Naïve Bayes

Data Mining for Business Analytics in Python

Shmueli, Bruce, Gedeck & Patel

Characteristics

- Data-driven, not model-driven
- Makes "conditional independence" assumption about the data
- Named after mid-16th century English statistician and Presbyterian minister Thomas Bayes

Naïve Bayes: The Basic Idea

- For a given new record to be classified, find other records like it (i.e., same values for the predictors)
- What is the prevalent class among those records?
- Assign that class to your new record

Application

- Gaussian (works with continuous numerical variables)

- Bernoulli (Designed for binary/Boolean categorical features)

- Multinomial (mostly for text classification, requires categorical/integers/count/frequencies not continuous. See below).

  - Numerical variable must be binned and converted to categorical

  - Can be used with very large data sets

  - Example: Spell check programs assign your misspelled word to an established "class" (i.e., correctly spelled word)

Exact (Complete) Bayes Classifier

- Relies on finding other records that share <u>same predictor values</u> as record-to-be-classified.

- Want to find "probability of belonging to class *C*, given specified values of predictors."

- Even with large data sets, may be hard to find other records that **exactly match** your record, in terms of predictor values.

Solution – Naïve Bayes

- Assume independence of predictor variables (within each class)

- Use multiplication rule

- Find same probability that record belongs to class C, given predictor values, <u>without</u> limiting calculation to records that share all those same values

Calculations

1. Take a record, and note its predictor values
2. Find the probabilities those predictor values occur across all records in C_1
3. Multiply them together, then by proportion of records belonging to C_1
4. Same for C_2, C_3, etc.
5. Prob. of belonging to C_1 is value from step (3) divided by sum of all such values $C_1 \ldots C_n$
6. Establish & adjust a "cutoff" prob. for class of interest
7. The above steps lead to the Naive Bayes formula for calculating the probability that a record with a given set of predictor values $x_1, \ldots, x_p$ belongs to class C_1 among m classes:

$$P_{nb}(C_1 \mid x_1, \ldots x_p)$$

$$= \frac{P(C_1)[P(x_1 \mid C_1)P(x_2 \mid C_1) \cdots P(x_p \mid C_1)]}{P(C_1)[P(x_1 \mid C_1)P(x_2 \mid C_1) \cdots P(x_p \mid C_1)] + \cdots + P(C_m)[P(x_1 \mid C_m)P(x_2 \mid C_m) \cdots P(x_p \mid C_m)]}.$$

Example: Financial Fraud

Target variable: Audit finds fraud, no fraud

Predictors:

Prior pending legal charges (yes/no)

Size of firm (small/large)

Charges?	Size	Outcome
y	small	truthful
n	small	truthful
n	large	truthful
n	large	truthful
n	small	truthful
n	small	truthful
y	small	fraud
y	large	fraud
n	large	fraud
y	large	fraud

Exact Bayes Calculations

Goal: classify (as “fraudulent” or as “truthful”) a small firm with charges filed

There are 2 firms like that, one fraudulent and the other truthful

P(fraud | charges=y, size=small) = ½ = 0.50

Note: Calculation is limited to the two firms matching those characteristics

Naïve Bayes Calculations

Same goal as before (fraud)

Compute 2 quantities:

- Proportion of "charges = y" among frauds, times proportion of "small" among frauds, times proportion of frauds = 3/4 * 1/4 * 4/10 = 0.075
- Prop. "charges = y" among truthfuls, times prop. "small" among truthfuls, times prop. truthfuls = 1/6 * 4/6 * 6/10 = 0.067

P(fraud | charges, small) = 0.075/(0.075+0.067)

= 0.53

Naïve Bayes, cont.

- Note that probability **estimate** does not differ greatly from **exact**
- All records are used in calculations, not just those matching predictor values
- This makes calculations practical in most circumstances
- Relies on assumption of independence between predictor variables within each class

Independence Assumption

- Not strictly justified (variables often correlated with one another)
- Often "good enough" – Ranking of probabilities (to determine the proper outcome) is more important than unbiased estimate of actual probabilities
- For classification purposes, it is the rank orderings that matter

Example - Flight Delays

Predictors

Day of Week	Coded as 1 = Monday, 2 = Tuesday, ..., 7 = Sunday
Sch. Dep. Time	Broken down into 18 intervals between 6:00 AM and 10:00 PM
Origin	Three airport codes: DCA (Reagan National), IAD (Dulles), BWI (Baltimore–Washington Int'l)
Destination	Three airport codes: JFK (Kennedy), LGA (LaGuardia), EWR (Newark)
Carrier	Eight airline codes: CO (Continental), DH (Atlantic Coast), DL (Delta), MQ (American Eagle), OH (Comair), RU (Continental Express), UA (United), and US (USAirways)

OUTCOME: On-time (1) , Delay (0)

Data Preparation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB

delays_df = pd.read_csv('FlightDelays.csv')


