Artificial Neural Networks

K NEAREST NEIGHBORS

Description:

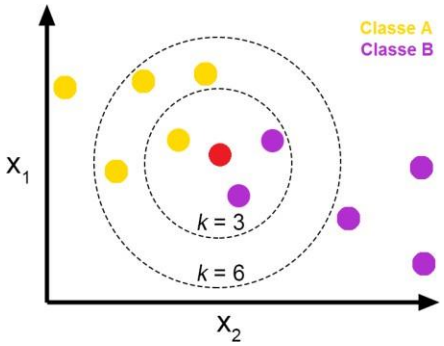
- K-Nearest Neighbors is one of the simplest supervised machine learning algorithms used for classification.
- It is an algorithm that stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a good suite category by using K- NN algorithm.
- It is an algorithm that can be used for Regression as well as for Classification but it is mostly used for Classification problems.
- It is a non-parametric algorithm, which means it does not make any assumptions about underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

How does KNN work?

We usually use Euclidean distance to calculate the nearest neighbor. If we have two points (x, y) and (a, b). The formula for Euclidean distance (d) will be

$$d = \text{sqrt}((x-a)^2+(y-b)^2)$$

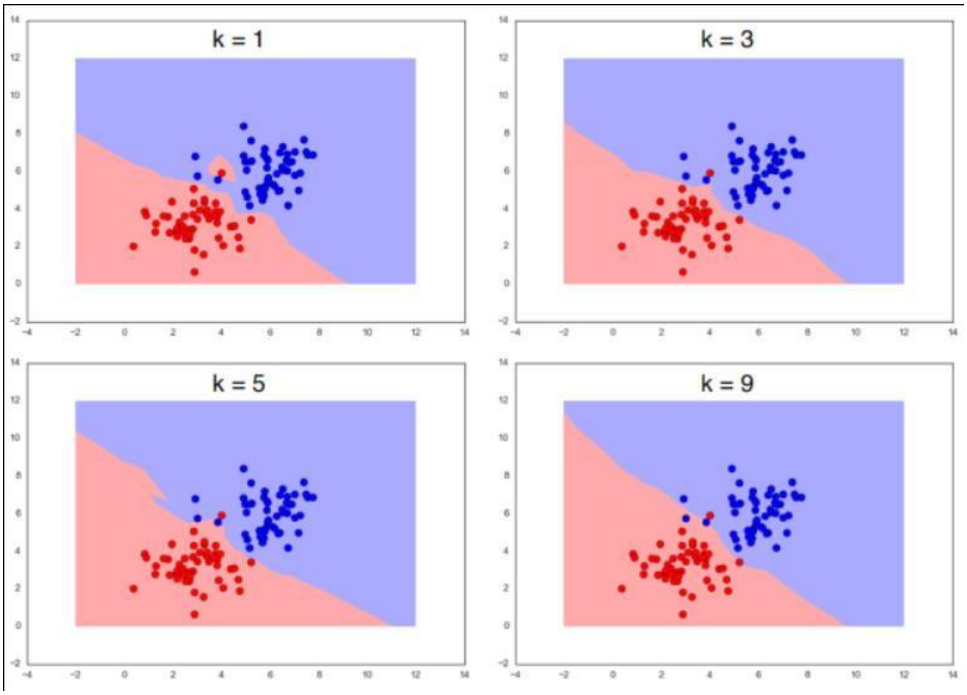
- Load the data
- Initialize K to your chosen number of neighbors
- For each example in the data
- Calculate the distance between the query example and the current example from the data. Add the distance and the index of the example to an ordered collection. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
- Pick the first K entries from the sorted collection
- Get the labels of the selected K entries
- If regression, return the mean of the K labels
- If classification, return the mode of the K labels



Real-life Applications:

K-nearest neighbor has a lot of applications in machine learning because of the nature of the problem which is solved by a k-nearest neighbor. In other words, the problem of the k-nearest neighbor is fundamental and it is used in a lot of solutions.

