Pima Indians Diabetes - KNN

Siyuan Zhang

1 Resources

Student: Siyuan Zhang
Language: Python
Resource: Pima Indians Diabetes from Kaggle

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt, matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
import time
import operator
from sklearn.preprocessing import StandardScaler
```

2 Dataset details

The dataset includes data from 768 women with 8 characteristics, in particular:

- **Pregnancies**: Number of times pregnant
- **Glucose**: Plasma glucose concentration 2 hours in an oral glucose tolerance test
- **Blood Pressure**: Diastolic blood pressure (mm Hg)
- **Skin Thickness**: Triceps skin fold thickness (mm)
- **Insulin**: 2-Hour serum insulin (mu U/ml)
- **BMI**: Body mass index (weight in kg/ (height in m)^2)
- **Diabetes Pedigree Function**: Diabetes pedigree function
- **Age**: Age (years)
- **Outcome**: Class variable (0 or 1)

Inspect the Dataset

```
#read the dataset
diabetes_data = pd.read_csv('diabetes.csv')
```

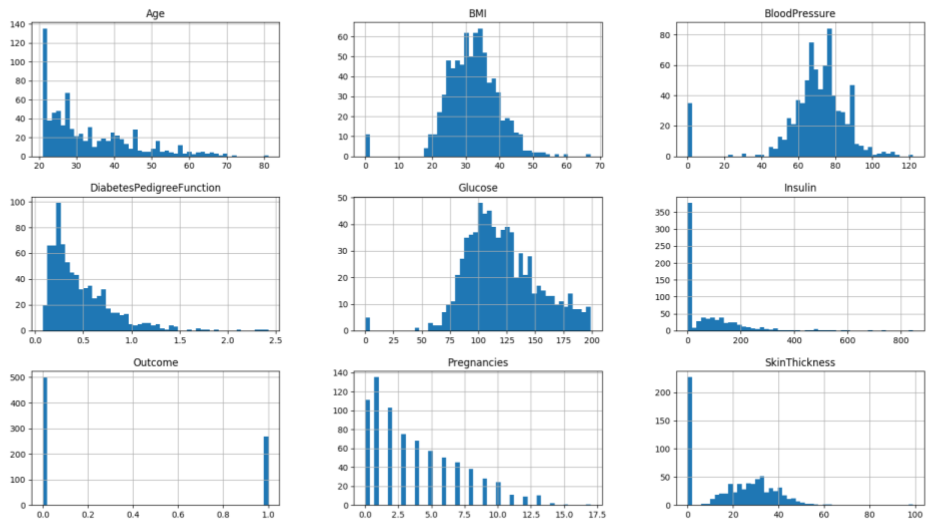
```
print(len(diabetes_data))
print(diabetes_data.head())
```

```
768
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1
```

Dataset Visualization

We can plot the dataset to visualize the data as histogram and chart

```
#visualization
diabetes_data.hist(bins=50, figsize=(20, 15))
plt.show()
```



Data Splitting

I split the dataset into training data (60%), validation data (20%) and testing data (20%), I used validation data to get the optimized K and test the testing data to get accuracy and confusion matrix.

```
#split dataset
# x is the columns
X = diabetes_data.values[:, 0:]
# y is the last column which is the result
y = diabetes_data.values[:, 8]
train, X_test, outcome, y_test = train_test_split(X, y, random_state=0, test_size=0.2)
X_train, X_valid, y_train, y_valid = train_test_split(train, outcome, random_state=2,test_size=0.25)
```

3 Algorithm Description

There are some data of zeros and null, which will negatively affect the accuracy of the result. Here, I replaced them with the median value of the columns where zeros and null locate.

```
#replace zeros and null
zeros_null = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
#change all zeros and null with median of those columns
for column in zeros_null:
    diabetes_data[column] = diabetes_data[column].replace(0, np.NaN)
    median = int(diabetes_data[column].mean(skipna=true))
    diabetes_data[column] = diabetes_data[column].replace(np.NaN, median)
```

Standardization

I standardized the dataset to change the attribute value to a Gaussian distribution with a mean of zero and a standard deviation of one. It is very useful when the algorithm expects the input features to be Gaussian.

