DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Project Based Learning Report on:

"Analysis of the Mathematical Model for KNN Algorithm and predict the survival of passengers in Titanic using the same with Python."

BACHELOR OF ENGINEERING
in
INFORMATION SCIENCE AND ENGINEERING
by

Gaurav Sarraf	**1BY16IS012**
Saikeerthi Reddy	**1BY16IS038**
Thrivikram Mudunuri	**1BY16IS057**

Under the Guidance of
Dr. S. K. PUSHPA
Associate Professor

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
BENGALURU-560064
2017-18

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project based learning (15CS43) entitled "Analysis of the Mathematical Model for KNN Algorithm and predict the survival of passengers in Titanic using the same with Python." is a bonafide work carried out by **Mr. Gaurav Sarraf (1BY16IS012), Ms. Saikeerthi Reddy(1BY16IS038), Mr. Thrivikram Mudunuri (1BY16IS057)** during the academic year 2017-18. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the departmental library.

Signature of the guide	**Signature of the Coordinator**	**Signature of the HOD**
Dr. Pushpa S. K	**Dr. Pushpa S. K**	**Dr. Manjunath T N**

“Analysis of the Mathematical Model for KNN Algorithm and predict the survival of passengers in Titanic using the same with Python.”

Table of Contents

Introduction

1.1 Abstract:

The KNN algorithm is a robust and versatile classifier that is often used as a benchmark for more complex classifiers such as Artificial Neural Networks (ANN) and Support Vector Machines (SVM). Despite its simplicity, KNN can outperform more powerful classifiers and is used in a variety of applications such as economic forecasting, data compression and genetics. For example, KNN was leveraged in a 2006 study of functional genomics for the assignment of genes based on their expression profiles.

What is KNN?

Let's first start by establishing some definitions and notations. We will use x to denote a feature (aka. predictor, attribute) and y to denote the target (aka. label, class) we are trying to predict.

KNN falls in the supervised learning family of algorithms. Informally, this means that we are given a labelled dataset consisting of training observations (x,y) and would like to capture the relationship between x and y. More formally, our goal is to learn a function h:X→Y so that given an unseen observation x, h(x) can confidently predict the corresponding output y.

Non-parametric means it makes no explicit assumptions about the functional form of h, avoiding the dangers of mismodeling the underlying distribution of the data. For example, suppose our data is highly non-Gaussian but the learning model we choose assumes a Gaussian form. In that case, our algorithm would make extremely poor predictions.

Instance-based learning means that our algorithm doesn't explicitly learn a model. Instead, it chooses to memorize the training instances which are subsequently used as "knowledge" for the prediction phase. Concretely, this means that only when a query to our database is made (i.e. when we ask it to predict a label given an input), will the algorithm use the training instances to spit out an answer.
KNN is non-parametric, instance-based and used in a supervised learning setting.

It is worth noting that the minimal training phase of KNN comes both at a memory cost, since we must store a potentially huge data set, as well as a computational cost during test time since classifying a given observation requires a rundown of the whole data set. Practically speaking, this is undesirable since we usually want fast responses.

1.2 Working:

In the classification setting, the K-nearest neighbor algorithm essentially boils down to forming a majority vote between the K most similar instances to a given “unseen” observation. Similarity is defined according to a distance metric between two data points. A popular choice is the Euclidean distance given by

$$d(x,x')=\sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \cdots + (x_n - x'_n)^2}$$

but other measures can be more suitable for a given setting and include the Manhattan, Chebyshev and Hamming distance.

More formally, given a positive integer K, an unseen observation x and a similarity metric d, KNN classifier performs the following two steps:

It runs through the whole dataset computing d between x and each training observation. We’ll call the K points in the training data that are closest to x the set A. Note that K is usually odd to prevent tie situations.

It then estimates the conditional probability for each class, that is, the fraction of points in A with that given class label. (Note I(x) is the indicator function which evaluates to 1 when the argument x is true and 0 otherwise)

$$P(y = j|X = x) = \frac{1}{K}\sum_{i \in A} I(y(i) = j)$$

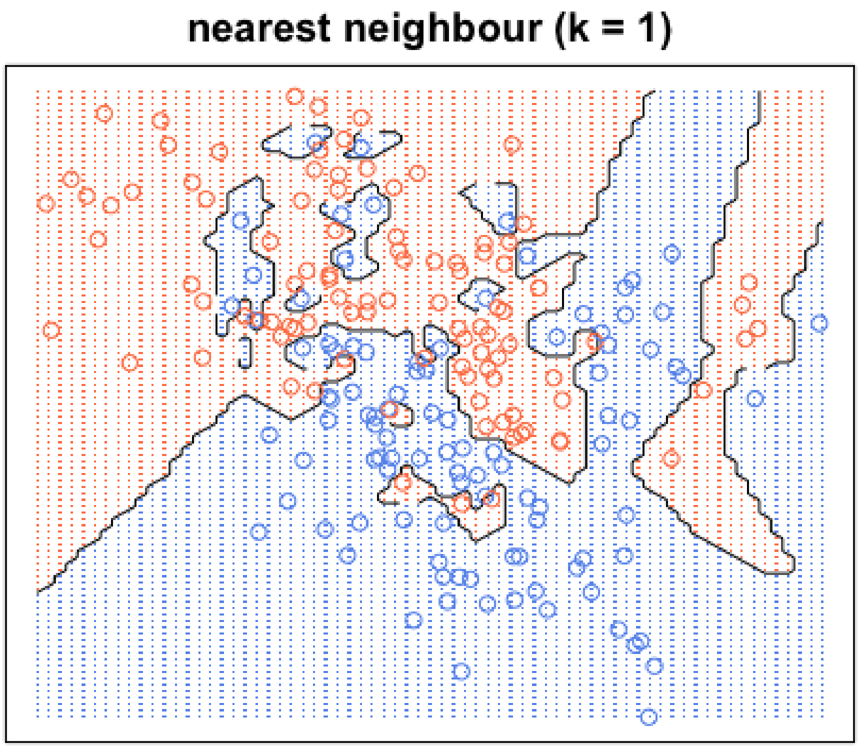
Finally, our input x gets assigned to the class with the largest probability.

