Predict Party Affiliation of Congressmen

GROUP MEMBERS:

SPANDAN MAITY, CEMK, REG NO- 151070110050 OF 2015-2016

DEBANJAN DAS, CEMK, REG NO- 151070110021 OF 2015-2016

Document sign date :Apr 23, 2018

CONTENTS

Document sign date :Apr 23, 2018

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my faculty **Mr. Titas Roy Chowdhury** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Document sign date :Apr 23, 2018

DESCRIPTION OF THE DATA

- The voting record of the current House of Representatives on 4 important votes in 2003.
- **Content :**
- State: Name of the States of the corresponding Congressmen.
- Name: Name of the Congressmen.
- Party: Name of party of the corresponding congressmen.
- Vote1: Vote given in 1st stage ("Y" for yes, "N" for vote against the topic, "-" for no vote)
- Vote2: Vote given in 2nd stage ("Y" for yes, "N" for vote against the topic, "-" for no vote)
- Vote3: Vote given in 3rd stage ("Y" for yes, "N" for vote against the topic, "-" for no vote)
- Vote4: Vote given in 4th stage ("Y" for yes, "N" for vote against the topic, "-" for no vote)
- **Goal :** To use Naive Bayes and nearest neighbors learning methods to predict the party affiliation of congressmen from their voting records.

Document sign date :Apr 23, 2018

- **COLUMN NAME DESCRIPTION:**

Columns	Types	Variable
State	object	Continuous
Name	object	Continuous
Party	object	Categorical(D , R)
Vote1	object	Categorical(Y , N , -)
Vote2	object	Categorical(Y , N , -)
Vote3	object	Categorical(Y , N , -)
Vote4	object	Categorical(Y , N , -)

- There are 434 rows and 7 columns in the Given Data.

Document sign date :Apr 23, 2018

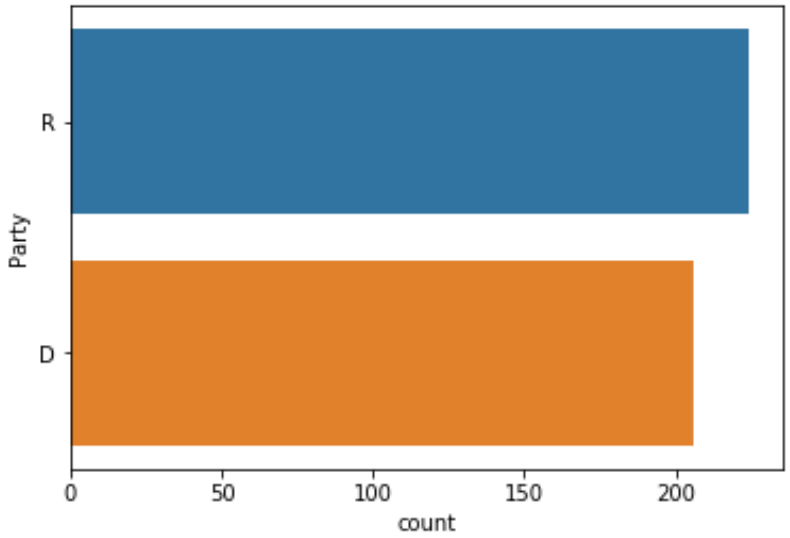
GRAPHICAL ANALYSIS:

Document sign date :Apr 23, 2018

PARTY:

Code:

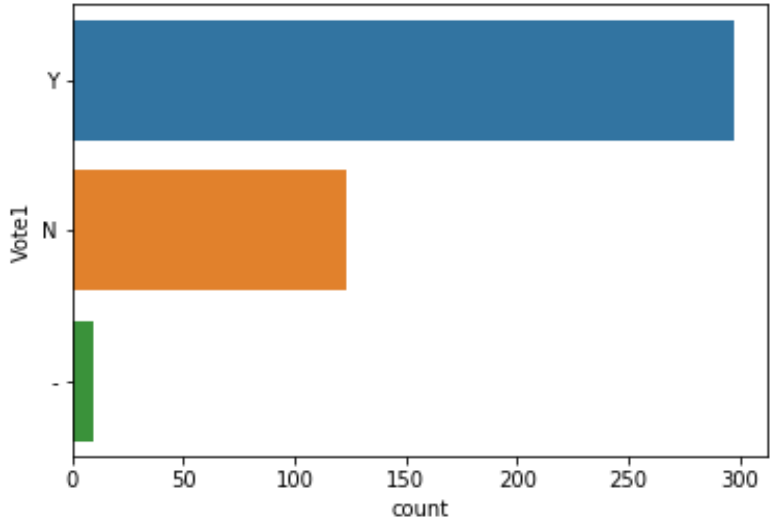
```
sns.countplot(y="Party",data=vote)
```



VOTE1:

Code:

```
sns.countplot(y="Vote1",data=vote)
```

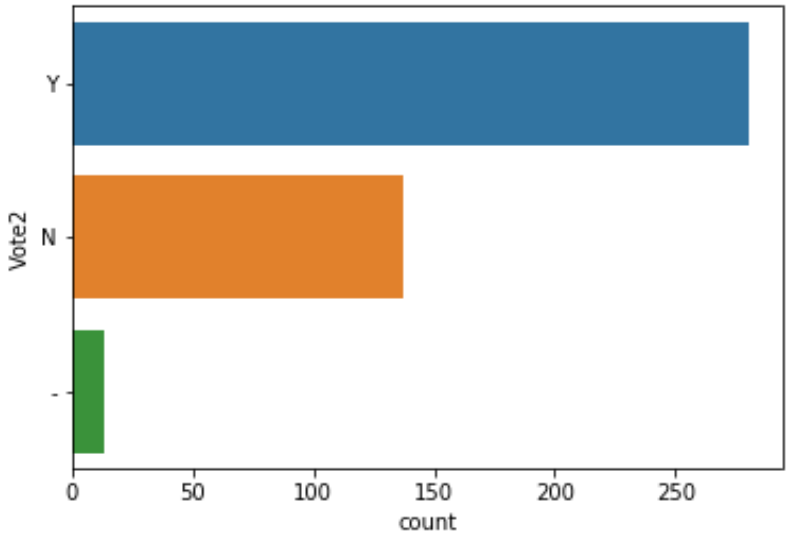


Document sign date :Apr 23, 2018

VOTE2:

Code:

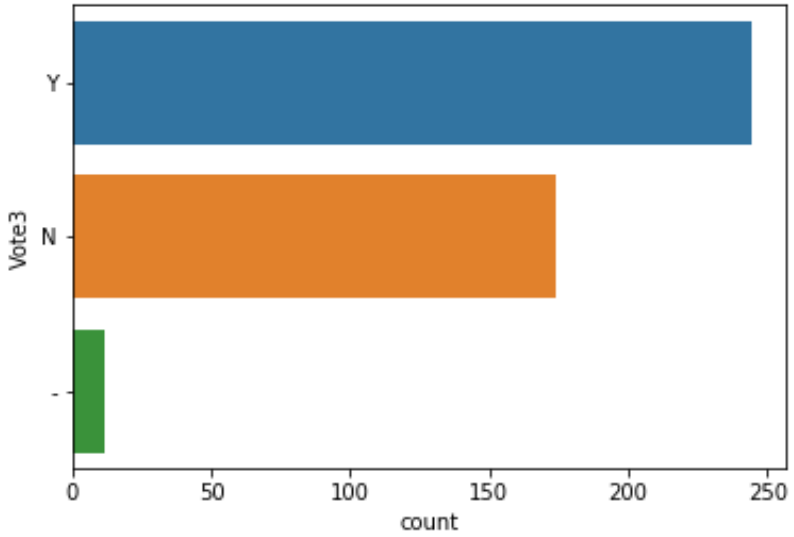
```
sns.countplot(y="Vote2",data=vote)
```



VOTE3:

Code:

```
sns.countplot(y="Vote3",data=vote)
```

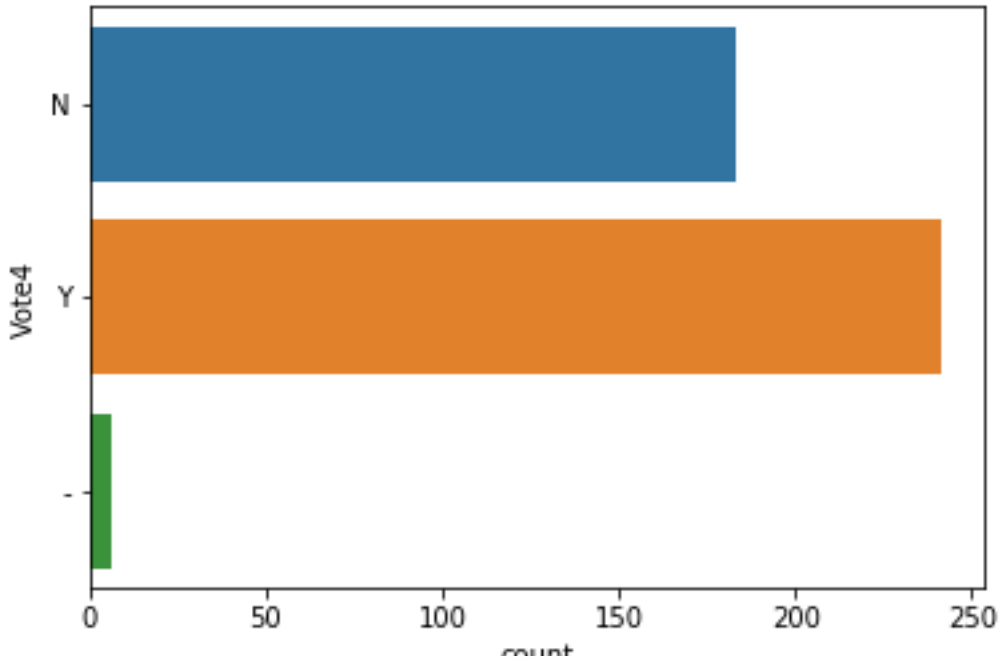


Document sign date :Apr 23, 2018

VOTE4:

Code:

```
sns.countplot(y="Vote4",data=vote)
```



DATA ANALYSIS

Document sign date :Apr 23, 2018

➢ NAIVE BAYES:

Naive Bayes is a simple but surprisingly powerful algorithm for predictive modeling.

In machine learning we are often interested in selecting the best hypothesis (h) given data (d).

In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d).

One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.

Bayes' Theorem is stated as:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

Where

- **P (h|d)** is the probability of hypothesis h given the data d. This is called the posterior probability.
- **P (d|h)** is the probability of data d given that the hypothesis h was true.
- **P (h)** is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.
- **P (d)** is the probability of the data (regardless of the hypothesis).

