An example on data analysis using Scikit-learn package

Van Le

Department of Mathematics
The University of Tennessee, Knoxville

May 1, 2024

Main goal

- Learn to use Scikit-Learn.
- Use some classification models to predict whether a patient is likely to get a stroke based on inputs such as gender, age, various diseases and smoking status.

Explore and visualize the data to gain insights

Downloading data

```python
stroke_data = pd.read_csv(Path("healthcare-dataset-stroke-data.csv"))
stroke_data.head(10)
```

Figure: Stroke prediction dataset[1]

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
6	53882	Male	74.0	1	1	Yes	Private	Rural	70.09	27.4	never smoked	1
7	10434	Female	69.0	0	0	No	Private	Urban	94.39	22.8	never smoked	1
8	27419	Female	59.0	0	0	Yes	Private	Rural	76.15	NaN	Unknown	1
9	60491	Female	78.0	0	0	Yes	Private	Urban	58.57	24.2	Unknown	1

[1] Source: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

```python
stroke_data.info()
```

Figure: Stroke prediction dataset's information

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5110 non-null   int64
 1   gender             5110 non-null   object
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64
 4   heart_disease      5110 non-null   int64
 5   ever_married       5110 non-null   object
 6   work_type          5110 non-null   object
 7   Residence_type     5110 non-null   object
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object
 11  stroke             5110 non-null   int64
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

A quick look at categorical attributes

```
stroke_data["NAME_OF_COLS"].value_counts(ascending = True)

>>>gender
Other          1
Male        2115
Female      2994
Name: count, dtype: int64


>>>work_type
Never_worked        22
Govt_job           657
children           687
Self-employed      819
Private           2925
Name: count, dtype: int64
```

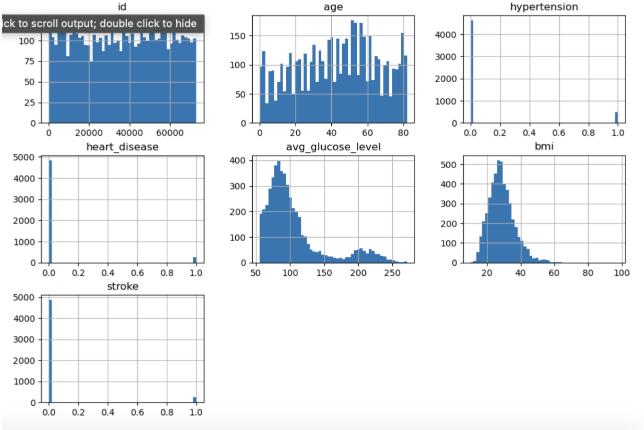
A quick look at categorical attributes

```
>>>ever_married
No      1757
Yes     3353
Name: count, dtype: int64

>>>Residence_type
Rural     2514
Urban     2596
Name: count, dtype: int64

>>>smoking_status
smokes                  789
formerly smoked         885
Unknown                1544
never smoked           1892
Name: count, dtype: int64


```

A quick look at numerical attributes

```
stroke_data.hist(bins=50, figsize=(12, 8))
```

Figure: Histrogram of numerical attributes

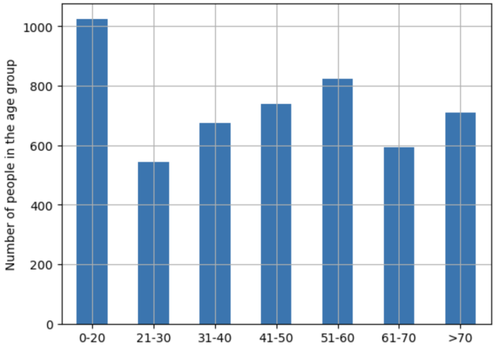


Create test and training sets

Create training set and test set

We can split the data randomly or based on a certain category. I suspect that each age group has a certain risk, and hence I split the data based on age groups.

```
stroke_data["age_groups"] = pd.cut(stroke_data["age"],
bins=[0., 20, 30, 40, 50, 60, 70, np.inf],
labels=["0-20", "21-30", "31-40", "41-50", "51-60", "61-70", ">70"])
stroke_data["age_groups"].value_counts().sort_index().plot.bar(rot=0, grid=True)

```



Create test and training sets

```
from sklearn.model_selection import train_test_split
x_train_set, x_test, y_train, y_test = train_test_split(
stroke_data, stroke_data["stroke"], test_size=0.20, stratify=
stroke_data['age_groups'], random_state=42)
x_train = x_train_set.copy()

```

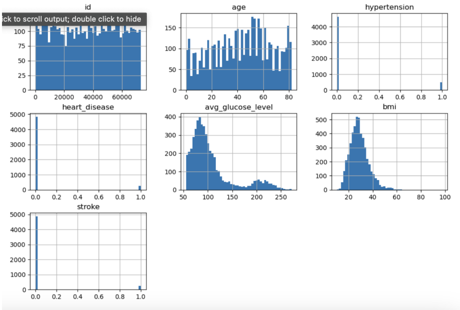
Let's see if this worked as expected. Let's look at the category proportions in the test set:

```
x_train['age_groups'].value_counts() / len(x_train)
>>>age_groups
0-20      0.200587
51-60     0.161204
41-50     0.144569
>70       0.138943
31-40     0.131849
61-70     0.116194
21-30     0.106654
Name: count, dtype: float64

```

Preparing data

Preparing data

Figure: Histrogram of numerical attributes



After looking at the data, we decide to separate them into the 3 groups

```
num_attribs = ["age"]
cat_attribs = ["gender", "ever_married", "work_type", "
    Residence_type",
  "smoking_status", "hypertension", "heart_disease"]
log_attribs = ["avg_glucose_level", "bmi"]
```

Missing data

We have two options to deal with missing data:

- Get rid of the whole attribute.
- Set the missing values to some value (zero, the mean, the median, etc.). This is called imputation.

