Name: Chuchu Jin   Email:mayflywing@gwu.edu   Class: Machine Learning

DataSet : Diabetes

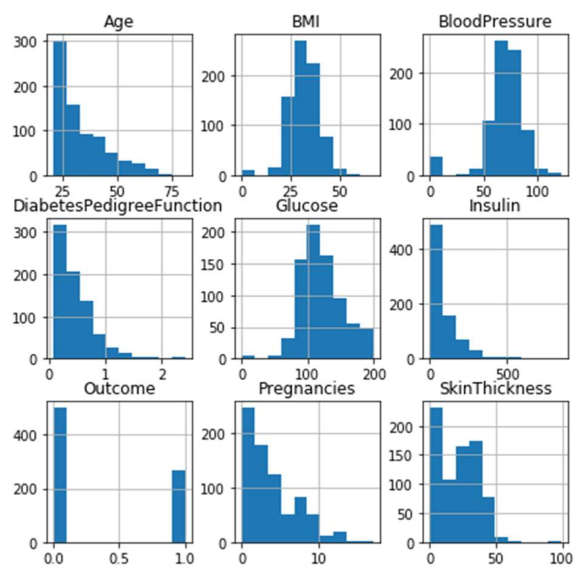
Dataset details

Take a quick look at the dataset and its data type.

	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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

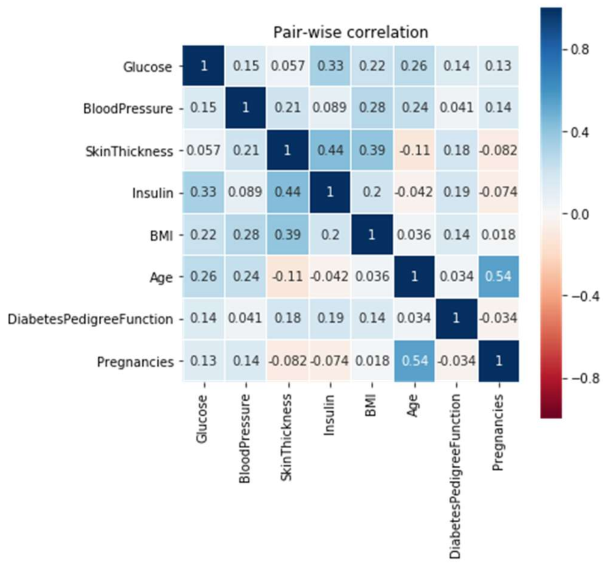
Visualize Analysis: Find the distribution of data.



Train/test split: Generally, use 90% of data for training, and 10% of data for testing. Shuffle the data first then split.

Algorithm Description

Data cleansing: Clean up data that should not be 0 (e.g. BMI cannot be 0). Since the data set is small, it cannot be deleted directly. Turn these 0 numbers into the median or average of other data.
Heat Map: From the picture of correlation of each pair of features, we found that there is no need to decrease the dimension



Feature Scaling: As the data have very different range of value, we need to scale the data to make it easy to train.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	0.468492	1.425995
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	-0.365061	-0.190672
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	0.604397	-0.105584
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	-0.920763	-1.041549
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	5.484909	-0.020496

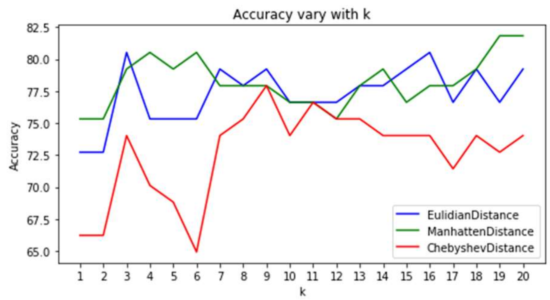
Distance Metrics: Use Euclidean distance, Manhattan distance and Chebyshev distance respectively.

Algorithm Results

Running results are as the picture below.

```
*******************************************
************    K =  1   *************
******   Distance =  Eulidian Distance *******
--> Calculating KNN...
[0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
--> Calculating Accuracy...
Accuracy =  72.727 %
The the count of true negatives is 40, false negatives is 11, true positives is 16, and false positives is 10
*******************************************
************    K =  2   *************
******   Distance =  Eulidian Distance *******
--> Calculating KNN...
[0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
```

Let K change between 1-20 and use three different distance metrics, save all the data and plot.
In general, the accuracy is higher when using Euclidean distance and Manhattan distance. When using the same distance metric, the accuracy is higher when k is larger.