You can see that we are interested in calculating the posterior probability of P (h|d) from the prior probability p

Document sign date :Apr 23, 2018

After calculating the posterior probability for a number of different hypotheses, you can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis.
This can be written as:

$$MAP\ (h) = max\ (P\ (h|d))$$

Or

$$MAP\ (h) = max\ ((P\ (d|h)\ *\ P\ (h))\ /\ P\ (d))$$

Or

$$MAP\ (h) = max\ (P\ (d|h)\ *\ P\ (h))$$

The P(d) is a normalizing term which allows us to calculate the probability. We can drop it when we are interested in the most probable hypothesis as it is constant and only used to normalize.

Back to classification, if we have an even number of instances in each class in our training data, then the probability of each class (e.g. P (h)) will be equal. Again, this would be a constant term in our equation and we could drop it so that we end up with:

$$MAP\ (h) = max\ (P\ (d|h))$$

Naive Bayes Classifier:

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.

It is called *naive Bayes* or *idiot Bayes* because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value P(d1, d2, d3|h), they are assumed to be conditionally independent given the target value and calculated as P(d1|h) * P(d2|H) and so on.
This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

Representation Used By Naive Bayes Models:

The representation for naive Bayes is probabilities.

A list of probabilities are stored to file for a learned naive Bayes model. This includes:

- Class Probabilities: The probabilities of each class in the training dataset.
- Conditional Probabilities: The conditional probabilities of each input value given each class value.

Learn a Naive Bayes Model From Data:

Learning a naive Bayes model from training data is fast.

Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

Calculating Class Probabilities:

The class probabilities are simply the frequency of instances that belong to each class divided by the total number of instances.

For example in a binary classification the probability of an instance belonging to class 1 would be calculated as:

P(class=1) = count(class=1) / (count(class=0) + count(class=1))

In the simplest case each class would have the probability of 0.5 or 50% for a binary classification problem with the same number of instances in each class.

Calculating Conditional Probabilities:

The conditional probabilities are the frequency of each attribute value for a given class value divided by t ss value.

For example, if a "*weather*" attribute had the values "*sunny*" and "*rainy*" and the class attribute had the class values "*go-out*" and "*stay-home*", then the conditional probabilities of each weather value for each class value could be calculated as:

- P(weather=sunny|class=go-out) = count(instances with weather=sunny and class=go-out) / count(instances with class=go-out)
- P(weather=sunny|class=stay-home) = count(instances with weather=sunny and class=stay-home) / count(instances with class=stay-home)
- P(weather=rainy|class=go-out) = count(instances with weather=rainy and class=go-out) / count(instances with class=go-out)
- P(weather=rainy|class=stay-home) = count(instances with weather=rainy and class=stay-home) / count(instances with class=stay-home)

Make Predictions With a Naive Bayes Model:

Given a naive Bayes model, you can make predictions for new data using Bayes theorem.

$$MAP(h) = max(P(d|h) * P(h))$$

Using our example above, if we had a new instance with the *weather* of *sunny*, we can calculate:

go-out = P(weather=sunny|class=go-out) * P(class=go-out)

stay-home = P(weather=sunny|class=stay-home) * P(class=stay-home)

We can choose the class that has the largest calculated value. We can turn these values into probabilities by normalizing them as follows:

P(go-out|weather=sunny) = go-out / (go-out + stay-home)
P(stay-home|weather=sunny) = stay-home / (go-out + stay-home)

If we had more input variables we could extend the above example. For example, pretend we have a "*car*" attribute with the values "*working*" and "*broken*". We can multiply this probability into the equation.
For example below is the calculation for the "go-out" class label with the addition of the car input variable set to "working":

go-out = P(weather=sunny|class=go-out) * P(car=working|class=go-out) * P(class=go-out)

Gaussian Naive Bayes:

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution.

This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

Representation for Gaussian Naive Bayes:

Above, we calculated the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Learn a Gaussian Naive Bayes Model From Data:

This is as simple as calculating the mean and standard deviation values of each input variable (x) for each class value.

$$mean(x) = 1/n * sum(x)$$

Where n is the number of instances and x are the values for an input variable in your training data.

We can calculate the standard deviation using the following equation:

$$standard\ deviation(x) = sqrt(1/n * sum(xi-mean(x)^2))$$

This is the square root of the average squared difference of each value of x from the mean value of x, where n is the number of instances, sqrt() is the square root function, sum() is the sum function, xi is a specific value of the x variable for the i'th instance and mean(x) is described above, and ^2 is the square.

Document sign date :Apr 23, 2018

Make Predictions With a Gaussian Naive Bayes Model:

Probabilities of new x values are calculated using the Gaussian Probability Density Function(PDF). When making predictions these parameters can be plugged into the Gaussian PDF with a new input for the variable, and in return the Gaussian PDF will provide an estimate of the probability of that new input value for that class.

pdf(x, mean, sd) = (1 / (sqrt(2 * PI) * sd)) * exp(-((x-mean^2)/(2*sd^2)))

Where pdf(x) is the Gaussian PDF, sqrt() is the square root, mean and sd are the mean and standard deviation calculated above, PI is the numerical constant, exp() is the numerical constant e or Euler's number raised to power and x is the input value for the input variable.
We can then plug in the probabilities into the equation above to make predictions with real-valued inputs.

For example, adapting one of the above calculations with numerical values for weather and car:

go-out = P(pdf(weather)|class=go-out) * P(pdf(car)|class=go-out) * P(class=go-out)

Advantages / Disadvantages:

Naives Bayes is a stable algorithm. A small change in the the training data will not make a big change in the model.