An alternate way of understanding KNN is by thinking about it as calculating a decision boundary (i.e. boundaries for more than 2 classes) which is then used to classify new points

1.3 More About KNN:

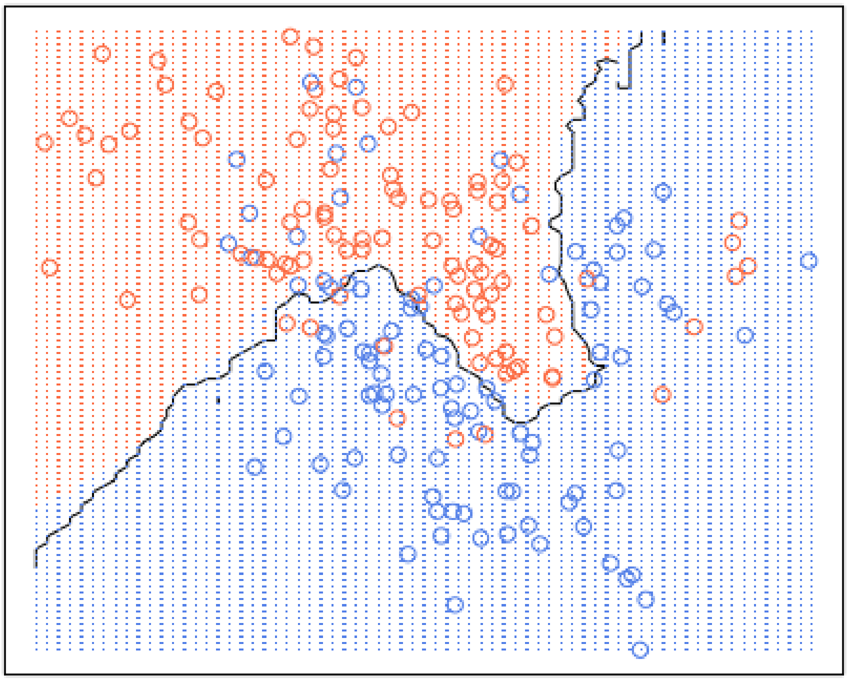
At this point, you're probably wondering how to pick the variable K and what its effects are on your classifier. Well, like most machine learning algorithms, the K in KNN is a hyperparameter that you, as a designer, must pick in order to get the best possible fit for the data set. Intuitively, you can think of K as controlling the shape of the decision boundary we talked about earlier.

When K is small, we are restraining the region of a given prediction and forcing our classifier to be "more blind" to the overall distribution. A small value for K provides the most flexible fit, which will have low bias but high variance. Graphically, our decision boundary will be more jagged.



On the other hand, a higher K averages more voters in each prediction and hence is more resilient to outliers. Larger values of K will have smoother decision boundaries which means lower variance but increased bias.

20-nearest neighbour



Background

2.1 Introduction:

The model for KNN is the entire training dataset. When a prediction is required for a unseen data instance, the KNN algorithm will search through the training dataset for the k-most similar instances. The prediction attribute of the most similar instances is summarized and returned as the prediction for the unseen instance. The similarity measure is dependent on the type of data. For real-valued data, the Euclidean distance can be used. Other types of data such as categorical or binary data, Hamming distance can be used.

In the case of regression problems, the average of the predicted attribute may be returned. In the case of classification, the most prevalent class may be returned.
The KNN algorithm is belongs to the family of instance-based, competitive learning and lazy learning algorithms.

Instance-based algorithms are those algorithms that model the problem using data instances (or rows) in order to make predictive decisions. The KNN algorithm is an extreme form of instance-based methods because all training observations are retained as part of the model.

It is a competitive learning algorithm, because it internally uses competition between model elements (data instances) in order to make a predictive decision. The objective similarity measure between data instances causes each data instance to compete to "win" or be most similar to a given unseen data instance and contribute to a prediction.

Lazy learning refers to the fact that the algorithm does not build a model until the time that a prediction is required. It is lazy because it only does work at the last second. This has the benefit of only including data relevant to the unseen data, called a localized model. A disadvantage is that it can be computationally expensive to repeat the same or similar searches over larger training datasets.
Finally, KNN is powerful because it does not assume anything about the data, other than a distance measure can be calculated consistently between any two instances. As such, it is called non-parametric or non-linear as it does not assume a functional form.

This is broken down into the following steps:

- Handle Data: Open the dataset from CSV and split into test/train datasets.
- Similarity: Calculate the distance between two data instances.
- Neighbors: Locate k most similar data instances.
- Response: Generate a response from a set of data instances.
- Accuracy: Summarize the accuracy of predictions.
- Main: Tie it all together.

2.2 Algorithm and Pseudocode:

Generic for KNN:

1. Calculate "$d(x, x_i)$" $i = 1, 2, \ldots, n$; where d denotes the Euclidean distance between the points.
2. Arrange the calculated n Euclidean distances in non-decreasing order.
3. Let k be a +ve integer, take the first k distances from this sorted list.
4. Find those k-points corresponding to these k-distances.
5. Let k_i denotes the number of points belonging to the i^{th} class among k points i.e. $k \geq 0$
6. If $k_i > k_j \; \forall \; i \neq j$ then put x in class i.

For Titanic Problem:

```
1. begin
2. initialize the n×n distance matrix D , initialize the Ω×Ω confusion matrix C , set t←0
, TotAcc←0 , and set NumIterations equal to the desired number of iterations (re-
partitions).
3. calculate distances between all the input samples and store in n×n matrix D . (For a
large number of samples, use only the lower or upper triangular of D for storage
since it is a square symmetric matrix.)
    for t← 1 to Num Iterations do
            set C←0, and ntotal←0 .
            partition the input samples into κ equally-sized groups.
    for fold← 1 to κ do
            assign samples in the fourth partition to testing, and use the remaining
samples for training. Set the number of samples used for testing as test .
            set ntotal←ntotal+ntest.
    for i ← 1 to test do
            for test sample xi determine the k closest training samples based on the
calculated distances. determine ω^ , the most frequent class label among the k
closest training samples.
```

```
            increment confusion matrix C by 1 in element cω,ω^ , where ω is the true
    and ω^ the predicted class label for test sample xi . If ω=ω^ then the increment of +1
    will occur on the diagonal of the confusion matrix, otherwise, the increment will
    occur in an off-diagonal.
            determine the classification accuracy using Acc=∑Ωjcjjntotal where cjj is a
    diagonal element of the confusion matrix C .
      calculate TotAcc=TotAcc+Acc .
      calculate AvgAcc=TotAcc/NumIterations
    4. end
```

2.3 Design and Mathematical Model:

In this subsection, we describe the problem of classification and notation used to model the dataset. The problem of classification is to estimate the value of the class variable based on the values of one or more independent variables (known as feature variables). We model the tuple as {x, y} where x is an ordered set of attribute values like $\{x_1, x_2, \ldots, x_d\}$ and y is the class variable to be predicted. Here xi is the value of the i^{th} attribute and there are d attributes overall corresponding to a d-dimensional space. Formally, the problem has the following inputs:

- A set of n tuples called the training dataset, $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.
- A query tuple x_t .