- The KNN algorithm is one of the most popular algorithms for text categorization or text mining.
- It can also be used in finance for forecasting the stock market, checking on the currency exchange rates, and bank bankruptcy
- It is widely used in the field of medicine as well, to predict whether a patient, can be hospitalized due to a heart attack, and estimate the amount of glucose in the blood of a diabetic person, and so on



Advantages of KNN:

- The algorithm is simple and easy to implement.
- There's no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section)

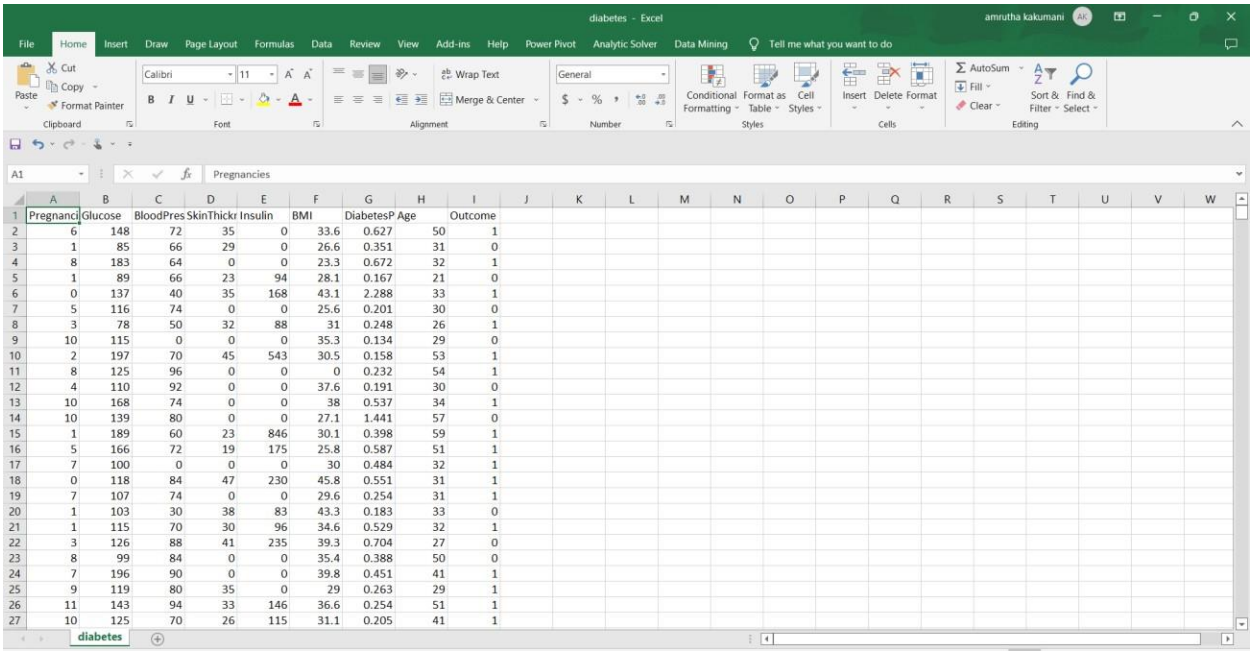
Disadvantages of KNN:

- Accuracy depends on the quality of the data.
- With large data, the prediction stage might be slow.
- Sensitive to the scale of the data and irrelevant features.
- Require high memory – need to store all of the training data.

Aim:

Classification of given Diabetes dataset using the K-nearest neighbor algorithm.

Dataset:



Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1

KNN Algorithm:

A detailed version of the algorithm can be found in pseudo-code:

Nearest-neighbor algorithm

a) A pseudo code for the nearest neighbor algorithm is

ALGORITHM *Nearest-neighbor*$(D[1..n,1..n],s)$

//Input: A $n \times n$ distance matrix $D[1..n,1..n]$ and an index s of the starting city.

//Output: A list Path of the vertices containing the tour is obtained.

for $i \leftarrow 1$ to n do Visited $[i] \leftarrow$ false

Initialize the list Path with s

Visited $[s] \leftarrow$ true

Current $\leftarrow s$

for $i \leftarrow 2$ to n do

  Find the lowest element in row current and unmarked column j containing the element.

  Current $\leftarrow j$

  Visited $[j] \leftarrow$ true

  Add j to the end of list Path

Add s to the end of list Path

return Path

Source: chegg.com

Code:

```
[231] import pandas as pd
      import numpy as np
      from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import StandardScaler
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import confusion_matrix
      from sklearn.metrics import f1_score
      from sklearn.metrics import accuracy_score
      from sklearn import preprocessing
```

```
[232] import warnings
      import numpy as np
      import pandas as pd
      import seaborn as sns
      import matplotlib.pyplot as plt
```

```
dataset = pd.read_csv('/content/diabetes.csv')
```

```
[282] len(dataset)
      dataset.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
[283] normalized = preprocessing.normalize(dataset)
```

```
dataset. dropna ( axis = 0 , inplace = True )
print(dataset)
print(dataset.isnull())
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  \
0          False    False          False          False    False  False
1          False    False          False          False    False  False
2          False    False          False          False    False  False
3          False    False          False          False    False  False
4          False    False          False          False    False  False
```