- **Independent:**

All features are independent given the target Y.

The Naive Bayes assumption say that the probabilities of the different event (ie predictors attributes) are completely independent given the class value. (i.e., knowing the value of one attribute says nothing about the value of another if the class is known)

The attributes must be independent otherwise they will get to much influence on the accuracy. If an attribute are dependent with the target, it's the same that to determine the accuracy of only this attribute.

- **Equal:**

The event (ie predictors attributes) are equally important a priori.

If the accuracy remains exactly the same when an attribute is removed, it seems that this attribute is irrelevantto the outcome of the class (target).

When NAIVE BAYES is used?

- To mark an email as spam, or not spam ?
- Classify a news article about technology, politics, or sports ?
- Check a piece of text expressing positive emotions, or negativeemotions?

➢ K NEAREST NEIGHBORS:

“Nearest-neighbor” learning is also known as “Instance-based” learning.

K-Nearest Neighbors, or KNN, is a family of simple:

- classification
- and regression algorithms

based on Similarity (Distance) calculation between instances.

Nearest Neighbor implements rote learning. It's based on a local average calculation.

KNN Model Representation:

The model representation for KNN is the entire training dataset.

KNN has no model other than storing the entire dataset, so there is no learning required.

Efficient implementations can store the data using complex data structures like k-d trees to make look-up and matching of new patterns during prediction efficient.
Because the entire training dataset is stored, you may want to think carefully about the consistency of your training data. It might be a good idea to curate it, update it often as new data becomes available and remove erroneous and outlier data.

Making Predictions with KNN:

KNN makes predictions using the training dataset directly.

Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbors) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification this might be the mode (or most common) class value.

To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.
Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j.

$$EuclideanDistance(x, xi) = sqrt(sum((xj – xij)^2))$$

Euclidean is a good distance measure to use if the input variables are similar in type (e.g. all measured widths and heights). Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.).

The value for K can be found by algorithm tuning. It is a good idea to try many different values for K (e.g. values [illegible] for your problem.

The computational complexity of KNN increases with the size of the training dataset. For very large training sets, KNN can be made stochastic by taking a sample from the training dataset from which to calculate the K-most similar instances.

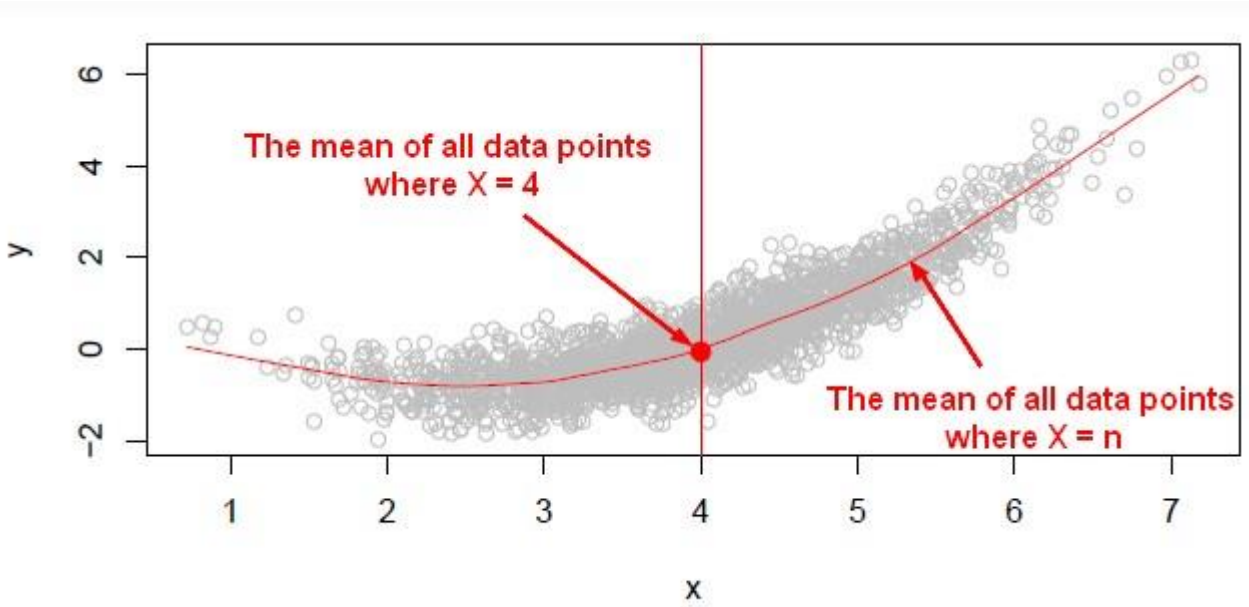
KNN has been around for a long time and has been very well studied. As such, different disciplines have different names for it, for example:

- **Instance-Based Learning**: The raw training instances are used to make predictions. As such KNN is often referred to as instance-based learning or a case-based learning (where each training instance is a case from the problem domain).
- **Lazy Learning**: No learning of the model is required and all of the work happens at the time a prediction is requested. As such, KNN is often referred to as a lazy learningalgorithm.
- **Non-Parametric**: KNN makes no assumptions about the functional form of the problem being solved. As such KNN is referred to as a non-parametric machine learning algorithm.
KNN can be used for regression and classification problems.

KNN for Regression:

When KNN is used for regression problems the prediction is based on the mean or the median of the K-most similar instances.