The output is an estimated value of the class variable for the given query x_t, mathematically it can be expressed as:

$$y_t = f(x_t, D, parameters), \qquad (1.2)$$

Where parameters are the arguments that the function f() takes. These are generally set by the user or are learned by some method

We present a mathematical model for KNN algorithm and show that KNN only makes use of local prior probabilities for classification. For a given query instance x_t, KNN algorithm works as follows:

$$y_t = arg\ max \sum_{x_i \epsilon N\ (x_t,k)} E(y_i, c) \qquad (1.3)$$

Where y_t is the predicted class for the query instance x_t and m is the number of classes present in the data.

Also

$$E(a,b) = \{\begin{matrix} 1 & if\ a = b \\ 0 & else \end{matrix} \qquad (1.4)$$

N(x, k) = Set of K nearest neighbor of x

Eq. (1.3) can also be written as:

$$y_t = \arg\max \left\{ \sum_{x_i \in N(x_t,k)} E(y_i, c_1), \sum_{x_i \in N(x_t,k)} E(y_i, c_2), \quad \ldots \quad , \sum_{x_i \in N(x_t,k)} E(y_i, c_m) \right\} \qquad (1.5)$$

$$y_t = \arg\max \left\{ \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_1)}{k}, \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_2)}{k}, \quad \ldots \quad , \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_m)}{k} \right\} \qquad (1.6)$$

and we know that

$$p(c_j)_{(x_t,k)} = \sum_{x_i \in N(x_t,k)} \frac{E(y_i, c_j)}{k} \qquad (1.7)$$

Where $p(c_j)_{(xt,k)}$ is the probability of occurrence of j^{th} class in the neighborhood of x_t . Hence Eq. 1.6 turns out to be

y_t = arg max {$p(c_1)_{(xt,k)}$, $p(c_2)_{(xt,k)}$, . . . , $p(c_m)_{(xt,k)}$} (1.8)

It is clear from Eq. 1.8, that KNN algorithm uses only prior probabilities to calculate the class of the query instance. It ignores the class distribution around the neighborhood of query point.

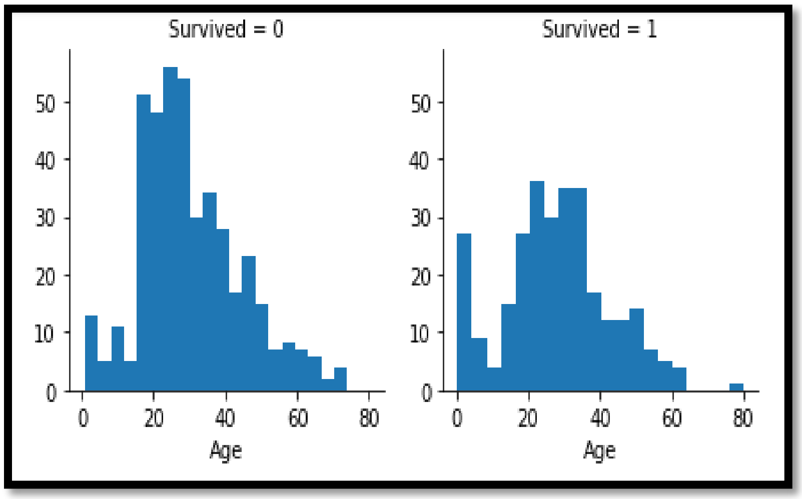
2.4 Code and Results:

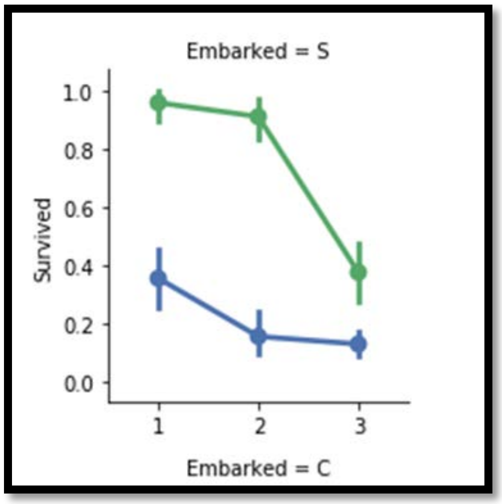
Source Code:

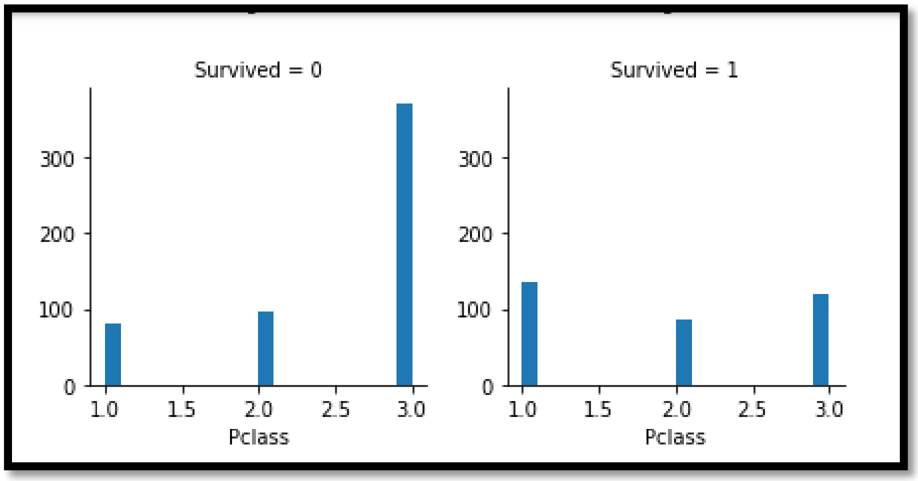
```
        ##Importing Packages###
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
        ##Getting data##
train = pd.read_csv('C:/Users/sarra/Downloads/train.csv')
test = pd.read_csv('C:/Users/sarra/Downloads/test.csv')
combine = [train,test]
complete=pd.concat(combine)
        ##basic data exploration##
complete.describe()
complete.isnull().sum()
print(complete.columns.values)
complete.info()
train[['Pclass', 'Survived']].groupby(['Pclass'],
     as_index=False).mean().sort_values(by='Survived',
ascending=False)
train[['Sex', 'Survived']].groupby(['Sex'],
     as_index=False).mean().sort_values(by='Survived',
ascending=False)
train[['Parch', 'Survived']].groupby(['Parch'],
     as_index=False).mean().sort_values(by='Survived',
ascending=False)
train[['SibSp', 'Survived']].groupby(['SibSp'],
     as_index=False).mean().sort_values(by='Survived',
ascending=False)
        ##Data Vizualization##
graph1 = sns.FacetGrid(train, col='Survived')
graph1.map(plt.hist, 'Age', bins=20)
graph2 = sns.FacetGrid(train, col='Survived')
graph2.map(plt.hist, 'Pclass', bins=20)
graph3 = sns.FacetGrid(train, col='Survived', row='Pclass')
```

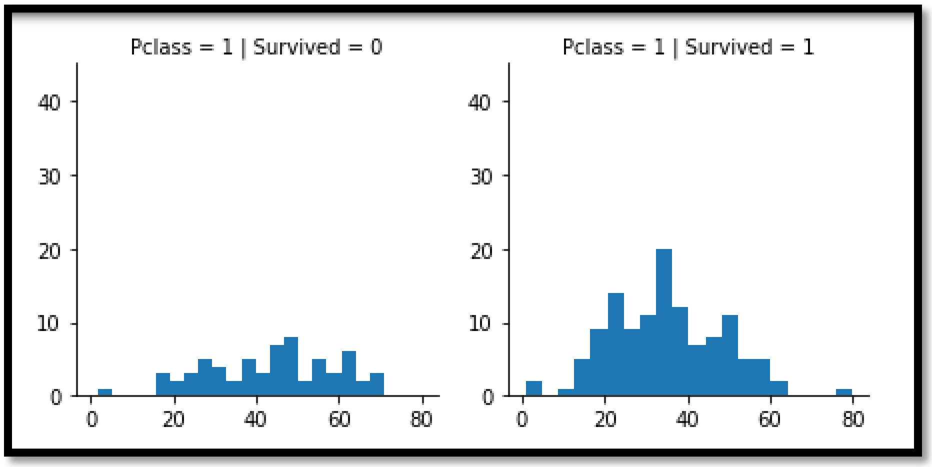
```
graph3.map(plt.hist, 'Age', bins=20)
graph3.add_legend()
graph4 = sns.FacetGrid(train, row='Embarked')
graph4.map(sns.pointplot, 'Pclass', 'Survived', 'Sex',
palette='deep')
graph4.add_legend()
graph5 = sns.FacetGrid(train, row='Embarked', col='Survived')
graph5.map(sns.barplot, 'Sex', 'Fare', ci=None)
graph5.add_legend()
        ##Feature Selection ##
complete=complete.drop(['Ticket', 'Cabin'], axis=1)
complete['Title'] = complete.Name.str.extract(' ([A-Za-
z]+)\.',expand=False)
complete['Title'] = complete['Title'].replace(['Lady',
'Countess','Capt', 'Col',\
 'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'],
'Rare')
complete['Title'] = complete['Title'].replace('Mlle', 'Miss')
complete['Title'] = complete['Title'].replace('Ms', 'Miss')
complete['Title'] = complete['Title'].replace('Mme', 'Mrs')
complete[['Title','Survived']].groupby(['Title'],
as_index=False).mean()
encoder = preprocessing.LabelEncoder()
complete['Title']=encoder.fit_transform(complete['Title'])
list(encoder.classes_)
##dropping name and PassengerID as required info. captured in
Title
complete=complete.drop(['PassengerId'], axis=1)
complete=complete.drop(['Name'], axis=1)
encoder2 = preprocessing.LabelEncoder()
complete['Sex']=encoder2.fit_transform(complete['Sex'])
        ##Filling mising values for Age
guess_age = np.zeros((2,3))
graph6 = sns.FacetGrid(train,row='Pclass',col='Sex')
graph6.map(plt.hist, 'Age')
graph6.add_legend()
for i in range(0,2):
    for j in range(0,3):
        guess=complete[(complete['Sex']==i)&\
(complete['Pclass']==j+1)]['Age'].dropna()
age_median=guess.median()
        guess_age[i,j]=int(age_median/0.5+0.5)*0.5
for i in range(0,2):
    for j in range(0,3):

complete.loc[(complete.Age.isnull())&(complete.Sex==i)&\
(complete.Pclass==j+1),'Age']=guess_age[i,j]

complete['Age']=complete['Age'].astype(int)
complete.isnull().sum()
        ##Putting Age data in classes
complete['age_band']=pd.cut(train['Age'],5)
complete[['age_band', 'Survived']].groupby(['age_band'],
     as_index=False).mean().sort_values(by='age_band',
ascending=True)
complete.loc[complete['Age']<=16,'Age']=0
complete.loc[(complete['Age']>16) &
(complete['Age']<=32),'Age']=1
complete.loc[(complete['Age']>32) &
(complete['Age']<=48),'Age']=2
complete.loc[(complete['Age']>48) &
(complete['Age']<=64),'Age']=3
complete.loc[(complete['Age']>64) &
(complete['Age']<=80),'Age']=4
complete=complete.drop(['age_band'],axis=1)
        ##Creating new feature 'Family Size'
complete['fam_size']=complete['SibSp']+complete['Parch']+1
complete[['fam_size','Survived']].groupby(['fam_size'],
        as_index=False).mean().sort_values(by='Survived',
ascending=False)
complete=complete.drop(['SibSp','Parch'])
        ##Filling missing values for Embarked
freq_port = complete.Embarked.dropna().mode()[0]
complete['Embarked']=complete['Embarked'].fillna(freq_port)
encoder3=preprocessing.LabelEncoder()
complete['Embarked']=encoder3.fit_transform(complete['Embarke
d'])
complete[['Embarked','Survived']].groupby(['Embarked'],
as_index=False).mean().sort_values(by='Survived')
        ##Putting Fare rates in classes
complete['fare_band']=pd.qcut(complete['Fare'],4)
complete[['fare_band','Survived']].groupby(['fare_band'],
        as_index=False).mean().sort_values(by='fare_band')
complete.loc[complete['Fare']<=7.896,'Fare']=0
complete.loc[(complete['Fare']>7.896) &
(complete['Fare']<=14.454),'Fare']=1
complete.loc[(complete['Fare']>14.454) &
(complete['Fare']<=31.275),'Fare']=2
complete.loc[(complete['Fare']>31.275) &
(complete['Fare']<=512.329),'Fare']=3
            #Filling missing values for Fare
freq_fare = complete.Fare.dropna().mode()[0]
complete['Fare']=complete['Fare'].fillna(freq_fare)
complete=complete.drop(['fare_band'],axis=1)
complete.info()
        ##Model fitting and Prediction using k
        ##Reextracting train and test datasets
X=complete.drop(['Survived'],axis=1)
Y=pd.DataFrame(complete['Survived'])
X_train=X.iloc[0:891,:]
Y_train=Y.iloc[0:891,:]
X_test=X.iloc[891:1309,:]
Y_test=Y.iloc[891:1310,:]
KNN = KNeighborsClassifier(n_neighbors=3)
KNN.fit(X_train, Y_train)
Y_pred = KNN.predict(X_test)
accuracy=round(KNN.score(X_train,Y_train)*100,2)
accuracy
final_prediction=pd.DataFrame({'PassengerId':test['PassengerI
d'], 'Survived':Y_pred})
final_prediction.to_csv('C:/Users/sarra/Downloads/titanic_sur
vival.csv', index=False)
```