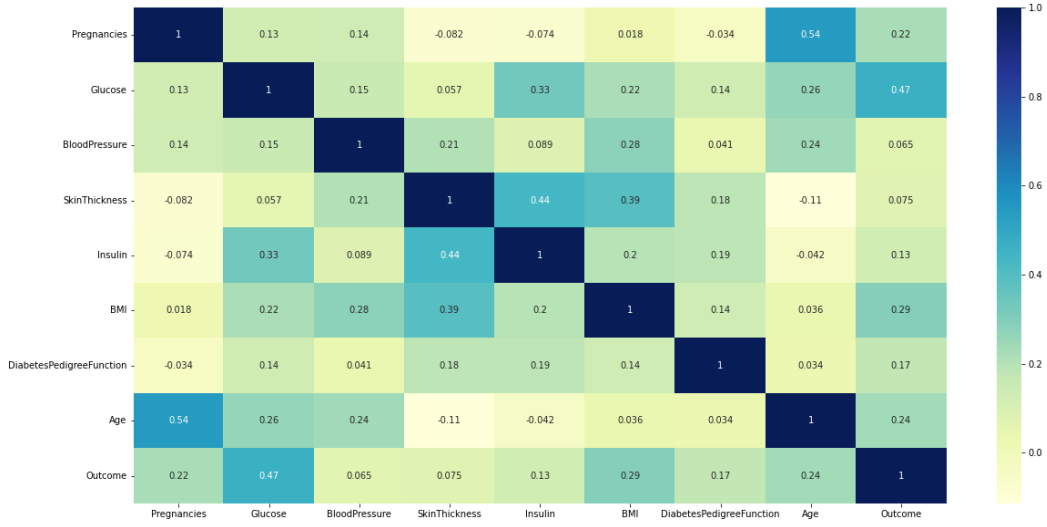
```
767                   0.315   23       0

[768 rows x 9 columns]
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI  \
0          False    False          False          False    False  False
1          False    False          False          False    False  False
2          False    False          False          False    False  False
3          False    False          False          False    False  False
4          False    False          False          False    False  False
..           ...      ...            ...            ...      ...    ...
763        False    False          False          False    False  False
764        False    False          False          False    False  False
765        False    False          False          False    False  False
766        False    False          False          False    False  False
767        False    False          False          False    False  False

     DiabetesPedigreeFunction    Age  Outcome
0                       False  False    False
1                       False  False    False
2                       False  False    False
3                       False  False    False
4                       False  False    False
..                        ...    ...      ...
763                     False  False    False
764                     False  False    False
765                     False  False    False
766                     False  False    False
767                     False  False    False

[768 rows x 9 columns]
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 60.0 KB
```

