naive-bayes-in-depth

January 10, 2024

1 Naive Bayes In Depth By Amritesh Kumar - Neuraldemy

This notebook is part of Neuraldemy tutorial on naive Bayes.

1.1 Gaussian Naive Bayes

Assumptions

- The features are continuous.
- Each feature follows a Gaussian (Normal) distribution.
- Features are conditionally independent given the class label.

Based on the derivation given in the notes. We can apply the same for Gaussian distribution.

Given Gaussian Naive Bayes model with MLE:

$$P(C_k|\mathbf{X}) \propto P(C_k)\prod_{i=1}^{d}\frac{1}{\sqrt{2\pi\hat{\sigma}_{k,i}^2}}\exp\left(-\frac{(X_i-\hat{\mu}_{k,i})^2}{2\hat{\sigma}_{k,i}^2}\right)$$

Apply the log transformation:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) - \frac{1}{2}\sum_{i=1}^{d}\left(\log(2\pi\hat{\sigma}_{k,i}^2) + \frac{(X_i-\hat{\mu}_{k,i})^2}{\hat{\sigma}_{k,i}^2}\right)$$

Introduce Maximum Likelihood Estimation (MLE) for Parameters:

$$\hat{\mu}_{k,i} = \frac{\sum_{j=1}^{N_k} X_{i,j}}{N_k}$$

$$\hat{\sigma}_{k,i}^2 = \frac{\sum_{j=1}^{N_k} (X_{i,j}-\hat{\mu}_{k,i})^2}{N_k}$$

Substitute MLE estimates into the log-likelihood expression:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) - \frac{1}{2}\sum_{i=1}^{d}\left(\log(2\pi\hat{\sigma}_{k,i}^2) + \frac{(X_i-\hat{\mu}_{k,i})^2}{\hat{\sigma}_{k,i}^2}\right)$$

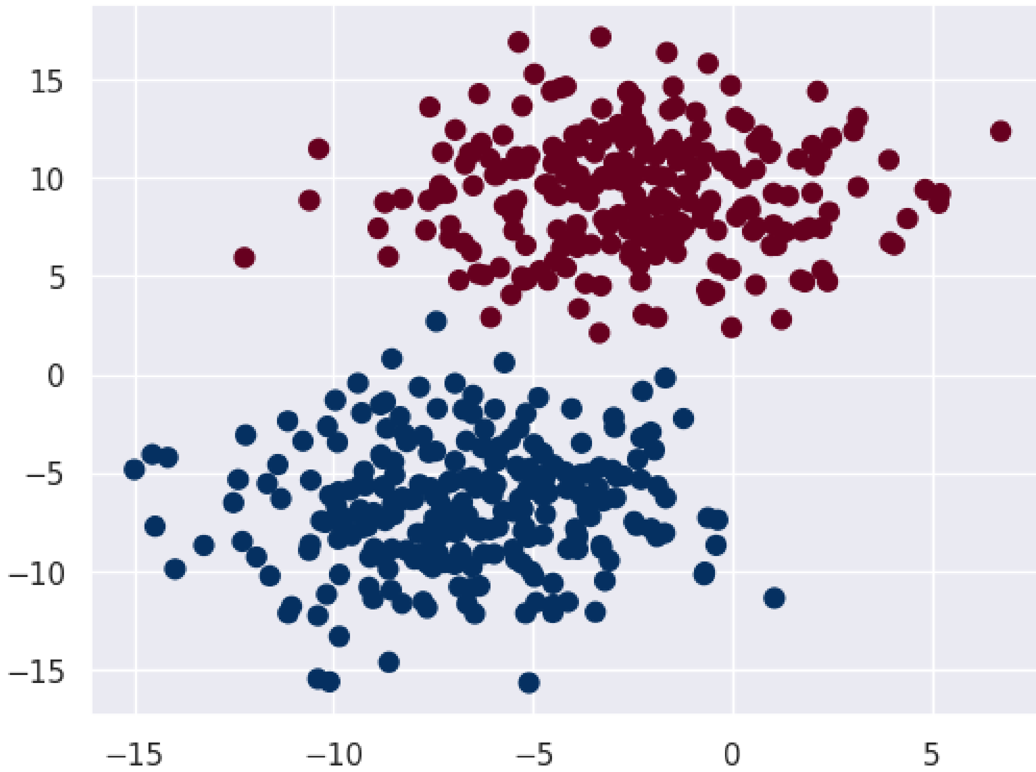
Combine with Prior Probabilities:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) - \frac{1}{2}\sum_{i=1}^{d}\left(\log(2\pi\hat{\sigma}_{k,i}^2) + \frac{(X_i-\hat{\mu}_{k,i})^2}{\hat{\sigma}_{k,i}^2}\right)$$