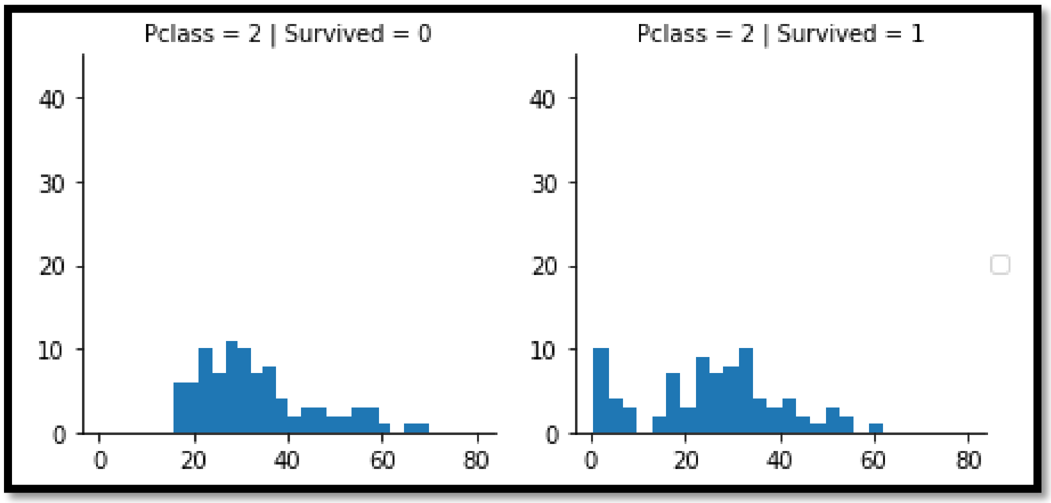
Output:

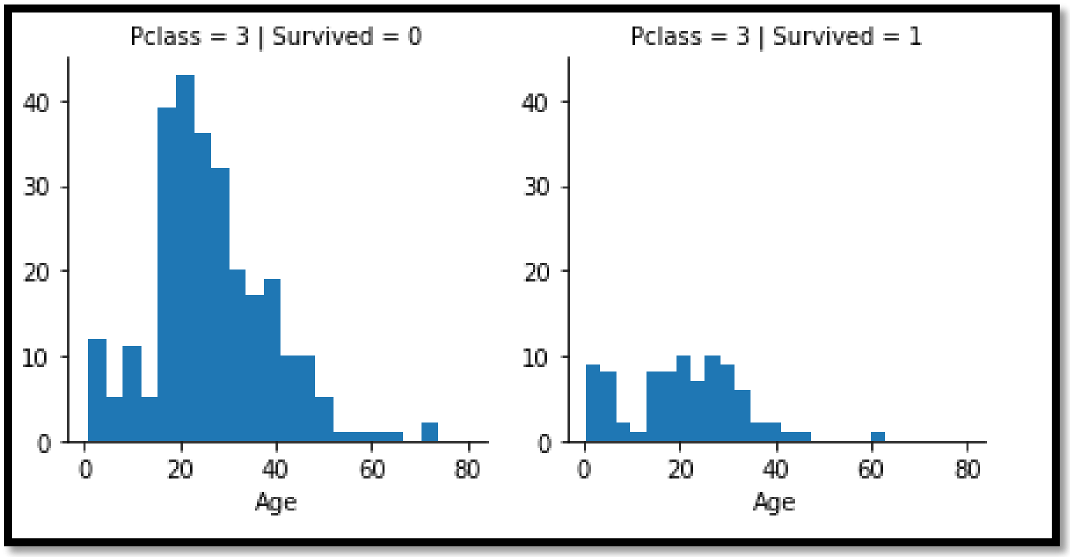

Survived = 0
Survived = 1
Age
Age

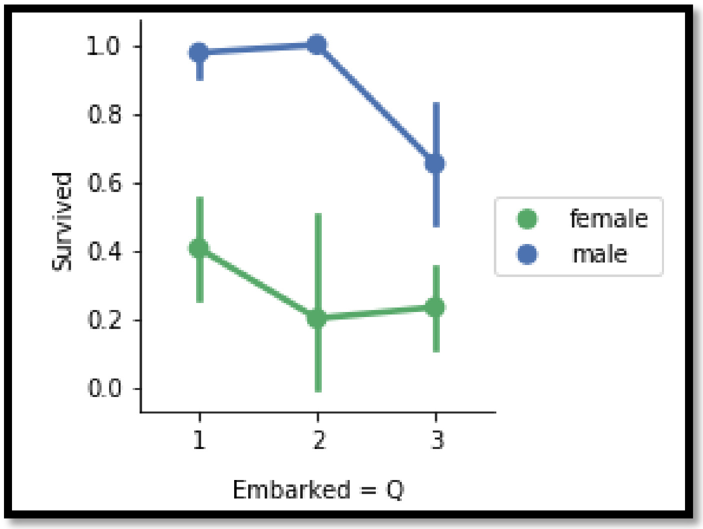

Embarked = S
Survived
Embarked = C

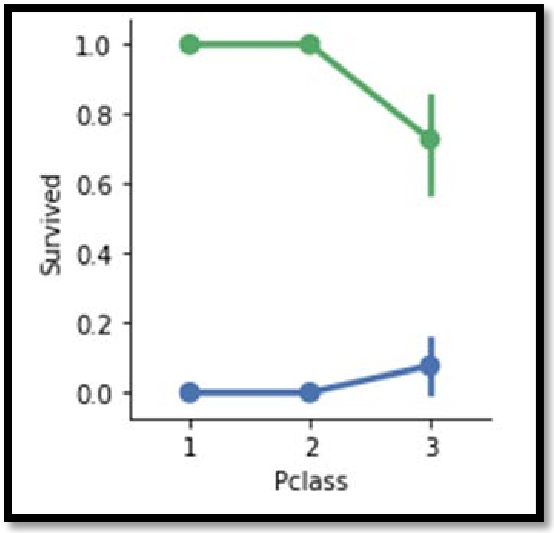

Survived = 0
Survived = 1
Pclass
Pclass

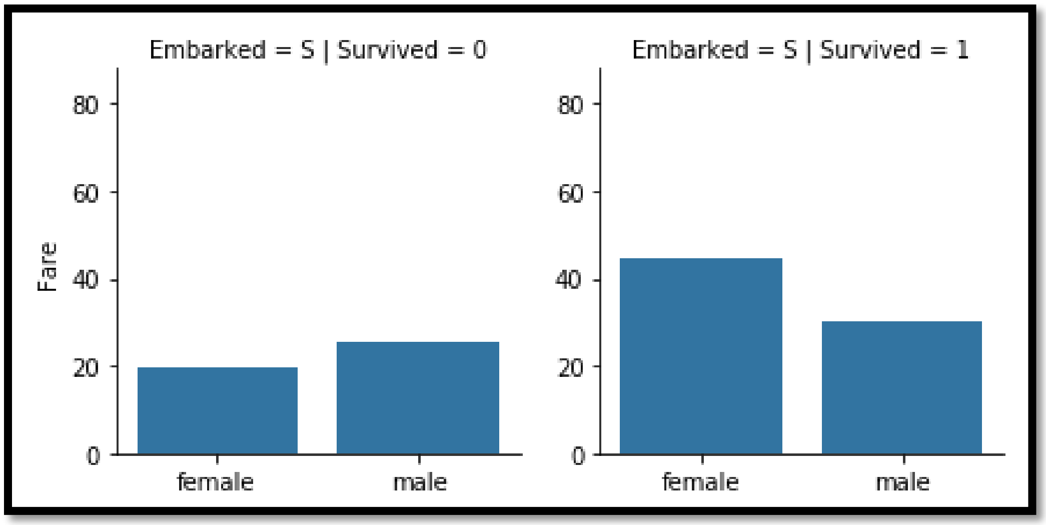

Pclass = 1 | Survived = 0
Pclass = 1 | Survived = 1

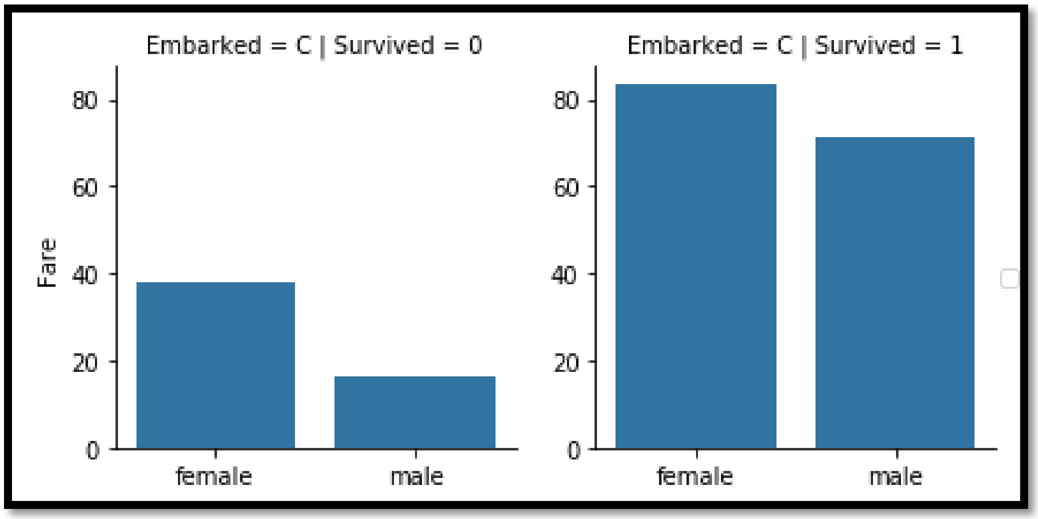

