02/06/2020

Supervised Learning with scikit-learn

[1] Classification

Supervised learning

* what is machine learning?
  - The art and science of:
    - Giving computers the ability to learn to make decisions from data.
    - without being explicitly programmed.
  - Examples:
    - learning to predict whether an email is spam or not.
    - clustering wikipedia entries into different categories.
  - Supervised learning: Uses labeled data.
  - Unsupervised learning: Uses unlabelled data.

03/06/2020

Unsupervised learning

- Uncovering hidden patterns from unlabelled data.
- Example.
  - Grouping customers into distinct categories (clustering).

Reinforcement learning

- Software agents interact with an environment
  - Learn how to optimize their behavior
  - Given a system of rewards & punishments.
  - Draws inspiration from behavioral psychology

- Applications
  - Economics
  - Genetics
  - Game Playing

- AlphaGo: First computer to defeat the world champion in Go.

Supervised learning

- Predictor variables / features & a target-variable.
- Aim: Predict the target variable, given the predictor variables.
  - classification: Target variable consists of categories
  - Regression: Target variable is continuous.

Naming Conventions

- Features = predictor variables = independent variables.
- Target variable = dependent variable = response variable.

The Iris Dataset

Features:
- Petal length
- Petal width
- Sepal length
- Sepal width

Target variable: Species
- versicolor
- Virginica
- Setosa

→ The iris dataset in scikit-learn

```
from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')

iris = datasets.load_iris()
type(iris)

iris.keys()

# EDA

X = iris.data
y = iris.target
df = pd.DataFrame(X, column = iris.
                              feature_names)
print(df.head())

# Visual EDA

_ = pd.plotting.scatter_matrix(df, c=y,
                    figsize=[8, 15], s=150,
                    marker='D')
```

* In this iris dataset we have used scatter plot. But if we have a dataset in the form of "Yes" or "No" then we can use seaborn's countplot().

→
```
plt.figure()
sns.countplot( x='education', hue='party',
    data = df, palette='RdBu')
plt.xticks ([0,1], ['No', 'Yes'])
plt.show()
```

* K-Nearest Neighbors

- Basic idea: Predict the label of a data point by
  - Looking at the 'K' closest labeled data points.
  - Taking a majority vote.

* What is classification?

- A supervised learning approach.
- Categorizing some unknown items into a discrete set of categories or "classes".
- These target attribute is a categorical variable.

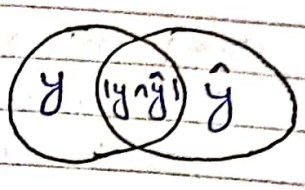
- classification algorithms in machine learning:
  1. Decision Trees
  2. Naïve Bayes
  3. Linear Discriminant Analysis.
  4. K-Nearest Neighbor
  5. Logistic Regression.
  6. Neural Networks.
  7. Support Vector Machines (SVM).

The K-Nearest Neighbor algorithm

1. Pick a value for K.
2. Calculate the distance of unknown case from all cases.
3. Select the K-observations in the training data that are "nearest" to the unknown data point.
4. Predict the response of the unknown data point using the most popular response value from the K-nearest neighbors.

2° Evaluation matrix in classification

1) Jaccard index



y : Actual labels
$\hat{y}$: Predicted labels

$$J(y,\hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} = \frac{|y \cap \hat{y}|}{|y|+|\hat{y}|-|y \cap \hat{y}|}$$

eg:- y : [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

$\hat{y}$: [1, 1, 0, 0, 0, 1, 1, 1, 1, 1]

8 correctly predicted

$$J(y,\hat{y}) = \frac{8}{10+10-8} = 0.66$$

2) Confusion matrix

TP	FN
FP	TN

- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F1-Score = $\frac{2 \times (\text{precision} \times \text{recall})}{\text{precision} + \text{recall}}$

3) Log loss

- Performance of a classifier whether the predicted o/p is a probability value b/w 0 & 1.
- Predicting log loss for each row:

$$(y \times \log(\hat{y}) + (1-y) \times \log(1-\hat{y}))$$

- Avg. log loss

$$\text{LogLoss} = -\frac{1}{n} \sum (y \times \log(\hat{y}) + (1-y) \times \log(1-\hat{y}))$$

4) Scikit-learn fit and predict

- All machine learning models implemented as Python classes
  - They implement the algorithm for learning and predicting.
  - Store the information learned from the data.
- Training a model on the data = "fitting" a model to the data
  - .fit() method.