[16]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
import seaborn as sns; sns.set()
from sklearn.datasets import make_blobs

# create dataset
X, y = make_blobs(500, 4, centers=2, random_state=42, cluster_std=3)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu');
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 42)
```



[17]:
```
# Initialize and train the Multinomial Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)
```

[17]: GaussianNB()

[21]:
```
# Make prediction on test data
y_pred = model.predict(X_test)
y_pred
```

[21]:
```
array([1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1])
```

[20]:
```
accuracy_score(y_test, y_pred)
```

[20]: 1.0

1.2 Multinomial Naive Bayes

The Multinomial distribution models the probability of observing counts in multiple categories. It is an extension of the Binomial distribution to more than two categories.

Probability Mass Function (PMF):

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k) = \frac{n!}{x_1!x_2!\ldots x_k!} p_1^{x_1} p_2^{x_2} \ldots p_k^{x_k}$$

Assumptions:
- There are k categories.
- Each observation falls into one of these k categories.
- The categories are mutually exclusive.

Consider an experiment where a fair six-sided die is rolled three times. Let X_1, X_2, X_3 be the counts of outcomes 1, 2, and 3, respectively. The Multinomial distribution can model this scenario.

Probability Mass Function (PMF) for the example:

$$P(X_1 = x_1, X_2 = x_2, X_3 = x_3) = \frac{3!}{x_1!x_2!x_3!} \left(\frac{1}{6}\right)^{x_1} \left(\frac{1}{6}\right)^{x_2} \left(\frac{1}{6}\right)^{x_3}$$

Here, $n = 3$ (number of trials), $k = 3$ (number of categories), and $p_1 = p_2 = p_3 = \frac{1}{6}$ (probability of each category).

Given Naive Multinomial Bayes model with Laplace smoothing:

$$P(C_k|\mathbf{X}) \propto P(C_k)\prod_{i=1}^{d} p_{ki}^{x_i}$$

Apply the log transformation:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) + \sum_{i=1}^{d} x_i \cdot \log(p_{ki})$$

Introduce smoothing for Parameters:

$$\hat{p}_{ki} = \frac{N_{ki} + \alpha}{N_k + \alpha n}$$

Setting = 1 is called Laplace smoothing, while is called < 1 Lidstone smoothing

Substitute smoothed estimates into the log-likelihood expression:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) + \sum_{i=1}^{d} x_i \cdot \log(\hat{p}_{ki})$$

Combine with Prior Probabilities:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) + \sum_{i=1}^{d} x_i \cdot \log(\hat{p}_{ki})$$

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice).

[25]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

# Load the 20 Newsgroups dataset
newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers',␣
↪'quotes'))

# Split the dataset into training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(newsgroups.data, newsgroups.
 ↪target, test_size=0.2, random_state=42)

# Convert the text data to a bag-of-words representation
vectorizer = CountVectorizer(stop_words='english')
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
```

