13. Naive Bayes

Bruce E. Shapiro

Getting Started in Machine Learning

Copyright (c) 2019. May not be distributed in any form without written permission.

Last revised: May 18, 2019

Bayes' Rule

$$P(c_k|\mathbf{x}) = \frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})}$$

$P(c_k)$ =Probability of class k

$P(\mathbf{x})$ =Probability of observation $\mathbf{x}$

$P(\mathbf{x}|k)$ =Probability of observation $\mathbf{x}$ assuming class c_k

Probabilistic method

$$\text{posterior} = P(c_k|\mathbf{x}) = \frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})} = \frac{\text{prior} \times \text{likelihood}}{\text{normalization}}$$

Probabilistic method

$$\text{posterior} = P(c_k|\mathbf{x}) = \frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})} = \frac{\text{prior} \times \text{likelihood}}{\text{normalization}}$$

- Denominator is fixed for any data point
- Make some theoretical assumption about likelihood
- Substitute data
- Find class assignments that maximize the posterior probability

Probabilistic method

$$\text{posterior} = P(c_k|\mathbf{x}) = \frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})} = \frac{\text{prior} \times \text{likelihood}}{\text{normalization}}$$

- Denominator is fixed for any data point
- Make some theoretical assumption about likelihood
- Substitute data
- Find class assignments that maximize the posterior probability

$$y' = \arg \max_{c_k} \Big[P(c_k|\mathbf{x}) \Big] = \arg \max_{c_k} \left[\frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})} \right]$$

$$= \arg \max_{c_k} \Big[P(c_k)P(\mathbf{x}|c_k) \Big]$$

Probabilistic Classification Algorithm

- Make class assignments so that

$$y' = \arg \max_{c_k} \left[P(c_k) P(\mathbf{x}|c_k) \right]$$

- Initialize $P(c_k)$ based on observed class distribution
- Assume some distribution (multivariate normal) for $P(\mathbf{x}|c_k)$
- Use iterative optimization algorithm to find class assignments k that maximizes $P(c_k)P(\mathbf{x}|c_k)$ for all k

Probabilistic Methods

- Naive Bayes: assumes statistically independent features
- KNN: based on K-neighbors
- Discriminant Methods: normal distributions with same σ for all classes
  - LDA - Linear: all features have same covariance
  - QDA - Quadratic: features may have different covariances

Naive Bayes'

Assumes features are independent: $P(\mathbf{x}|c_k) = \prod_{j=1}^{d} P(x_j|c_k)$

Naive Bayes'

Assumes features are independent: $P(\mathbf{x}|c_k) = \prod_{j=1}^{d} P(x_j|c_k)$

Therefore $P(c_k|\mathbf{x}) = \frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})} = \frac{P(c_k)}{P(\mathbf{x})} \prod_{j=1}^{d} P(x_j|c_k)$

Naive Bayes'

Assumes features are independent: $P(\mathbf{x}|c_k) = \prod_{j=1}^{d} P(x_j|c_k)$

Therefore $P(c_k|\mathbf{x}) = \frac{P(c_k)P(\mathbf{x}|c_k)}{P(\mathbf{x})} = \frac{P(c_k)}{P(\mathbf{x})} \prod_{j=1}^{d} P(x_j|c_k)$

Algorithm becomes: $y' = \arg \max_{c_k} \left[P(c_k) \prod_{j=1}^{d} P(x_j|c_k) \right]$

Example: Predict Car Cylinders (1/14)

Read Libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

Read auto-mpg data file and drop lines with no data.

```
data=pd.read_fwf("https://archive.ics.uci.edu/ml/
   machine-learning-databases/auto-mpg/auto-mpg.data",
   header=None,na_values="?")
data.columns=("mpg","cyl","displ","hp","weight",
   "accel","model","origin","carname")
data = data.dropna(axis=0)
```

Example: Predict Car Cylinders (2/14)

Create a 3-class file (extract 4,6,8 cylinder data):

```
cardata=np.array(data[["cyl","mpg","accel"]])
cars=np.array([line for line in cardata
   if line[0] in [4,6,8]])

X=cars[:,1:] # columns 1 and 2, mpg and accel

cylinders=np.unique(cars[:,0])
Y=np.array([float(cylinders.tolist().index(j))
  for j in cars[:,0]]) # 0, 1, 2 for 4, 6, 8 cylinders
```

Example: Predict Car Cylinders (2/14)