➢ Mean:

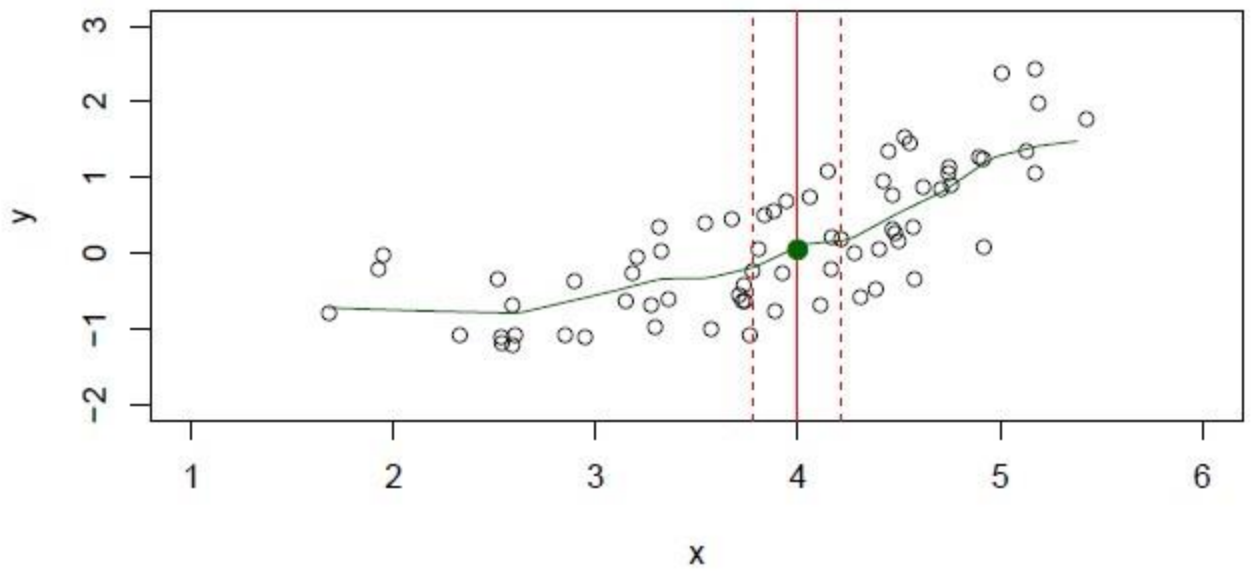


Of all Y value for a X.

➢ Neighborhood:

When they are too few data points for a X value in order to calculate the mean, the neighbourhood can be used.

Document sign date :Apr 23, 2018



Nearest neighbour averaging can be pretty good for a small number of variable (See accurate) and large N(in order to get enough point to calculate the average).

- **KNN for Classification:**

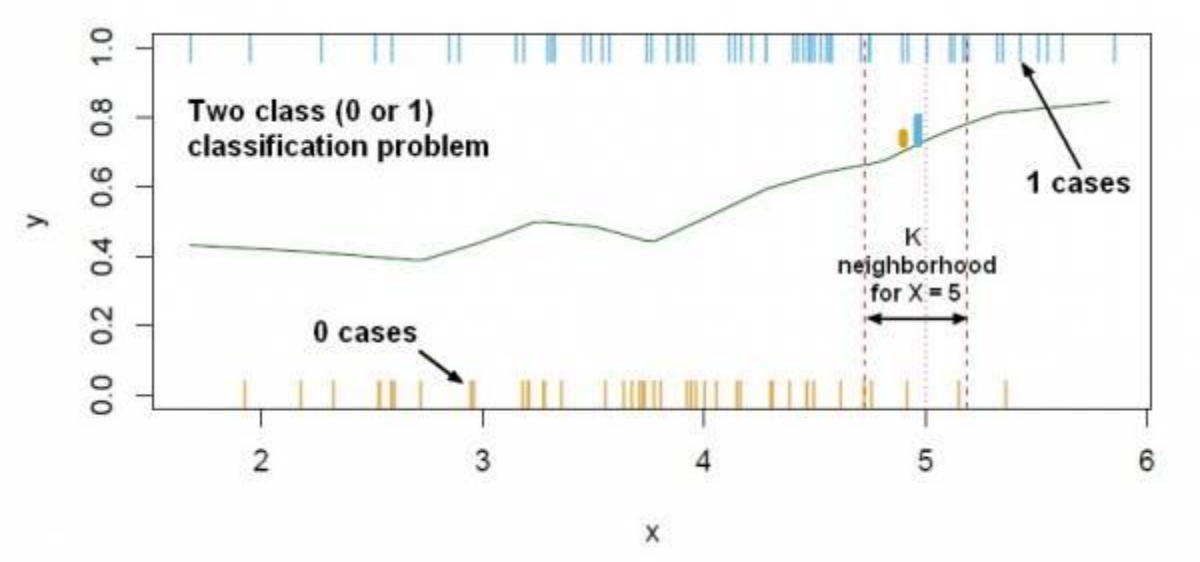
When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction.

Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance. For example, in a binary classificatior

p(class=0) = count(class=0) / (count(class=0)+count(class=1))

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.

Ties can be broken consistently by expanding K by 1 and looking at the class of the next most similar instance in the training dataset.

