012-02-ML_basics_solution

April 18, 2024

1 012-02 - ML Basics - Solution Notebook

- Written by Alexandre Gazagnes
- Last update: 2024-02-01

1.1 About

Context :

You will need to know some very core concepts about 'tabular' Machine learning before talking about NLP.

Data :

You can find the dataset here.

1.2 Preliminaries

1.2.1 System

These commands will display the system information:

Uncomment theses lines if needed.

[ ]:
```
# pwd
```

[ ]:
```
# cd ..
```

[ ]:
```
# ls
```

These commands will install the required packages:

Please note that if you are using google colab, all you need is already installed

[ ]:
```
# !pip install pandas matplotlib seaborn plotly scikit-learn
```

or copy the file requirements.txt and :

[ ]:
```
#! pip install -r requirements.txt
```

Try to use a virtual enviromement with venv, virtualenv or pipenv

```
[ ]: #! python3 -m venv .venv # create the .venv folder
#! source .venv/bin/activate # activate the virtual env
#! pip install -r requirements.txt # install the requirements.txt
```

Please uncomment and run the following lines if needed (to download the dataset)

```
[ ]: # !wget https://gist.githubusercontent.com/AlexandreGazagnes/
  ↪cb63600b7a6a71b5f7b714bfe7540137/raw/
  ↪cc8563822e19b196aebe0a52c9f74598888d9c29/iris_exam.csv
```

1.2.2 Import

Import data libraries:

```
[ ]: import pandas as pd
import numpy as np
```

Import Graphical libraries:

```
[ ]: import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

Import Machine Learning libraries:

```
[ ]: # must to have (mandarory)
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import *
from sklearn.model_selection import *
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.impute import *
from sklearn.preprocessing import *
from sklearn.ensemble import *
from sklearn.neighbors import *
from sklearn.dummy import *
```

Ignore the warnings :

```
[ ]: import warnings

warnings.filterwarnings("ignore")
```

If needed we can use a TEST_MODE to run the notebook to have a very fast execution :

```
[ ]: TEST_MODE = True
```

```
[ ]: CV = 10  # number of folds for the  cross val
N_JOBS = 7  # number of cpu to use for computations
FRAC = 1.0  # we keep 100% of the dataframe
DISPLAY = True  # display complex viz
```

```
TEST_SIZE = 0.25  # Train vs Test %

if TEST_MODE:
    CV = 2
    N_JOBS = -1
    FRAC = 0.1
    DISPLAY = False
    TEST_SIZE = 0.5
```

1.2.3 Get the data

1st option : Download the dataset from the web

[ ]:
```
url = "https://gist.githubusercontent.com/AlexandreGazagnes/
 ↪cb63600b7a6a71b5f7b714bfe7540137/raw/
 ↪cc8563822e19b196aebe0a52c9f74598888d9c29/iris_exam.csv"
df = pd.read_csv(url)
df.head()
```

If needed let's take just a specific % of the dataframe :

[ ]:
```
if TEST_MODE:
    df = df.sample(frac=FRAC)
```

2nd Option : Read data from a file

[ ]:
```
# or

# fn = "my/super/file.csv"
# df = pd.read_csv(fn)
# df.head()
```

1.3 First Tour

Print out the first rows of the dataset

[ ]:
```
df.head()
```

Print out the last rows of the dataset

[ ]:
```
df.tail()
```

Print out 10 random lines of the data set

[ ]:
```
df.sample(10)
```

Global information about the dataframe

[ ]:
```
df.info()
```

List of data types for each column

[ ]:
```
df.dtypes
```

The shape of our dataframe

[ ]:
```
df.shape
```

Compute all missing values for each column

[ ]:
```
df.isna().sum()
```

Do we have some missing values ? If so how many, and what should we do ?

Compute mean, std, median, min, max etc

[ ]:
```
df.describe().round(2)
```

Compute the number of unique values for each column

[ ]:
```
df.nunique()
```

Keep in mind the shape of our data set

[ ]:
```
df.shape
```

Let's plot the correlation matrix

[ ]:
```
def make_corr_heatmap(df):
    corr = df.select_dtypes(include="number").corr()
    mask = np.triu(corr)
    sns.heatmap(
        corr, annot=True, cmap="coolwarm", fmt=".2f", vmin=-1, vmax=1, mask=mask
    )
```

[ ]:
```
if DISPLAY:
    make_corr_heatmap(df)
```

What is your conclusion?

Let's display the pair plot visualisation for numerical features

[ ]:
```
if DISPLAY:
    sns.pairplot(df, corner=True)
```

Without any statistical analysis, what can you say about the data? Regarding the pair plot, how many clusters do we have ?

Let's do the same but with the hue parameter (the true value of each flower's species)

[ ]:
```
if DISPLAY:
    sns.pairplot(df, hue="Species", corner=True)
```

1.4 Cleaning and Preparation

1.4.1 Cleaning

Keep in mind the number of missing values

[ ]:
```
df.isna().sum()
```

We need to fill the missing values for the column "sepal lenght" with the median value.

Filling missing values with the mean could be another option, but you know why it is not the best one ;)

So, Let's compute the median value :

[ ]:
```
_median = df.SepalLengthCm.median()
_median
```

We can then fill the missing values with the median value

[ ]:
```
df["SepalLengthCm"] = df["SepalLengthCm"].fillna(_median)
```

Let's check if the problem is solved

[ ]:
```
df.isna().sum()
```

Keep in mind our data numerical description :

[ ]:
```
df.describe().round(2)
```

Do you think we have outliers in our data frame ? If so, what is the column concerned, and what value seems to be an outlier ?

Let's select the specific line

[ ]:
```
df.loc[df.PetalWidthCm > 10, :]
```

Compute the median of the column "petal width"

[ ]:
```
_median = df.PetalWidthCm.median()
_median
```

Let's change the outlier value

[ ]:
```
df.loc[df.PetalWidthCm > 10, "PetalWidthCm"] = _median
```

The problem is solved, let's check it

[ ]:
```
df.describe().round(2)
```

Keep in mind our data types

[ ]:
```
df.dtypes
```

Keep in mind our number of unique values per column:

[ ]:
```
df.nunique()
```

Do you think we have useless columns in our data frame ? What are theses columns and why ?

Even if "Species" is a special column, we need to keep this one.

Please drop the useless columns

[ ]:
```
cols = ["Date", "Id"]
df = df.drop(columns=cols, errors="ignore")
df
```

Good, now we need to create our X matrix.

1.4.2 X and y

We need to extract X (our data) from y (our target) :

[ ]:
```
X = df.drop(columns="Species")
X
```

[ ]:
```
y = df.Species
y
```

1.5 Modelisation

1.5.1 First Try

We need an estimator :

[ ]:
```
estimator = LogisticRegression()
estimator
```

Let's fit this estimator :

[ ]:
```
estimator.fit(X, y)
estimator
```

Let's predict :

[ ]:
```
y_pred = estimator.predict(X)
y_pred[:10]
```

Our score :

[ ]:
```
estimator.score(X, y)
```

The same :

[ ]:
```
accuracy_score(y_true=y, y_pred=y_pred)
```

Using the confusion matrix :

[ ]:
```
y = pd.Series(y, name="y_true")
y_pred = pd.Series(y_pred, name="y_pred")
pd.DataFrame(confusion_matrix(y, y_pred))
```

More readable output :

[ ]:
```
pd.crosstab(y, y_pred)
```

1.5.2 Using Train and Test values

Creating train and test values :

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=TEST_SIZE,
    shuffle=True,
    random_state=42,
)

# other possible values : 0.25, 0.2
```