Pclass = 2 | Survived = 0
Pclass = 2 | Survived = 1
0
10
20
30
40
0
20
40
60
80

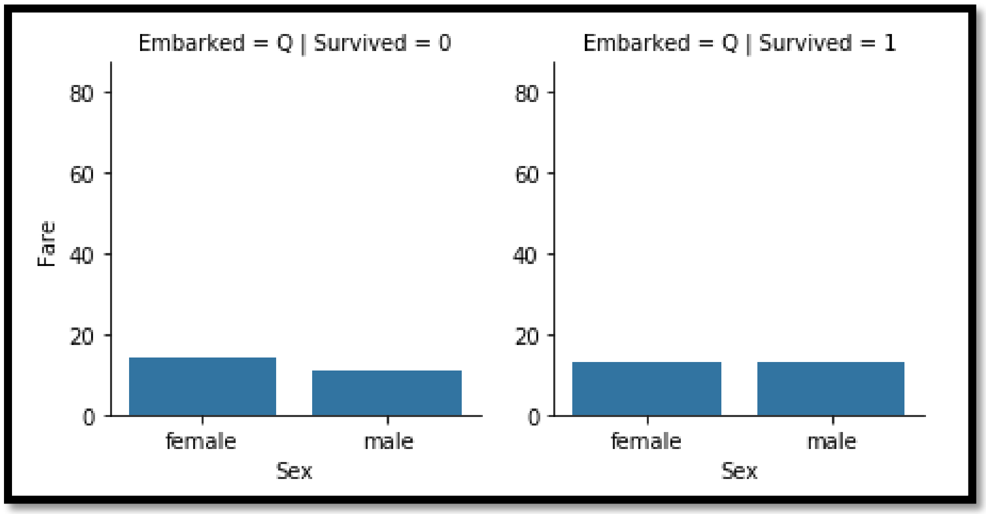

Pclass = 3 | Survived = 0
Pclass = 3 | Survived = 1
0
10
20
30
40
0
20
40
60
80
Age
Age

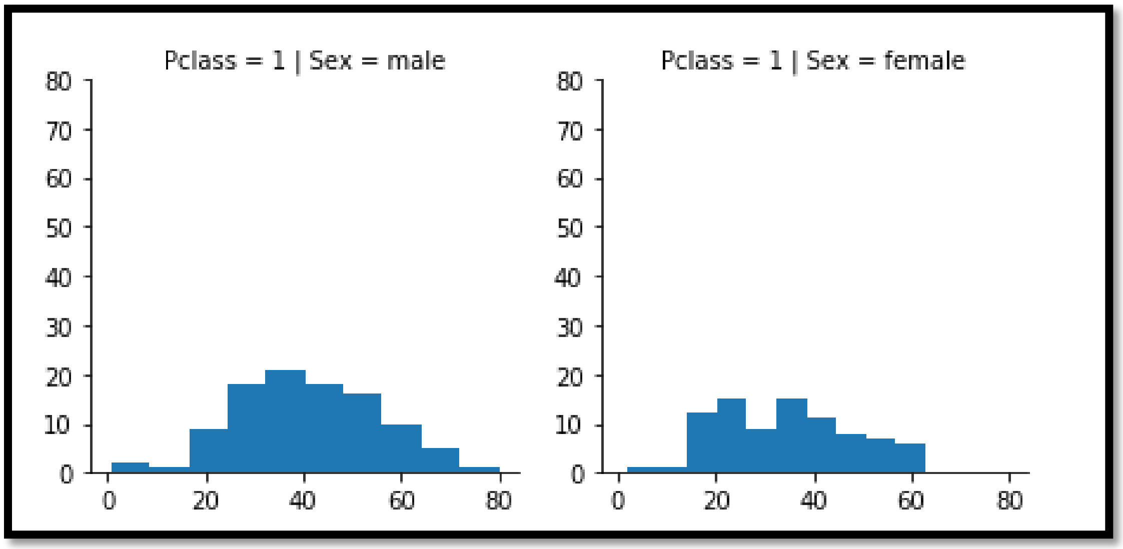

Survived
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
Embarked = Q
● female
● male