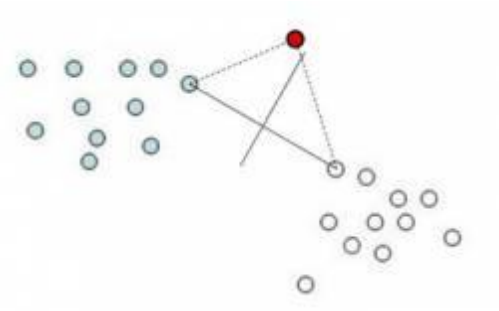


- **Decision boundary:**

The nearest neighbor methode produces a linear decision boundary. It's a little bit more complicate as it produces a piece-wise linear decision of the decision boundary with sometimes a bunch of little linear p

Document sign date :Apr 23, 2018

The perpendicular bisector of the line that joins the two closest points:



- **Noisy instances:**

Noisy instance are instance with a bad target class.

If the dataset is noisy , then by accident we might find an incorrectly classified training instance as the nearest one to our test instance.

You can guard against that by using:

- Majority vote over K nearest neighbours instances (k might be 3 or 4). You look for the 3 or 5 nearest neighbours and choose the majority class amongst those when classifying an unknown point. See K parameter
- Weight instances according to prediction accuracy
- Identification of reliable “prototypes” for each class

K Parameter:

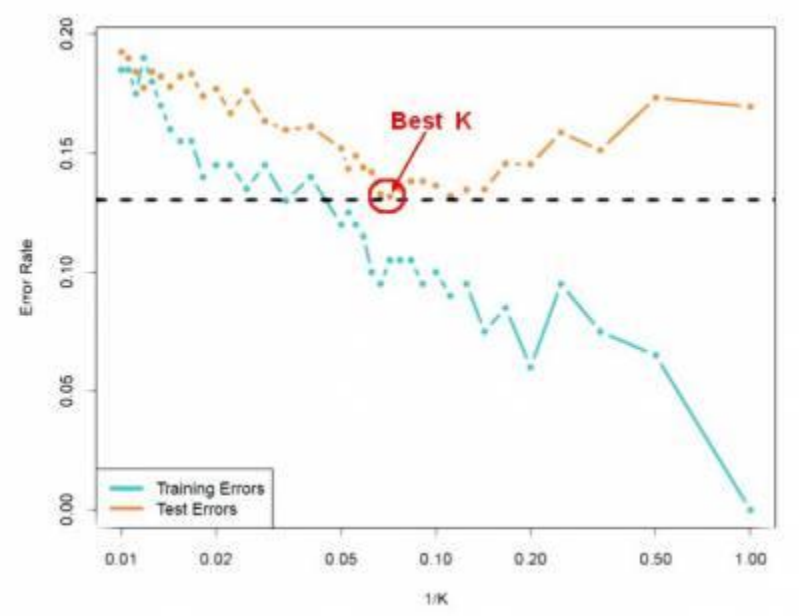
The K param ethod against noisy dataset.

➢ Best k value:

An obvious issue with k nearest neighbour is how to choose a suitable value for the number of nearest neighbours used.

- If it's too small, the method is susceptible to noise in the data.
- If it's too large, the decision is smeared out, covering too great an area of instance space

Weka uses cross-validation to select the best value.



The X axis shows the formula 1/K because since K is the number of neighbours, 1/k ⇐ 1.

Document sign date :Apr 23, 2018

- **With a K close to the size of the whole data set:**

If we set k to be an extreme value, close to the size of the whole data set, then we're taking the distance of the test instance to all of points in the dataset and averaging those which will probably gives us something close to the baseline accuracy.

Advantage/Inconvenient:

- **Curse of Dimensionality:**

KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large.

Each input variable can be considered a dimension of a p-dimensional input space. For example, if you had two input variables x1 and x2, the input space would be 2-dimensional.

As the number of dimensions increases the volume of the input space increases at an exponential rate.

In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the “Curse of Dimensionality“.

- **Slow:**

slow as you need to scan entire training data to make each prediction.

- **Checking wheather the data is noisy:**

By changing the k data set if the accuracy changes a lot between 1 and 5 k it's may be a noisy data set If the data set is noisy, the accuracy figures improves as k got little bit larger but then it would be starting to decrease again.

When KNN is used?

- k-NN is often used in search applications to look for “similar” items; that is, when task is some form of “find items similar to this one”.
- The biggest use case of k-NN search might be Recommender Systems. If a user likes a particular item, then k-NN can recommend similar items for them. To find similar items,the set of users who like each item are compared , if a similar set of users like two different items, then the items themselves are probably similar!

CODE

Document sign date :Apr 23, 2018

```
import pandas as pd
import os
import re
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn import neighbors

os.chdir ( "c:/datasets" )
reg=re.compile(r'\s\s+')
to_file=open("formatted_vote.csv",mode="w",encoding="utf-8")
with open("vote.csv",encoding="utf-8") as a_file:
    for a_line in a_file:
        formatted_rec=re.sub( r'\s\s+' , ',' , a_line )
        to_file.write(formatted_rec)
    to_file.clo
```

Document sign date :Apr 23, 2018

```
#we have to manually edit the column names
vote=pd.read_csv("formatted_vote.csv")
vote=vote[(vote.Party != "Y") & (vote.Party != "N")]

sns.countplot(y="Party",data=vote)
sns.countplot(y="Vote1",data=vote)
sns.countplot(y="Vote2",data=vote)
sns.countplot(y="Vote3",data=vote)
sns.countplot(y="Vote4",data=vote)

vote=vote.dropna(axis=0,how='any',inplace=False)
```

