KNN by Yunsu Han

KNN Algorithm

K-Nearest Neighbors (KNN) algorithm is a supervised learning that counts “K” nearest neighbors near the initial value.

Import Packages

Before we run the code, we must import some necessary packages. Those packages are NumPy, pandas (do simple tasks such as creating arrays, sorting tables out, etc.), matplotlib (draws plots such as scatterplots), and KNN from sklearn.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Load Dataset

We need some datasets to perform this algorithm! I brought some data from a website called iris.

[2]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv(url, names=names)
```

Let’s take a slight peek at the data.

[3]:
```
dataset.head()
```

[3]:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Preprocessing

The next step is to split our dataset into its attributes and labels. The X variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.

[5]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
```

Splitting Train and Test Dataset

In order to avoid overfitting, we need to find a good ratio of train dataset: test dataset. This way, the algorithm is tested on unseen data, and so on.

[6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

Scaling

Before making predictions, it is always good to scale the dataset so that we can evaluate every single one of them.

[7]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Training and Predictions

The next step is to train the model and predict in which class the data is included. We set the k value as 5 and fit the model.

[8]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

[8]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[10]:
```
y_pred = classifier.predict(X_test)
```

Evaluating the Algorithm

The final step is to evaluate the accuracy of the model. We use the confusion matrix in order to prevent one-sided predictions. This confusion matrix shows not only accuracy but also recall, support, and f1-score.

[11]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
print(classification_report(y_test, y_pred))
```

```
[[11  0  0]
 [ 0  8  0]
 [ 0  1 10]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.89	1.00	0.94	8
Iris-virginica	1.00	0.91	0.95	11
accuracy			0.97	30
macro avg	0.96	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30