We suspect that "bmi" may affect the risk of getting strokes. Thus, we decide to imputate the data. We use the function SimpleImputer as below.

```
from sklearn.impute import SimpleImputer
```

There are several ways to impute the data. For example,

```
SimpleImputer(strategy='median')
SimpleImputer(strategy="most_frequent")
```

Feature scaling

One of the most important transformations is *scaling*. The models do not perform well then the input have very different scales.

For example, the age attribute is between 0 and 100 while avg_glucose_level is above 200. In Scikit-Learn, there are two basic transformers called MinMaxScaler and StandardScaler.

- MinMaxScaler: This is performed by subtracting the min value and dividing by the difference between the min and the max.
- StandardScaler: It subtracts the mean value and divides the result by the standard deviation.

Encoding

In categorical attribute, each text represents a category. We can choose a function to use from Scikit-Learn package. There are many encoders supported in the package such as OrdinalEncoder, OneHotEncoder.

- OrdinalEncoder: Label the groups as $0, 1, 2, 3...$
- OneHotEncoder: Create a binary attribute, 1 for a category and 0 otherwise.

Transformation pipelines

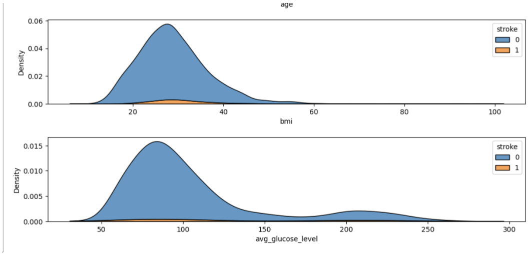
Numerical attributes

Now we are ready to make a pipeline for numerical attributes.

```
from sklearn.pipeline import make_pipeline
num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
    StandardScaler())
```

Long-tail attributes

Figure: Long-tail attributes



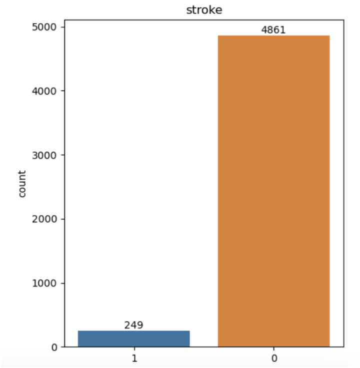

```python
from sklearn.preprocessing import FunctionTransformer
log_pipeline = make_pipeline(
SimpleImputer(strategy="median"),
FunctionTransformer(np.log, feature_names_out="one-to-one"),
StandardScaler())

```

Categorical attributes

A pipeline for categorical attributes.

```
cat_pipeline = make_pipeline(SimpleImputer(strategy="most_frequent"
    ), OneHotEncoder(handle_unknown="ignore"))
```

Imbalanced data



Notice that we are
dealing with imbalanced datasets. The models may have poor performance on the minority class (that had a stroke). We can use Synthetic Minority Over-sampling Technique (SMOTE) in Imbalanced-learn package.

- How does SMOTE work?

Selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

- Pros: Enhance generalization, reduce over-fitting
- Cons: Samples are created without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap for the classes.

```python
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=2)
x_train_smote, y_train_smote = sm.fit_resample(x_train, y_train.
    ravel())
```

Classification models

Here are some models included in Scikit-learn

- LogisticRegression
- DecisionTreeRegressor
- RandomForestClassifier
- SVC
- SVM with Poly Kernel
- SVM with Gaussian RBF Kernel

Accuracy measure

- Accuracy score: $\frac{\text{Correct Predictions}}{\text{Total Predictions}}$
- Cross validation: divide the data into k-subsets, the model is trained and evaluated k-times. The result is the average of k validations.

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

from sklearn.model_selection import cross_val_score
cross_val_score(estimator = NAME_OF_MODELS , X = x_train_smote , y =
    y_train_smote , cv = 3)

```

Compare accuracy of models

Data is cleaned in the pipelines:

	Accuracy score	K-fold cross-validation
LogisticRegression	83.37%	82.41%
RandomForestClassifier	91.59%	93.82%
SVC	84.76%	87.91%
SVM with Poly Kernel	85.03%	88.41%
SVM with Gaussian RBF Kernel	86.59%	90.95%

Data is cleaned before and in the pipelines:

	Accuracy score	K-fold cross-validation
LogisticRegression	95.79%	96.20%
RandomForestClassifier	95.11%	96.92%
SVC	95.69%	94.99%
SVM with Poly Kernel	95.69%	96.50%
SVM with Gaussian RBF Kernel	91.88%	94.59%

Thank you for listening!