##

```
##########################################################################

def_r=(vote.Party == "R")
def_v1=(vote.Vote1 == "Y")
def_v2=(vote.Vote2 == "Y")
def_v3=(vote.Vote3 == "Y")
def_v4=(vote.Vote4 == "Y")


le = preprocessing.LabelEncoder()
bin_r=le.fit_transform(def_r)
bin_v1=le.fit_transform(def_v1)
bin_v2=le.fit_transform(def_v2)
bin_v3=le.fit_transform(def_v3)
bin_v4=le.fit_transform(def_v4)
```

```
df_r = pd.DataFrame(bin_r)
df_v1 = pd.DataFrame(bin_v1)
df_v2 = pd.DataFrame(bin_v2)
df_v3 = pd.DataFrame(bin_v3)
df_v4 = pd.DataFrame(bin_v4)

df1 = df_r.assign(df_v1=df_v1.values)
df1 = df1.assign(df_v2=df_v2.values)
df1 = df1.assign(df_v3=df_v3.values)
df1 = df1.assign(df_v4=df_v4.values)

df1.columns=['Party','Vote1','Vote2','Vote3','Vote4']

Xnb=df1[["Vote1","Vote2","Vote3","Vote4"]]
ynb=df1["Party"]
nb = GaussianNB()
Xtrain, Xtest ,test_size=.25,random_state=42)
```

```
nb.fit(Xtrain,ytrain)


performance_nb_test=nb.score(Xtest,ytest)
performance_nb_train=nb.score(Xtrain,ytrain)


print("")
print("Accuracy over train set in Naive Bayes: ",performance_nb_train)
print("Accuracy over test set in Naive Bayes: ",performance_nb_test)
print("")
#############################################################################
                    #knn accuracy
#############################################################################
vote=vote[vote.Vote1 != "-"]
vote=vote[vote.Vote2 != "-"]
vote=vote[vote.Vote3 != "-"]
vote=vote[vote.Vote4 != "-"]
```

```
def_r=(vote.Party == "R")
def_v1=(vote.Vote1 == "Y")
def_v2=(vote.Vote2 == "Y")
def_v3=(vote.Vote3 == "Y")
def_v4=(vote.Vote4 == "Y")


le = preprocessing.LabelEncoder()
bin_r=le.fit_transform(def_r)
bin_v1=le.fit_transform(def_v1)
bin_v2=le.fit_transform(def_v2)
bin_v3=le.fit_transform(def_v3)
bin_v4=le.fit_transform(def_v4)


df_r = pd.DataFrame(bin_r)
df_v1 = pd.D
```

Document sign date :Apr 23, 2018

```
df_v2 = pd.DataFrame(bin_v2)
df_v3 = pd.DataFrame(bin_v3)
df_v4 = pd.DataFrame(bin_v4)

df1 = df_r.assign(df_v1=df_v1.values)
df1 = df1.assign(df_v2=df_v2.values)
df1 = df1.assign(df_v3=df_v3.values)
df1 = df1.assign(df_v4=df_v4.values)

df1.columns=['Party','Vote1','Vote2','Vote3','Vote4']

X=df1[["Party","Vote1","Vote2","Vote3"]]
y=df1["Vote4"]

Xtrain, Xtest, ytrain, ytest=train_test_split(X,y,test_size=.20,random_state=42)
knn= neighbors.KNeighborsClassifier()
knn.fit(Xtrai
```

```
performance_knn_test=knn.score(Xtest,ytest)
performance_knn_train=knn.score(Xtrain,ytrain)

print("Accuracy over train set in KNN: ",performance_knn_train)
print("Accuracy over test set in KNN: ",performance_knn_test)

print("")

#############################################################################
                #predict party naive bayes
#############################################################################

os.chdir("c:/datasets")
vote=pd.read_csv("formatted_vote.csv")
vote=vote[(vote.Party != "Y") & (vote.Party != "N")]
```

```
vote=vote.dropna(axis=0,how='any',inplace=False)

x={'v1y':[0,0],'v1n':[0,0],'v2y':[0,0],'v2n':[0,0],'v3y':[0,0],'v3n':[0,0],'v4y':[0,0],'v4n':[0,0],'all':[0,0]}
index=["R","D"]
freqdf=pd.DataFrame(x,index)

vote1=vote[["Party","Vote1","Vote2","Vote3","Vote4"]]

for rows in vote1.iterrows():
    if rows[1][0]=="R":
        freqdf.loc[rows[1][0],"all"]=freqdf.loc[rows[1][0],"all"]+1
    if rows[1][0]=="D":
        freqdf.loc[rows[1][0],"all"]=freqdf.loc[rows[1][0],"all"]+1
    if rows[1][1]=="Y":
        freqdf.loc[rows[1][0],"v1y"]=freqdf.loc[rows[1][0],"v1y"]+1
    if rows[1]
```