Create a 3-class file (extract 4,6,8 cylinder data):

```
cardata=np.array(data[["cyl","mpg","accel"]])
cars=np.array([line for line in cardata
   if line[0] in [4,6,8]])

X=cars[:,1:] # columns 1 and 2, mpg and accel

cylinders=np.unique(cars[:,0])
Y=np.array([float(cylinders.tolist().index(j))
  for j in cars[:,0]]) # 0, 1, 2 for 4, 6, 8 cylinders
```

Imports:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
```

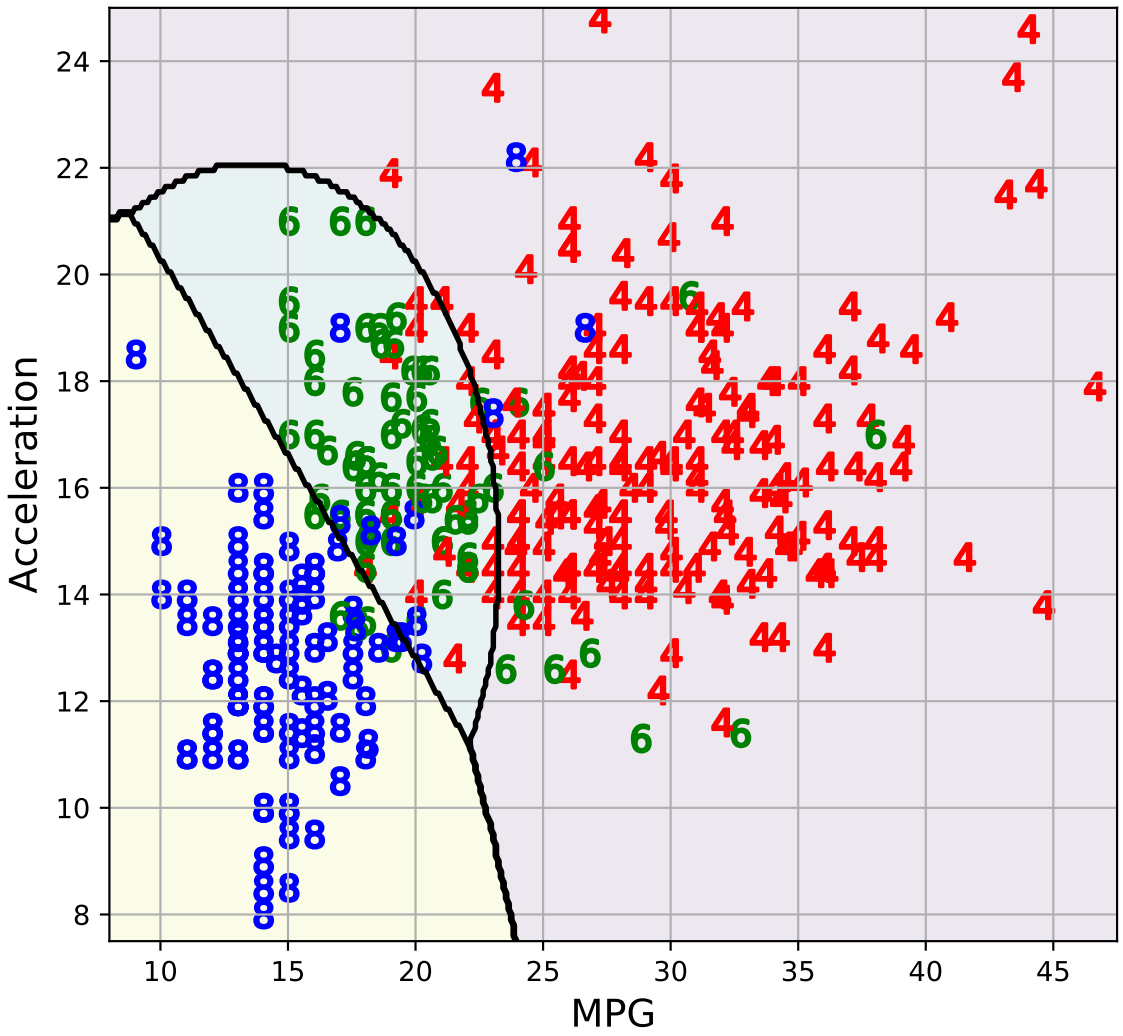
Example: Predict Car Cylinders (3/14)

100 Train/Test Splits and Final Confusion Matrix

```
errs=[]; nsplits = 100
for split in range(nsplits):
    XTRAIN, XTEST, YTRAIN, YTEST=train_test_split(X,Y)
    gnb=GaussianNB()
    gnb.fit(XTRAIN,YTRAIN)
    YP=gnb.predict(XTEST)
    errs.append(1-accuracy_score(YTEST,YP))
print("%d Splits: Mean Error=%7.6f +/- %7.6f (95%%)"\
      %(nsplits, np.mean(errs),1.96*np.std(errs)))
print(confusion_matrix(YTEST,YP))
```