K=19 Euclidian Distance

	Predict		
		Negative	Positive
Actual	Negative	43	7
	Positive	11	16

K=19 Manhatten Distance

	Predict		
		Negative	Positive
Actual	Negative	45	5
	Positive	9	18

K=19 ChebyshevDistance

	Predict		
		Negative	Positive
Actual	Negative	44	6
	Positive	15	12

Runtime

O（Train_Set_Size * Test_Set_Size * Features_Size）

The real wall time is 187ms.

```
%%time
knn(5,nX_train,nX_test,ny_train,ny_test,1)
```

```
*******************************************
************    K =  5   *************
******   Distance =  Eulidian Distance *******
--> Calculating KNN...
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
--> Calculating Accuracy...
Accuracy =  75.325 %
CPU times: user 196 ms, sys: 0 ns, total: 196 ms
Wall time: 187 ms
```

DataSet : Digtal Recognizer

Dataset details

Take a quick look at the dataset.

In[5]:
```
train_df.describe()
```
Out[5]:

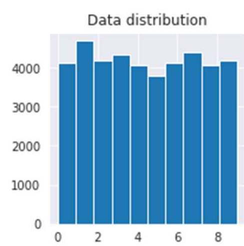
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	...	42000.000000	42000.000000	42000.000000	42000.00000	42000.000000	42
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.219286	0.117095	0.059024	0.02019	0.017238	
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.312890	4.633819	3.274488	1.75987	1.894498	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	254.000000	253.000000	253.00000	254.000000	

8 rows × 785 columns

+ Code + Markdown

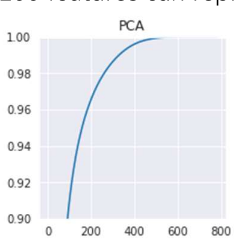
In[6]:
```
train_df.shape
```
Out[6]:
```
(42000, 785)
```

Visualize Analysis: Take out the first few data and draw it to see what it is like. Find the distribution of data.



Algorithm Description

Feature Scaling: We don't need to scale this data set because the data is already in the same scale.
PCA: As the data have very different range of value (shows in the picture of data distribution), we need to scale the data to make it easy to train. From the PCA , we can compress the data set as only 100 features can represent 92% of the data set, so we compress the features to 100



Distance Metrics: Use Euclidean distance

Algorithm Results

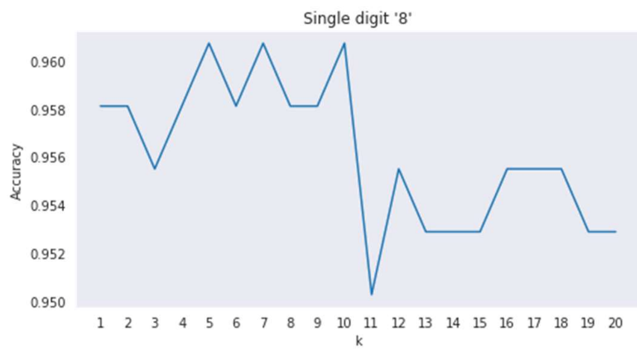
Using k=5, Euclidean distance, we can get a confusion matrix as below.

```
confusion_matrix(y_test,pred)
```

```
array([[410,   0,   0,   0,   0,   0,   1,   0,   0,   0],
       [  0, 481,   1,   1,   0,   0,   0,   1,   1,   0],
       [  2,   0, 396,   0,   0,   0,   2,   1,   1,   1],
       [  0,   1,   1, 404,   0,   4,   0,   0,   7,   1],
       [  0,   2,   0,   0, 447,   0,   3,   1,   0,   8],
       [  2,   0,   0,   3,   0, 357,   5,   0,   2,   3],
       [  4,   1,   0,   0,   0,   1, 407,   0,   0,   0],
       [  1,   4,   3,   0,   1,   0,   0, 428,   0,   9],
       [  0,   4,   1,   2,   0,   1,   1,   1, 367,   5],
       [  2,   0,   0,   2,   3,   1,   0,   2,   0, 399]])
```

Array[0][0] means the label is 0 and we predict it to be 0, array[0][1] means the label is 0 and we predict it to be 1, etc.

Let K change between 1-20, we choose digit ‘8’ to calculate accuracy.



When k = 5, the accuracy is the highest.

Runtime

O（Train_Set_Size * Test_Set_Size * Features_Size）

The real wall time is 1 h 11 min 52s. (For the whole data set)

I think this is because of the efficiency of underlying Python operations and the lack of optimization of the KNN algorithm. So I only take part of data to calculate the accuracy of single digit.

```
--> Calculating Accuracy...
Accuracy =  97.548 %
CPU times: user 1h 11min 51s, sys: 504 ms, total: 1h 11min 52s
Wall time: 1h 11min 52s
```