```
#standardization
from sklearn.preprocessing import StandardScaler
rescaledX = StandardScaler()
X_train = rescaledX.fit_transform(X_train)
X_test = rescaledX.transform(X_test)
```

Equations:

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Min-Max scaling:

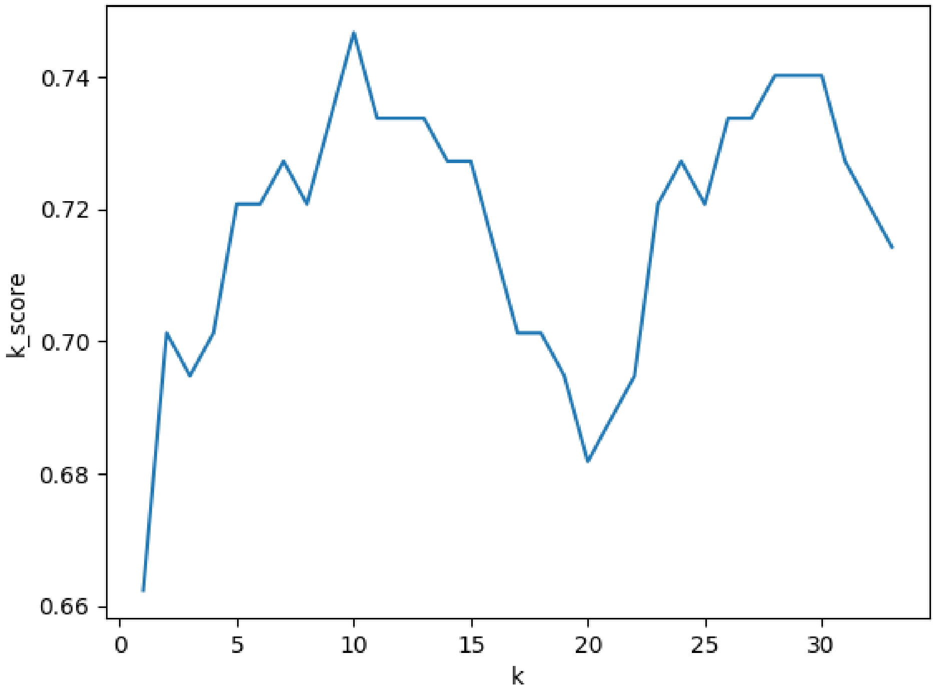
$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

4 Algorithm Results

Selection of K

Different selection of K will affect the accuracy significantly. To save time, here I selected 'test_images' as sample to get optimized accuracy with different K, by using the knn algorithm described above.

```
def diffK():
    k_score = []
    test_range = X_valid.shape[0]
    for k in range(1, 33 + 1):
        print("k = {} Training.".format(k))
        start = time.time()
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        predict_result = knn.predict(X_valid[:test_range])
        predict_score = accuracy_score(y_valid[:test_range], predict_result)
        k_score.append(predict_score)
        end = time.time()
        print("Score: {}.".format(predict_score))
        print("Complete time: {} Secs.".format(end - start))
    print(k_score)
    plt.plot(range(1, 33 + 1), k_score)
    plt.xlabel('k')
    plt.ylabel('k_score')
    plt.show()
```



We can know that when k is 10, the accuracy is the highest, which is 74.67%. So, I chose k = 10 to predict the testing data.

Algorithm accuracy and Confusion Matrix

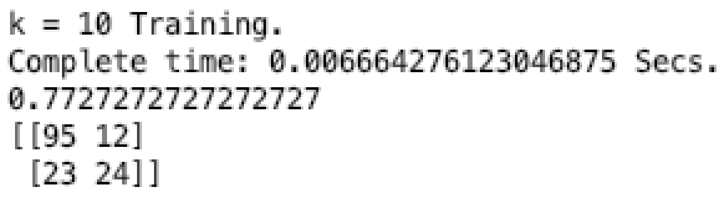
Predict the test data result and print the accuracy and confusion matrix

```
#predict the test result by k = 10
k = 10
print("k = {} Training.".format(k))
start = time.time()
resultList, accuracy = check(k)
end = time.time()
print("Complete time: {} Secs.".format(end - start))
print(accuracy)
#print the confusion matrix
```

```
cm = confusion_matrix(y_test, resultList)
print(cm)
```

Euclidean distance

Output:



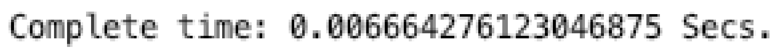
```
k = 10 Training.
Complete time: 0.006664276123046875 Secs.
0.7727272727272727
[[95 12]
 [23 24]]
```

The table below shows the true positive, true negative, false positive and false negative. In total 154 test samples, there are 95 true positives and 24 false negatives. So, the accuracy is (95 + 24) / 153 = 77.27%

	TRUE	FALSE
Positive	**95**	**12**
Negative	**23**	**24**

5 Runtime

For d dimension, we need $O(d)$to compute one distance between two data, then we need to sort the distance, which takes $O(nlogn)$, at last, we need to select K nearest neighbors. In total, the runtime is $O(nd + nlogn + k)$ to classify the data.



```
Complete time: 0.006664276123046875 Secs.
```

The program takes 0.00557 seconds.