X_train :

[ ]:
```
X_train.shape
```

X_test :

[ ]:
```
X_test.shape
```

Estimator :

[ ]:
```
estimator = LogisticRegression()
```

Fit :

[ ]:
```
estimator.fit(X_train, y_train)
```

Train score :

[ ]:
```
estimator.score(X_train, y_train)
```

Test score :

[ ]:
```
estimator.score(X_test, y_test)
```

1.5.3 Using a Grid Search

About the grid Search

[ ]:
```
grid = GridSearchCV(
    LogisticRegression(),
```

```
    param_grid={},
    cv=CV,
    return_train_score=True,
    refit=True,
    n_jobs=N_JOBS,
    verbose=2,
)
```

Fit :

[ ]:
```
grid.fit(X_train, y_train)
```

Our results :

[ ]:
```
grid.cv_results_
```

In a dataframe :

[ ]:
```
pd.DataFrame(grid.cv_results_)
```

Lets's create a function :

[ ]:
```
def resultize(grid):

    res = grid.cv_results_
    res = pd.DataFrame(res)

    cols = [i for i in res.columns if "split" not in i]
    res = res.loc[:, cols]

    res = res.drop(columns=["mean_score_time", "std_score_time"])

    return res.round(2).sort_values("mean_test_score", ascending=False)
```

Resultize :

[ ]:
```
resultize(grid)
```

1.5.4 Using a pipeline

Our first pipeline :

[ ]:
```
pipe = Pipeline(
    [
        ("imputer", KNNImputer()),
        ("scaler", StandardScaler()),
        ("estimator", LogisticRegression()),
    ]
)
```

[ ]:
```
pipe
```

Using the pipeline :

[ ]:
```
grid = GridSearchCV(
    pipe,
    param_grid={},
    cv=CV,
    return_train_score=True,
    refit=True,
    n_jobs=N_JOBS,
    verbose=2,
)
```

Fit :

[ ]:
```
grid.fit(X_train, y_train)
```

Resultize :

[ ]:
```
resultize(grid)
```

1.5.5 Using a Param Grid

Keep in mind :

[ ]:
```
pipe
```

[ ]:
```
grid
```

Writing a beautiful param grid :

[ ]:
```
param_grid = {
    "imputer": [
        "passthrough",
        KNNImputer(n_neighbors=3),
        KNNImputer(n_neighbors=5),
        SimpleImputer(strategy="median"),
    ],
    "scaler": [
        # "passthrough",
        StandardScaler(),
        Normalizer(),
        QuantileTransformer(n_quantiles=10),
    ],
    "estimator": [
        DummyClassifier(),
        LogisticRegression(),
        KNeighborsClassifier(n_neighbors=5),
        RandomForestClassifier(),
```

```
    ],
}
```

[ ]:
```
param_grid
```

Using the param grid :

[ ]:
```
grid = GridSearchCV(
    pipe,
    param_grid=param_grid,
    cv=CV,
    return_train_score=True,
    refit=True,
    n_jobs=N_JOBS,
    verbose=1,
)
```

Fit :

[ ]:
```
grid.fit(X_train, y_train)
```

Resultize !

[ ]:
```
resultize(grid).head(10)
```