Recall
- F1 score

Regression

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-Squared

Syntax (import)

```
from sklearn.metrics import ...
```

Examples

```
>> from sklearn.metrics import accuracy_score
>> accuracy = accuracy_score(y_test, y_pred)
   accuracy
```

Exercise

KNN_exercise.pdf