[28]:
```python
X_train[:5]
```

[28]:
```
["#\n# I've gotten very few posts on this group in the last couple days.  (I\n#
recently added it to my feed list.)  Is it just me, or is this group\n# near
death?\n#\n\nSeen from the mailing list side, I'm getting about the right amount
of\ntraffic.\n\nPatrick L. Mahan\n\n--- TGV Window Washer
------------------------------- Mahan@TGV.COM ---------\n\nWaking a person
unnecessarily should not be considered  - Lazarus Long\na capital crime.  For a
first offense, that is            From the Notebooks of\n\t\t\t\t\t\t\t  Lazarus
Long\n\nPatrick L. Mahan\n\n--- TGV Window Washer
------------------------------- Mahan@TGV.COM ---------",
 "Interesting.  I'd fight the ticket.  First off, there's a 50/50 chance\nthe
cop won't show up.  Secondly, if he does show up, you should point\nout that he
lied (purgered) on the ticket.  Why 70+?  I beleive that if\nyo're charged with
going more than 15mph that the posted speed it's a\nmore severe ticket.  You
couldn't have p[ossibly been going 70+, right?!\n",
 "I remember as a kid visiting my relatives on Kauai, and one of the
things\nthat really frightened me was centipedes.  I'd been told they were
poisonous\nand infrequently one would pop up and scare the heck out of me.
Once\none came out of the vacuum cleaner and it seemed like it was at least a
foot\nlong and moving at 35 miles an hour!\n",
 "\n\nIt can be painless, so it isn't cruel.  And, it has occurred
frequently\nsince the dawn of time, so it is hardly unusual.\n\n\nBut, innocents
die due to many causes.  Why have you singled out\naccidental or false execution
as the one to take issue with?",
 'the owners are whining about baseball not being popular among a\nlarge enough
portion of the population, and have suggested various\n"remedies", such as
shortening the game or trying to convince us
that\n"smoke\'embake\'emdominatebysheerintimidation" is an accurate
description\nof what is, essentially, a laid-back game.\n\nforget those lame
ideas.  here is my new and exciting two-point plan to\ngenerate interest in
baseball among the masses.\n\npoint one: sex.\npoint two: violence.\n\nlet\'s
face it, sex and violence are the only things that sell in\namerica.  here\'s
how we can implement them in the game:\n\nsex: cheerleaders, cheerleaders, and
more cheerleaders.  dancing on top\n     of the dugouts.  bringing hot dogs to
the umps during the seventh\n     inning stretch.  running up and down the
stands.  (the south bend\n     white sox actually do this).\n\nviolence:
baseball players are such utter wuss boys.  the pitcher beans\n     the batter,
and both benches empty in what is called a "bench-clearing\n     brawl".
```

```
EVERYBODY JUST STANDS THERE AND LOOKS AT EACH OTHER. stand,\n     stand, stand.
look, look, look.  ho, hum.  then, the bullpens\n     come running in.  when
they reach the "fight", they just stand\n     there, too.\n\n     anybody coming
off the bench who does not throw at least one punch\n     should be suspended
and fined.  further, the bullpens should fight\n     it out in the outfield, so
as not to waste time and energy running\n     to the infield.\n\nfootball: sex,
violence.\nbasketball: sex, violence.\nhockey: violence.\nbaseball: "da pastime
of da nayshun!" - yawn.']
```

[29]:
```python
X_train_counts
```

[29]:
```
<15076x111275 sparse matrix of type '<class 'numpy.int64'>'
        with 965357 stored elements in Compressed Sparse Row format>
```

[31]:
```python
# Initialize and train the Multinomial Naive Bayes model
clf = MultinomialNB()
clf.fit(X_train_counts, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test_counts)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Display classification report
print(classification_report(y_test, y_pred, target_names=newsgroups.
 ↪target_names))
```

```
Accuracy: 67.53%
                          precision    recall  f1-score   support

             alt.atheism       0.63      0.41      0.50       151
           comp.graphics       0.49      0.75      0.59       202
 comp.os.ms-windows.misc       0.69      0.05      0.09       195
comp.sys.ibm.pc.hardware       0.51      0.75      0.61       183
   comp.sys.mac.hardware       0.82      0.64      0.72       205
          comp.windows.x       0.69      0.81      0.74       215
            misc.forsale       0.87      0.65      0.75       193
               rec.autos       0.85      0.70      0.77       196
         rec.motorcycles       0.51      0.68      0.58       168
      rec.sport.baseball       0.94      0.78      0.85       211
        rec.sport.hockey       0.89      0.87      0.88       198
               sci.crypt       0.68      0.79      0.73       201
         sci.electronics       0.81      0.57      0.67       202
                 sci.med       0.83      0.85      0.84       194
               sci.space       0.77      0.78      0.78       189
```

```
soc.religion.christian       0.49      0.93      0.64       202
    talk.politics.guns       0.73      0.68      0.70       188
 talk.politics.mideast       0.65      0.81      0.72       182
    talk.politics.misc       0.51      0.66      0.57       159
    talk.religion.misc       0.80      0.09      0.16       136

              accuracy                           0.68      3770
             macro avg       0.71      0.66      0.65      3770
          weighted avg       0.71      0.68      0.66      3770
```

`ComplementNB` implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the model's weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks.

