Scikit-Learn Tutorial

Machine Learning

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

In [ ]:
```
# import dataset already loaded in scikit-learn
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

In [ ]:
```
print(housing['DESCR'])
```

In [ ]:
```
# store data into dataframe
df=pd.DataFrame(housing.data)
```

In [ ]:
```
# set features as columns on the dataframe
df.columns = housing.feature_names
```

In [ ]:
```
# view first 5 observation
df.head()
```

In [ ]:
```
# append price - target, as a new column to the dataset
df['Price'] = housing.target
df.info()
```

In [ ]:
```
X=housing.data
Y=housing.target
```

linear regression

In [ ]:
```
from sklearn.linear_model import LinearRegression
lineReg = LinearRegression()
```

In [ ]:
```
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_
```

In [ ]:
```
# fit the training sets into the model
lineReg.fit(X_train, Y_train)
```

In [ ]:
```
# print the coefficient
print('The coefficient is %d' %len(lineReg.coef_))
```

In [ ]:
```
# calculate variance
from sklearn.metrics import r2_score
var_score=lineReg.score(X_test, Y_test)
print('Variance score is ',var_score)
test_r2 = r2_score(Y_test, lineReg.predict(X_test))
print('r2 score is: ',test_r2)
```

In this case, both scores are approximately 0.59, indicating that the model explains about 59% of the variance in the target variable. This means that the model captures 59% of the variability in the observed data, which is a moderately good performance.

Learning Models

supervised learning models

- The model is trained using labeled data, where each input comes with a corresponding output.
- To learn a mapping from inputs to outputs to make predictions on new data.

unsupervised learning models

- The model is trained using unlabeled data, which means there are no output labels.
- To find patterns, groupings, or structures within the data.

supervised learning models: Logistic Regression

- logistic Regression is primarily used for binary classification problems, where the output is either 0 or 1.
- Logistic Regression predicts the probability that a given data point belongs to a certain class.

In [ ]:
```
# import sklearn load dataset
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

In [ ]:
```
print(iris_dataset.DESCR)
```

In [ ]:
```
X=iris_dataset.data
Y=iris_dataset.target
```

KNN model

- KNN, or k-nearest neighbors, is a machine learning algorithm used for classification tasks
- The K-NN algorithm works by finding the K nearest neighbors to a given data point based on a distance
- predicted class for a new data point will be the class of its closest neighbor.

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [ ]:
```
# fit data into knn model(estimator)
knn.fit(X, Y)
```

In [ ]:
```
# create object with new values for prediction
X_new = [[3,5,4,1],[5,3,4,2]]
knn.predict(X_new)
```

In [ ]:
```
# Use logistic regression estimator
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression()
```

In [ ]:
```
logReg.fit(X, Y)
```

In [ ]:
```
# predict the outcome using Logistic Regression estimator
logReg.predict(X_new)
```

Unsupervised Learning Models: Clustering

KMeans Clustering

In [ ]:
```
# import KMeans class from sklearn.cluster
from sklearn.cluster import KMeans
```

In [ ]:
```
# import make_blobs dataset from sklearn.cluster
from sklearn.datasets import make_blobs
```

In [ ]:
```
# define number of feature as 5
X,y = make_blobs(n_samples=300, n_features=5, random_state=None)

predict_y = KMeans(n_clusters=3, random_state=20).fit_predict(X)

# print the estimator prediction
predict_y
```

Unsupervised Learning Models : Dimensionality Reduction

Helps cut down dimentions without losing any data from dataset

Techniques used for dimensionality reduction :

- Drop data columns with missing values
- Drop data columns with low variance
- Drop data columns with high corelations
- Apply statistical functions - PCA(Principal Component Analysis)

Principal component analysis (PCA)

PCA is a tool for simplifying data by reducing the number of features while keeping the most important information.

In [ ]:
```
# import required library PCA
from sklearn.decomposition import PCA
from sklearn.datasets import make_blobs
```

In [ ]:
```
# Generate the dataset with 10 features (dimension)
X,y = make_blobs(n_samples=20, n_features=10, random_state=20)
X.shape
```

In [ ]:
```
# Define the PCA estimator with number of reduced components
pca = PCA(n_components=3)
```

In [ ]:
```
# Fit the data into the PCA estimator
pca.fit(X)
```

pac.explained_variance_ratio tells you how much information (variance) each principal component captures from the original dataset.

In [ ]:
```
print(pca.explained_variance_ratio_)
```

In [ ]:
```
# Transform the fitted data using transform method
pca_reduced= pca.transform(X)
pca_reduced.shape
```

Pipeline

A Pipeline is a tool to streamline the process of building and evaluating machine learning models. It allows you to chain together multiple steps into a single object, making your code cleaner and easier to manage. Each step in the pipeline can include data preprocessing, dimensionality reduction, feature selection, and the final modeling.

In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA
```

chain estimators together

In [ ]:
```
estimator = [('dim_reduction',PCA()), ('linear_model',LinearRegression())]
```

Put the chain of estimators in a pipeline object

In [ ]:
```
pipeline_estimator = Pipeline(estimator)
pipeline_estimator
```

In [ ]:
```
pipeline_estimator.steps
```

Model Persistance

the process of saving a trained machine learning model to disk so that it can be loaded and used later without having to retrain it.

In [ ]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

In [ ]:
```
X = iris_dataset.data
Y = iris_dataset.target
```

In [ ]:
```
# Create object with new values for prediction
X_new = [[3,5,4,1],[5,3,4,2]]
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
```

In [ ]:
```
logreg.fit(X,Y)
```

In [ ]:
```
logreg.predict(X_new)
```

persistance

In [ ]:
```
# Use joblib.dump to persist the model to a file
import joblib
joblib.dump(logreg, 'regresfilename.pkl')
```

In [ ]:
```
# Create new estimator from the saved model
new_logreg_estimator = joblib.load('regresfilename.pkl')
```

In [ ]:
```
# Validate and use new estimator to predict
new_logreg_estimator.predict(X_new)
```

In [ ]:
```
```