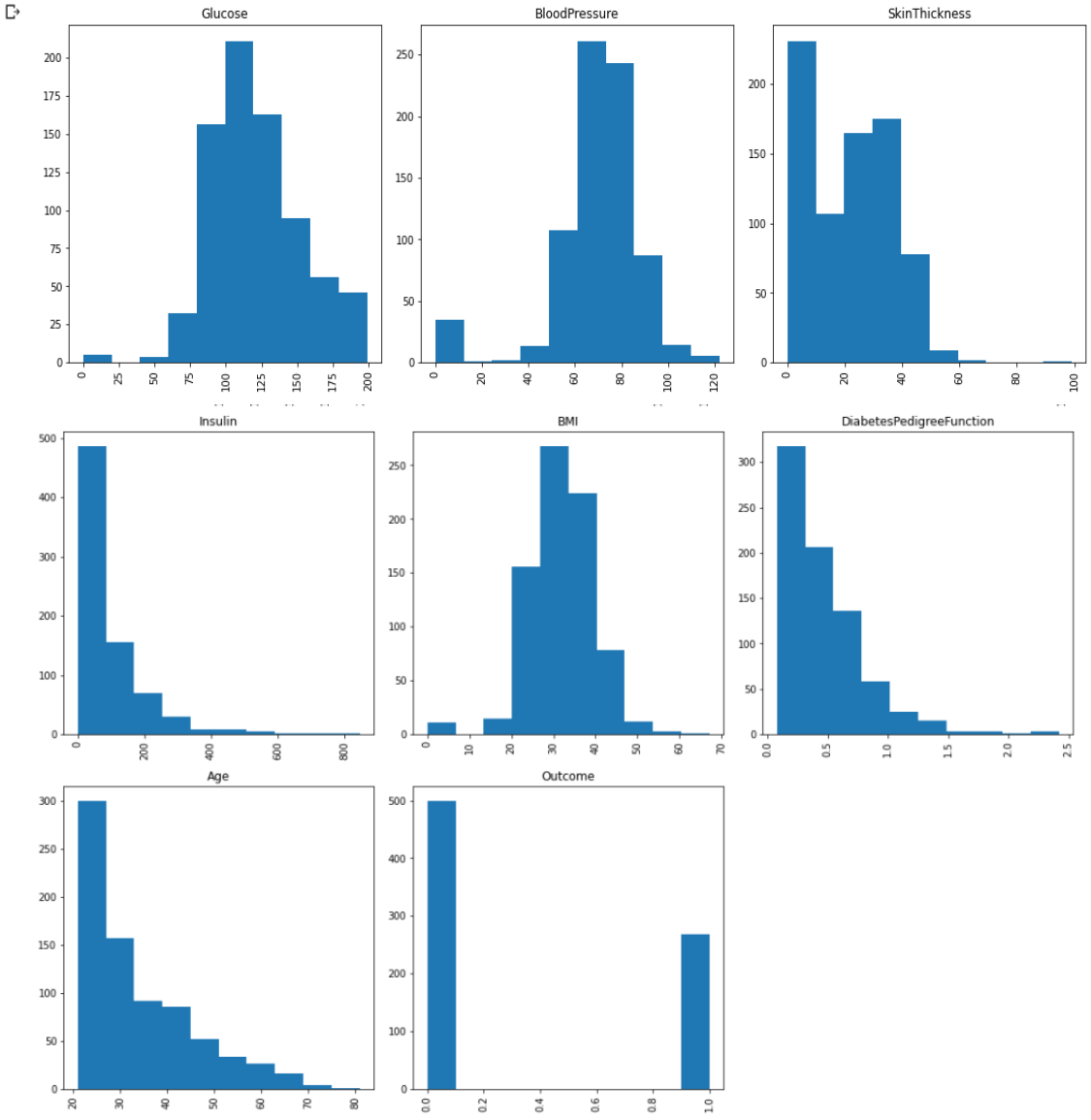
```
plt.figure(figsize=(20,10))
sns.heatmap(dataset.corr(), annot=True, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd479c59250>
```



0s completed at 00:27

```
df_vis=dataset.copy()
cols = list(df_vis.columns)
cols_df=cols[1:]
plt.figure(figsize=(15,40))
for i in range(len(cols_df)):
    plt.subplot(8,3,i+1)
    plt.title(cols_df[i])
    plt.xticks(rotation=90)
    plt.hist(df_vis[cols_df[i]])

plt.tight_layout()
```



```
[288] # Replace zeroes
      zero_not_accepted = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Insulin']

      for column in zero_not_accepted:
          dataset[column] = dataset[column].replace(0, np.NaN)
          mean = int(dataset[column].mean(skipna=True))
          dataset[column] = dataset[column].replace(np.NaN, mean)
```

```
# split dataset
X = dataset.iloc[:, 0:8]
y = dataset.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.2)
```

```
print(len(X_train))
print(len(y_train))
print(len(X_test))
print(len(y_test))
```

```
614
614
154
154
```

```
[291] #Feature scaling
      sc_X = StandardScaler()
      X_train = sc_X.fit_transform(X_train)
      X_test = sc_X.transform(X_test)
```

```
[312] # Define the model: Init K-NN
      classifier = KNeighborsClassifier(n_neighbors=11, p=6,metric='euclidean')
```

```
[313] # Fit Model
      classifier.fit(X_train, y_train)

      KNeighborsClassifier(metric='euclidean', n_neighbors=11, p=6)
```

```
[314] # Predict the test set results
      y_pred = classifier.predict(X_test)
      y_pred

      array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
             0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
             1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1,
             1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
             0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
[315] # Evaluate Model
      cm = confusion_matrix(y_test, y_pred)
      print (cm)

      [[94 13]
       [15 32]]
```

```
print(accuracy_score(y_test, y_pred))

0.8181818181818182
```

References:

Dataset:
https://drive.google.com/file/d/1RjslN7snj-zcZpiFfA0WQddNMRhmzKq6/view?usp=sharing

Content:
https://neptune.ai/blog/knn-algorithm-explanation-opportunities-limitations
https://www.ibm.com/docs/en/ias?topic=procedures-k-nearest-neighbors-knn
https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761