[33]:
```python
from sklearn.naive_bayes import ComplementNB

model = ComplementNB()
model.fit(X_train_counts, y_train)

y_pred = model.predict(X_test_counts)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 71.91%
```

As you can see the model performance has improved. It was designed to correct the "severe assumptions" made by the standard Multinomial Naive Bayes classifier. It is particularly suited for imbalanced data sets.

1.3 Bernoulli Naive Bayes

`BernoulliNB` implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a BernoulliNB instance may binarize its input (depending on the binarize parameter).

Given Bernoulli Naive Bayes model:

$$P(C_k|\mathbf{X}) \propto P(C_k) \prod_{i=1}^{d} P(x_i|C_k)^{x_i} (1 - P(x_i|C_k))^{(1-x_i)}$$

Apply the log transformation:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) + \sum_{i=1}^{d} (x_i \cdot \log P(x_i|C_k) + (1 - x_i) \cdot \log(1 - P(x_i|C_k)))$$

Introduce Maximum Likelihood Estimation (MLE) for Parameters:

$$P(x_i|C_k) = \frac{\sum_{j=1}^{N_k} x_{i,j}}{N_k}$$

Substitute MLE estimates into the log-likelihood expression:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) + \sum_{i=1}^{d} \left(x_i \cdot \log \left(\frac{\sum_{j=1}^{N_k} x_{i,j}}{N_k} \right) + (1 - x_i) \cdot \log \left(1 - \frac{\sum_{j=1}^{N_k} x_{i,j}}{N_k} \right) \right)$$

Combine with Prior Probabilities:

$$\log P(C_k|\mathbf{X}) = \log P(C_k) + \sum_{i=1}^{d} \left(x_i \cdot \log \left(\frac{\sum_{j=1}^{N_k} x_{i,j}}{N_k} \right) + (1 - x_i) \cdot \log \left(1 - \frac{\sum_{j=1}^{N_k} x_{i,j}}{N_k} \right) \right)$$

[35]:
```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import BernoulliNB

# Set random seed for reproducibility
np.random.seed(42)

# Number of instances
n_instances = 1000

# Number of features
n_features = 5
```

```
# Generate binary dataset
data = np.random.choice([0, 1], size=(n_instances, n_features), p=[0.5, 0.5])
labels = np.random.choice([0, 1], size=n_instances)

# Create a DataFrame for better visualization
columns = [f'Feature_{i+1}' for i in range(n_features)]
df = pd.DataFrame(data, columns=columns)
df['Label'] = labels

# Display the first few rows of the dataset
print(df.head())
```

```
   Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Label
0          0          1          1          1          0      0
1          0          0          1          1          1      1
2          0          1          1          0          0      0
3          0          0          1          0          0      1
4          1          0          0          0          0      1
```

[36]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, :-1],
 ↪df['Label'], test_size=0.2, random_state=42)

# Train a Bernoulli Naive Bayes classifier
clf = BernoulliNB()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 56.50%
```

1.4 Categorical Naive Bayes

The `CategoricalNB` classifier in scikit-learn is designed for datasets where features are categorical rather than binary. It is based on the categorical distribution, which is suitable for representing discrete data.

Here's some information about CategoricalNB:

- Data Representation: Categorical features are assumed, meaning each feature can take on a limited, discrete set of values.
- Probability Estimation: The model estimates probabilities using the categorical distribution.
- Laplace Smoothing: Similar to other naive Bayes classifiers, CategoricalNB can apply Laplace smoothing to handle cases where certain feature values may not appear in the training data.

Input Data:

The input data is expected to be a 2D array-like or sparse matrix with shape (n_samples, n_features). Each feature is treated as a categorical variable.

[38]:
```python
from sklearn.datasets import load_iris
from sklearn.naive_bayes import CategoricalNB

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Train Categorical Naive Bayes classifier
clf = CategoricalNB()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 96.67%
```