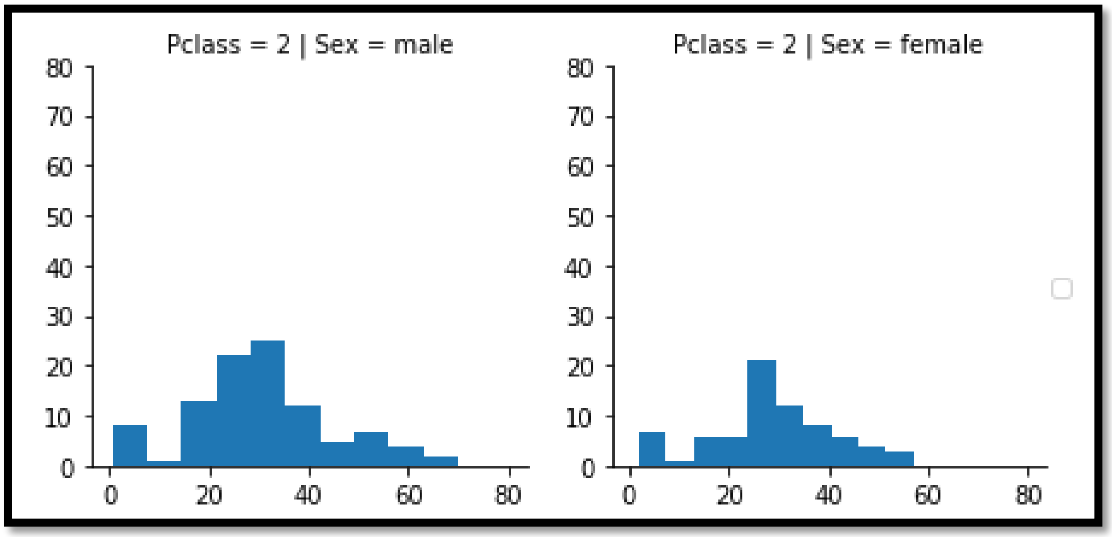

Survived
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
Pclass

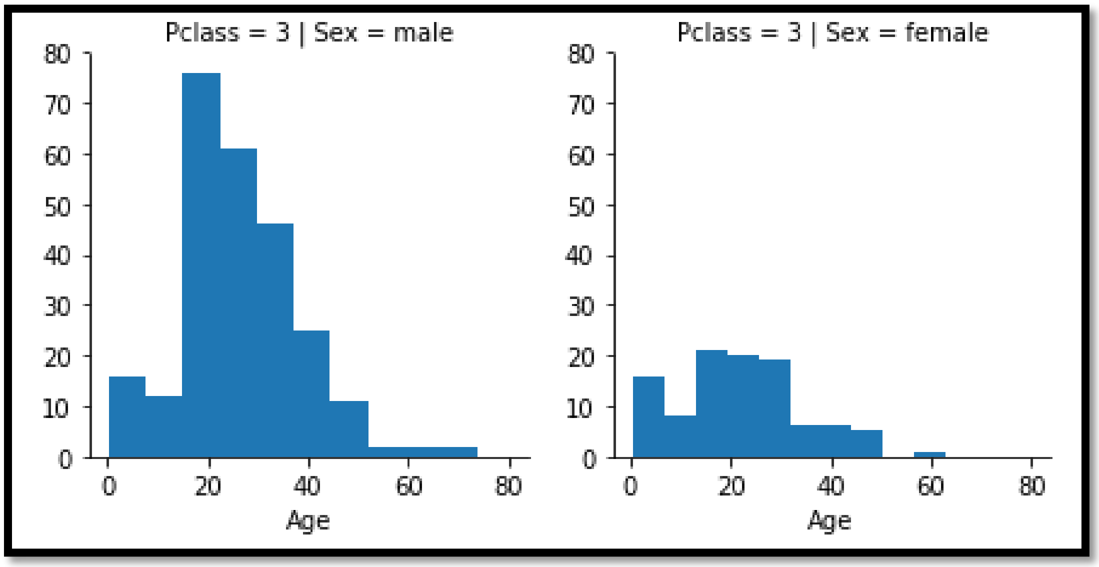

Embarked = S | Survived = 0
Embarked = S | Survived = 1
Fare
80
60
40
20
0
female
male


Embarked = C | Survived = 0
Embarked = C | Survived = 1
Fare
80
60
40
20
0
female
male


Embarked = Q | Survived = 0
Embarked = Q | Survived = 1
Fare
80
60
40
20
0
female
male
Sex


Pclass = 1 | Sex = male
Pclass = 1 | Sex = female


Pclass = 2 | Sex = male
Pclass = 2 | Sex = female


Pclass = 3 | Sex = male
Pclass = 3 | Sex = female
Age
Age

Statistical Data on the trained model:

Pass ID: Passenger ID, Survived: 0=dead ; 1=alive

Pass ID	Survived	Pass ID	Survived	Pass ID	Survived	Pass ID	Survived
892	0	936	1	987	0	1035	0
893	0	937	0	988	1	1036	1
894	0	938	0	989	0	1037	0
895	0	939	0	990	1	1038	0
896	0	940	1	991	1	1039	0
897	0	941	1	992	1	1040	1
898	1	942	1	993	0	1041	1
899	0	943	0	994	0	1042	1
900	1	944	1	995	0	1043	0
901	0	945	1	996	0	1044	0
902	0	946	0	997	0	1045	1
903	1	947	0	998	0	1046	0
904	1	948	0	999	0	1047	0
905	0	949	0	1000	0	1048	1
906	1	950	0	1006	1	1049	1
907	1	951	1	1007	0	1050	1
908	0	952	0	1008	0	1051	1
909	0	953	0	1009	1	1052	1
910	0	954	0	1010	0	1053	1
911	1	955	1	1011	1	1054	1
912	1	956	1	1012	1	1055	0
913	1	957	1	1013	0	1056	0
914	1	958	1	1014	1	1057	0
915	1	959	0	1015	0	1058	0
916	1	960	0	1016	0	1059	0
917	0	961	0	1017	1	1060	1
918	1	962	1	1018	0	1061	1
919	0	963	0	1019	1	1062	0
920	1	964	1	1020	0	1063	0
921	1	965	0	1021	0	1064	0
922	0	966	1	1022	0	1065	0
923	0	968	0	1023	0	1066	0
924	1	969	1	1024	0	1067	1
925	0	970	0	1025	0	1068	1
926	0	971	1	1026	0	1069	1
927	0	978	1	1027	0	1070	1
928	1	979	1	1028	0	1071	1
929	1	980	1	1029	0	1072	0
930	0	981	1	1030	1	1073	1
931	1	982	0	1031	0	1074	1
932	0	983	0	1032	0	1075	0
933	1	984	1	1033	1	1076	1
934	0	985	0	1034	0	1077	0

2.5 Applications:

Let’s assume a money lending company “XYZ” like UpStart, IndiaLends, etc. Money lending XYZ company is interested in making the money lending system comfortable & safe for lenders as well as for borrowers. The company holds a database of customer’s details.

Using customer’s detailed information from the database, it will calculate a credit score(discrete value) for each customer. The calculated credit score helps the company and lenders to understand the credibility of a customer clearly. So they can simply take a decision whether they should lend money to a particular customer or not.

The customer’s details could be:

- Educational background details.
- ➢ Highest graduated degree.
- ➢ Cumulative grade points average (CGPA) or marks percentage.
- ➢ The reputation of the college.
- ➢ Consistency in his lower degrees.
- ➢ Whether to take the education loan or not.
- ➢ Cleared education loan dues

- Employment details.
- ➢ Salary.
- ➢ Year of experience.
- ➢ Got any onsite opportunities.
- ➢ Average job change duration.

The company(XYZ) use’s these kinds of details to calculate credit score of a customer. The process of calculating the credit score from the customer’s details is expensive. To reduce the cost of predicting credit score, they realized that the customers with similar background details are getting a similar credit score.

So, they decided to use already available data of customers and predict the credit score using it by comparing it with similar data. These kinds of problems are handled by the k-nearest neighbor classifier for finding the similar kind of customers.

3.1 Conclusions

Advantages of K-nearest neighbors algorithm:

- KNN is simple to implement.
- KNN executes quickly for small training data sets.
- performance asymptotically approaches the performance of the Bayes Classifier.
- Don't need any prior knowledge about the structure of data in the training set.
- No retraining is required if the new training pattern is added to the existing training set.

Limitation to K-nearest neighbors algorithm:

- When the training set is large, it may take a lot of space.
- For every test data, the distance should be computed between test data and all the training data. Thus a lot of time may be needed for the testing.

3.2 References:

- http://dataaspirant.com/2016/12/23/k-nearest-neighbor-classifier-intro/
- http://www.scholarpedia.org/article/K-nearest_neighbor
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2243774/?page=1
- https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)
- http://me.seekingqed.com/files/intro_KNN.pdf
- https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/
- https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

3.3 Foot Note:

The training data set, testing data set, full output, other .csv files, output images, reference materials can be found in this link below:

https://bit.ly/2w6fBr4