- To predict the labels of new data:
  - .predict() method

:- Using scikit-learn to fit a classifier

→ from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(iris['data'], iris['target'])

→ iris['data'].shape

→ iris['target'].shape

:- Predicting on unlabelled data

→ X_new = np.array([[5.6, 2.8, 3.9, 1.1],
[5.7, 2.6, 3.8, 1.3],
[4.7, 3.2, 1.3, 0.2]])

→ prediction = knn.predict(X_new)

→ X_new.shape

→ print("Prediction: {}".format(prediction))

↳ output

Prediction: [1 1 0]

Measuring model performance

- Could compute accuracy on data used to fit classifier.
- NOT indicative of ability to generalize.
- Split data into training & test set.
- Fit/train the classifier on training set.
- Make predictions on the test set.
- Compare predictions with the known labels.

Train/Test split

→ from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=21, stratify=y)

knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(x_test)
print("Test set predictions: \n {}".format(y_pred))

To check accuracy we use score method.

knn.score (x_test, y_test)

↳ O/P : 0.95555..

04/06/2020

Codes on K-NN

* About the dataset

- Imagine a telecommunication provider has segmented its customer base by service usage patterns, categorizing the customers into 4 groups.
- The example focuses on using demographic data, such as region, ~~stores~~ age and marital to predict usage patterns.
- The target field called "custcat", has 4 possible values that correspond to the 4 customer groups, as follows:
  1- Base service
  2- E-service
  3- Plus-service
  4- Total service

```
# let's load required libraries

import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import matplotlib.ticker as ticker
from sklearn import preprocessing
% matplotlib inline
```

import itertools :

This module is used for implementing number of iteration tools.

from matplotlib.ticker import NullFormatter :
import matplotlib.ticker as ticker :
matplotlib.ticker

This module contains classes to support completely configurable tick locating and formatting.

load data from csv file

```
df = pd.read_csv('teleCust1000t.csv')
df.head()
```

Data visualization & analysis

let's see how many of each class is in our dataset.

```
df['custcat'].value_counts()
```

⇓

281 - Plus service
266 - Basic service
236 - Total service
217 - E-service

Feature set

Let's define feature sets, X:

```
df.columns
```

```
X = df[['region', 'tenure', 'age', 'marital',
        'address', 'income', 'ed', 'employ',
        'retire', 'gender', 'reside']].values
X[0:5]
```

```
y = df['custcat'].values
y[0:5]
```

Normalize Data

→ Data standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases:

```
X = preprocessing.StandardScaler().fit(X).
                    transform(X.astype(float))
X[0:5]
```

Train/Test Split

→ Out of sample Accuracy is the percentage of correct predictions that the model makes on data that the model has not been trained on.

→ Doing a train and test on the same dataset will most likely have low out-of-sample accuracy, due to the likelihood of being over-fit.

→ It is important that our models have a high, out-of-sample accuracy, because the purpose of any model, of course, is to make correct predictions on unknown data.

```
from sklearn.model_selection import
                train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)

# Classification

# K Nearest Neighbors (KNN)

# import library

from sklearn.neighbors import KNeighborsClassifier

# Training
```

→ let's start the algorithm with k=4 for now.

```
k=4
# Train model
neigh = KNeighborsClassifier(n_neighbors = k).
                fit(X_train, y_train)
neigh
```

Predicting

→ We can use the model to predict the test set:

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Accuracy Evaluation

→ In multilabel classification, accuracy classification score is a function that computes subset accuracy.

```
from sklearn import metrics
print("Train set Accuracy:", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy:", metrics.accuracy_score(y_test, yhat))
```

what about other K?

→ K in KNN is the number of nearest neighbors to examine. It is supposed to be specified by user. So, how can we ~~use~~ choose right value for K?

↳ The general solution is to reserve a part of your data for testing the accuracy of the model.

↳ Then choose K=1, use the training part for modelling, and calculate the accuracy of prediction using all samples in your test set.

↳ Repeat this process increasing the K, and see which K is the best for your model.

→ We can calculate the accuracy of KNN for different Ks.

```
Ks =10
mean_acc= np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
confusionMx = [];
for n in range(1,Ks):
    # Train model & predict
    neigh = KNeighborsClassifier(n_neighbors=n).
            fit(X_train, y_train)
    yhat = neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score
            (y_test, yhat)

    std_acc[n-1] = np.std(yhat == y_test)/
            np.sqrt(yhat.shape[0])

mean_acc
```

Plot model accuracy for different number of Neighbors

```
plt.plot (range(1,Ks), mean_acc, 'g')
plt.fill_between (range(1, Ks), mean_acc-1 *
        std_acc, mean_acc +1 * std_acc, alpha=0.10)
plt.legend('Accuracy', '+/- 3 x std'))
plt.ylabel ('Accuracy')
plt.xlabel ('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```

```
print ("The best accuracy was with",
        mean_acc.max(), "with k=",
        mean_acc.argmax() + 1)
```

The best accuracy was with 0.34 with k=9