Document sign date :Apr 23, 2018

```
        freqdf.loc[rows[1][0],"v1n"]=freqdf.loc[rows[1][0],"v1n"]+1
    if rows[1][2]=="Y":
        freqdf.loc[rows[1][0],"v2y"]=freqdf.loc[rows[1][0],"v2y"]+1
    if rows[1][2]=="N":
        freqdf.loc[rows[1][0],"v2n"]=freqdf.loc[rows[1][0],"v2n"]+1
    if rows[1][3]=="Y":
        freqdf.loc[rows[1][0],"v3y"]=freqdf.loc[rows[1][0],"v3y"]+1
    if rows[1][3]=="N":
        freqdf.loc[rows[1][0],"v3n"]=freqdf.loc[rows[1][0],"v3n"]+1
    if rows[1][4]=="Y":
        freqdf.loc[rows[1][0],"v4y"]=freqdf.loc[rows[1][0],"v4y"]+1
    if rows[1][4]=="N":
        freqdf.loc[rows[1][0],"v4n"]=freqdf.loc[rows[1][0],"v4n"]+1


x={'v1y':[0,0],'v1n':[0,0],'v2y':[0,0],'v2n':[0,0],'v3y':[0,0],'v3n':[0,0],'v4y':[0,0],'v4n':[0,0],'all':[0,0]}
index=["R","
```

```
probdf=pd.DataFrame(x,index)
r=0
for row in freqdf.iterrows():
    i=1
    while i<=8:
        probdf.iloc[r,i]=row[1][i]/row[1][0]
        i=i+1
    r=r+1
probdf.iloc[0,0]=freqdf.iloc[0,0]/(freqdf.iloc[0,0]+freqdf.iloc[1,0])
probdf.iloc[1,0]=freqdf.iloc[1,0]/(freqdf.iloc[0,0]+freqdf.iloc[1,0])

print("In Naive Bayes, party prediction for last four Congressmen :")

vote2=vote.tail(4).copy()
count_yes=0
count_no=0
for row in vo
```

```
v1=row[1][3]
v2=row[1][4]
v3=row[1][5]
v4=row[1][6]
prod_r=probdf.loc["R","all"]
prod_d=probdf.loc["D","all"]

if v1=="Y":
    prod_r=prod_r*probdf.loc["R","v1y"]
    prod_d=prod_d*probdf.loc["D","v1y"]
else:
    prod_r=prod_r*probdf.loc["R","v1n"]
    prod_d=prod_d*probdf.loc["D","v1n"]

if v2=="Y":
    prod_r=prod_r*probdf.loc["R","v2y"]
    prod_d
```

```
else:
    prod_r=prod_r*probdf.loc["R","v2n"]
    prod_d=prod_d*probdf.loc["D","v2n"]

if v3=="Y":
    prod_r=prod_r*probdf.loc["R","v3y"]
    prod_d=prod_d*probdf.loc["D","v3y"]
else:
    prod_r=prod_r*probdf.loc["R","v3n"]
    prod_d=prod_d*probdf.loc["D","v3n"]

if v4=="Y":
    prod_r=prod_r*probdf.loc["R","v4y"]
    prod_d=prod_d*probdf.loc["D","v4y"]
else:
    prod_r=prod_r*probdf.loc["R","v4n"]
    prod_d
```

```
    predicted = "R" if prod_r >= prod_d else "D"
    if predicted==row[1][2]:
        count_yes=count_yes+1
    else:
        count_no=count_no+1
    print("Predicted : ",predicted," Original : ",row[1][2])

accuracy_test=count_yes/(count_yes+count_no)
print("Accuracy is: ",accuracy_test)

print("")

#########################################################################
                #knn_Predict_Vote4
#########################################################################
```

```
vote2=df1.tail(4)
vote2=vote2[["Party","Vote1","Vote2","Vote3"]]

result2=knn.predict(vote2)

yresult2=df1.tail(4)
yresult2=yresult2["Vote4"]

accuracy=knn.score(vote2,yresult2)

result21= pd.DataFrame(result2)
df=result21.assign(yresult2=yresult2.values)
df.columns=['Predicted','Original']
df2=df
df2 = df2.replace( 1, "Y")
df2 = df2.replace( 0, "N")
```

```
print("In knn, The last four Congressmen Vote4 will be :")
print(df2)

print("Accuracy is: ",accuracy)
```

#End_of_code

Document sign date :Apr 23, 2018

RESULT:

Accuracy over train set in Naive Bayes: 0.850931677019

Accuracy over test set in Naive Bayes: 0.935185185185

Accuracy over train set in KNN: 0.708860759494

Accuracy over test set in KNN: 0.75

In Naive Bayes, party prediction for last four Congressmen:

Predicted: D Original: D

Predicted: D Original: D

Predicted: R Original: R

Predicted: D Original: R

Accuracy is: 0.75

In knn, the last four Congressmen Vote4 will be:

Predicted C

Document sign date :Apr 23, 2018

0 Y Y

1 Y Y

2 N N

3 N Y

Accuracy is: 0.75

- **FINAL COLUMN EXCLUSION:**

  No column has been eliminated.

- **EXPERIMENT:**

  The accuracy in Naïve Bayes over test set is 0.93 while in KNN, it is 0.75.

  So, here we can see Naïve Bayes has performed a better result.

FUTURE IMPROVEMENT

For future improvement of the model & Betterment of the scores ; Deep Learning , Nural Network & other advance methods may be introduced to it.

Document sign date :Apr 23, 2018

CERTIFICATE

This is to certify that **Mr. SPANDAN MAITY** of **College of Engineering and Management,Kolaghat** , registration number: **151070110050 OF 2015-2016**, has successfully completed a project on **"Machine Learing using Python"** under the guidance of **Mr. Titas Roy Chowdhury.**

_____

[Mr. Titas Roy Chowdhury]

Document sign date :Apr 23, 2018

CERTIFICATE

This is to certify that **Mr. DEBANJAN DAS** of **College of Engineering and Management,Kolaghat** , registration number: **151070110021 OF 2015-2016**, has successfully completed a project on **"Machine Learing using Python"** under the guidance of **Mr. Titas Roy Chowdhury.**

_____

[Mr. Titas Roy Chowdhury]

Document sign date :Apr 23, 2018