```
100 Splits: Mean Error=0.120928 +/- 0.064840 (95%)
[[42  2  0]
 [ 4 15  0]
 [ 1  5 28]]
```

Example: Predict Car Cylinders (4/14)



Example: Predict Car Cylinders (5/14)

Add more features:

```
X=np.array(data[["mpg","displ","hp","accel"]])
Y=np.array(data["cyl"])
unis=np.unique(Y).tolist()
print(unis)
Y=np.array([unis.index(j) for j in Y])
```

```
[3, 4, 5, 6, 8]
```

Example: Predict Car Cylinders (6/14)

```
errs=[]
for j in range(100): # 100 iterations, 4 features, 5 classes
    XTRAIN,XTEST,YTRAIN,YTEST=train_test_split(X,Y)
    gnb=GaussianNB()
    gnb.fit(XTRAIN,YTRAIN)
    YP=gnb.predict(XTEST)
    errs.append(1-accuracy_score(YP,YTEST))
print("%d Splits: Mean Error=%7.6f +/- %7.6f (95%%)"\
      %(nsplits, np.mean(errs),1.96*np.std(errs)))
print("Last confusion matrix:")
print(confusion_matrix(YP, YTEST))
```

```
100 Splits: Mean Error=0.043673 +/- 0.037956 (95%)
Last confusion matrix:
[[ 1  0  0  0  0]
 [ 0 46  1  1  0]
 [ 0  0  0  0  0]
 [ 0  0  0 20  0]
 [ 0  0  0  1 28]]
```

One-Hot Encoding

- Encode *categorical* variables numerically
- Suppose cars are sold in the following colors:

  white, black, silver, red, blue
- Can't replace **`color`** with **`colorn`**, as it introduces an artificial ordering and a bias.

name	**`color`**	**`colorn`**
car 1	red	4
car 2	white	1
car 3	black	2
car 4	white	1
car 5	blue	5
car 6	silver	3
car 7	black	2
⋮		

One-Hot Encoding

- Encode *categorical* variables numerically
- Suppose cars are sold in the following colors:

  white, black, silver, red, blue
- Instead: introduce a new variable for each color; each of these variables is orthogonal and there is no artificial ordering

name	**color**	**white**	**black**	**silver**	**red**	**blue**
car 1	red	0	0	0	1	0
car 2	white	1	0	0	0	0
car 3	black	0	1	0	0	0
car 4	white	1	0	0	0	0
car 5	blue	0	0	0	0	1
car 6	silver	0	0	1	0	0
car 7	black	0	1	0	0	0
⋮						

Large: Create new data frame with Origin

```
DF=data[["mpg","displ","hp","accel","origin"]]
DF[10:21]  # print lines 10 through 20
```

	mpg	displ	hp	accel	origin
10	15.0	383.0	170.0	10.0	1
11	14.0	340.0	160.0	8.0	1
12	15.0	400.0	150.0	9.5	1
13	14.0	455.0	225.0	10.0	1
14	24.0	113.0	95.0	15.0	3
15	22.0	198.0	95.0	15.5	1
16	18.0	199.0	97.0	15.5	1
17	21.0	200.0	85.0	16.0	1
18	27.0	97.0	88.0	14.5	3
19	26.0	97.0	46.0	20.5	2
20	25.0	110.0	87.0	17.5	2

Example: Predict Car Cylinders (7/14)

Convert **origin** using "1-hot" encoding with **get_dummies**

- Data Set has 1,2,3 for American, European, Asian
- Change this to three features (1,0,0) (0,1,0) and (0,0,1)

```
DF=pd.get_dummies(DF,columns=["origin"])
DF[10:21]
```

mpg	displ	hp	accel	origin_1	origin_2	origin_3
15.0	383.0	170.0	10.0	1	0	0
14.0	340.0	160.0	8.0	1	0	0
15.0	400.0	150.0	9.5	1	0	0
14.0	455.0	225.0	10.0	1	0	0
24.0	113.0	95.0	15.0	0	0	1
22.0	198.0	95.0	15.5	1	0	0
18.0	199.0	97.0	15.5	1	0	0
21.0	200.0	85.0	16.0	1	0	0
27.0	97.0	88.0	14.5	0	0	1
26.0	97.0	46.0	20.5	0	1	0
25.0	110.0	87.0	17.5	0	1	0

```
X=np.array(DF)
print(X[10:21]) # print lines 10-20 of feature matrix
```

```
[[ 15.  383.  170.   10.    1.    0.    0. ]
 [ 14.  340.  160.    8.    1.    0.    0. ]
 [ 15.  400.  150.    9.5   1.    0.    0. ]
 [ 14.  455.  225.   10.    1.    0.    0. ]
 [ 24.  113.   95.   15.    0.    0.    1. ]
 [ 22.  198.   95.   15.5   1.    0.    0. ]
 [ 18.  199.   97.   15.5   1.    0.    0. ]
 [ 21.  200.   85.   16.    1.    0.    0. ]
 [ 27.   97.   88.   14.5   0.    0.    1. ]
 [ 26.   97.   46.   20.5   0.    1.    0. ]
 [ 25.  110.   87.   17.5   0.    1.    0. ]]
```

Example: Predict Car Cylinders (8/14)

```
Y=np.array(data["cyl"])
unis=np.unique(Y).tolist()
Y=np.array([unis.index(j) for j in Y])
```

Now there are 7 features, still 5 classes

```
errs=[]
for j in range(100):
    gnb=GaussianNB()
    XTRAIN, XTEST, YTRAIN, YTEST=train_test_split(X,Y)
    gnb.fit(XTRAIN,YTRAIN)
    YP=gnb.predict(XTEST)
    errs.append(1-accuracy_score(YTEST,YP))
print("%d Splits: Mean Error=%7.6f +/- %7.6f (95%%)"\
      %(nsplits, np.mean(errs),1.96*np.std(errs)))
print("Last confusion matrix:")
print(confusion_matrix(YP,YTEST))
```

Example: Predict Car Cylinders (9/14)

```
100 Splits: Mean Error=0.356531 +/- 0.121188 (95%)
Last confusion matrix:
[[ 0  0  0  0  0]
 [ 1 38  1  2  0]
 [ 0  6  0  0  0]
 [ 0  4  0  2  1]
 [ 0  0  0 16 27]]
```

- Much higher error
- Based on the single confusion matrix, maybe difficult to distinguish between 6 and 8 cylinder classes now

Example: Predict Car Cylinders (10/14)

Possible error sources

- Features are not independent
- Curse of dimensionality
- Over-fitting of data
- Data is not normally distributed

Example: Predict Car Cylinders (11/14)

Independent Component Analysis

```
from sklearn.decomposition import FastICA
nsplits=100
for ncomp in range(2,8):
    ICA=FastICA(n_components=ncomp)
    ICA.fit(X)
    F=ICA.transform(X)
    errs=[]
    for j in range(nsplits):
        gnb=GaussianNB()
        XTRAIN, XTEST, YTRAIN, YTEST=train_test_split(F,Y)
        gnb.fit(XTRAIN,YTRAIN)
        YP=gnb.predict(XTEST)
        errs.append(1-accuracy_score(YTEST,YP))
    print("%d Splits, %d Components: Mean Error=%7.6f +/- %7.6
      %(nsplits,ncomp, np.mean(errs),1.96*np.std(errs)))
```

Example: Predict Car Cylinders (12/14)

Result of ICA

```
100 Splits, 2 Components: Mean Error=0.086837 +/- 0.053103 (95%)
100 Splits, 3 Components: Mean Error=0.110816 +/- 0.053593 (95%)
100 Splits, 4 Components: Mean Error=0.106735 +/- 0.059358 (95%)
100 Splits, 5 Components: Mean Error=0.135204 +/- 0.064831 (95%)
100 Splits, 6 Components: Mean Error=0.116633 +/- 0.053610 (95%)
100 Splits, 7 Components: Mean Error=0.293673 +/- 0.103637 (95%)
```

Example: Predict Car Cylinders (13/14)

High Kurtosis is Pointy, Low Kurtosis is Flat

```
from scipy.stats import kurtosis
kurtosis(X, fisher=False)
```

```
array([2.47529742, 2.21630802, 3.67282189,
       3.42332033, 1.26666667,
       3.97458243, 3.21442148])
```

Example: Predict Car Cylinders (14/14)

Dimensionality of the Data

```
from sklearn.decomposition import PCA
pca=PCA(n_components=7)
pca.fit(X)
print(pca.explained_variance_ratio_)
```

```
[9.77103869e-01 2.09376473e-02 1.64085614e-03
 2.88975479e-04 1.44868553e-05 1.41656401e-05
 6.19652520e-37]
```

References

1. MPG data from: Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository `http://archive.ics.uci.edu/ml`. Irvine, CA: University of California, School of Information and Computer Science.