# convert to categorical
delays_df.DAY_WEEK = delays_df.DAY_WEEK.astype('category')
delays_df['Flight Status'] = delays_df['Flight
Status'].astype('category')
# create hourly bins departure time
delays_df.CRS_DEP_TIME = [round(t / 100) for t in
delays_df.CRS_DEP_TIME]
delays_df.CRS_DEP_TIME = delays_df.CRS_DEP_TIME.astype('category')
predictors = ['DAY_WEEK', 'CRS_DEP_TIME', 'ORIGIN', 'DEST',
   'CARRIER']
outcome = 'Flight Status'
```

Video

Dummies and Partitioning

```
X = pd.get_dummies(delays_df[predictors], drop_first=True)
y = delays_df['Flight Status'].astype('category')


# split into training and validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
   test_size=0.40, random_state=1)
```

Gaussian vs. Multinomial Naive Bayes

- When referred to NB, people usually want to learn about the multinomial NB Classifier which considers categorical predictors. However, there is another commonly used version of NB, called Gaussian NB Classification.
- A Gaussian NB is based on continuous variables that are assumed to have a Gaussian (Normal) distribution. It, however, can still work even if the data is not normally distributed or the predictors are dependent.
- The steps to compute Gaussian NB: (1) Calculate the sample mean and standard deviation of each group in outcome, (2) Draw Normal distribution for each group (using their mean and STD) and show them all on a single axis, (3) Determine cutoff values where curves cut each other, (4) Calculate the value of normal distribution by plugging in values in this formula:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

(5) Multiply the values from (4) for all predictors within each class by the probability of that class, similar to the previous formula, shown again below:

$$P_{nb}(C_1 \mid x_1, \ldots x_p)$$
$$= \frac{P(C_1)[P(x_1 \mid C_1)P(x_2 \mid C_1) \cdots P(x_p \mid C_1)]}{P(C_1)[P(x_1 \mid C_1)P(x_2 \mid C_1) \cdots P(x_p \mid C_1)] + \cdots + P(C_m)[P(x_1 \mid C_m)P(x_2 \mid C_m) \cdots P(x_p \mid C_m)]}.$$

[Video]

Run Multinomial Naive Bayes

```
# run naive Bayes
delays_nb=MultinomialNB(alpha=0.01) #alpha is a smoothing hyperparameter
delays_nb.fit(X_train, y_train)

# predict probabilities. The predict_proba method returns the estimated probability of each class for each data point.
predProb_train = delays_nb.predict_proba(X_train)
predProb_valid = delays_nb.predict_proba(X_valid)

# predict class membership
y_valid_pred = delays_nb.predict(X_valid)
```

Run Multinomial Naive Bayes

- Before using the output, let's see how the algorithm works. We start by generating pivot tables for the outcome vs. each of the five predictors using the training set, in order to obtain conditional probabilities. Note that in this example, there are no predictor values that were not represented in the training data.

```
# split the original data frame into a train and test using the same random_state
train_df, valid_df = train_test_split(delays_df, test_size=0.4, random_state=1)

pd.set_option('display.precision', 4)

# probability of flight status
print(train_df['Flight Status'].value_counts() / len(train_df))
print()

for predictor in predictors:
    # construct the frequency table
    df = train_df[['Flight Status', predictor]]
    freqTable = df.pivot_table(index='Flight Status', columns=predictor, aggfunc=len,
    observed=False)
```

The aggfunc=len is the aggregation function to apply when summarizing the data. In this case, len counts the number of occurrences (frequency) of each combination of Flight Status and the specific predictor variable values. This will yield the count of flights for each combination.

When observed=False, it includes all categories of the specified columns, even if there are no occurrences of certain combinations. If True, only the observed combinations will be included. This is useful for ensuring that all possible categories are represented in the pivot table, even if some combinations do not appear in the data.

```
    # divide each value by the sum of the row to get conditional probabilities
    propTable = freqTable.apply(lambda x: x / sum(x), axis=1)
    print(propTable)
    print()
```

Run Multinomial Naive Bayes

```
ontime     0.8023
delayed    0.1977

DAY_WEEK            1       2       3       4       5      6       7
Flight Status
delayed        0.1916  0.1494  0.1149  0.1264  0.1877  0.069  0.1609
ontime         0.1246  0.1416  0.1445  0.1794  0.1690  0.136  0.1048

CRS_DEP_TIME        6       7       8       9      10      11      12      13  \
Flight Status
delayed        0.0345  0.0536  0.0651  0.0192  0.0307  0.0115  0.0498  0.0460
ontime         0.0623  0.0633  0.0850  0.0567  0.0519  0.0340  0.0661  0.0746

CRS_DEP_TIME       14      15      16      17      18      19      20      21
Flight Status
delayed        0.0383  0.2031  0.0728  0.1533  0.0192  0.0996  0.0153  0.0881
ontime         0.0576  0.1171  0.0774  0.1001  0.0349  0.0397  0.0264  0.0529

ORIGIN            BWI     DCA     IAD
Flight Status
delayed        0.0805  0.5211  0.3985
ontime         0.0604  0.6478  0.2918

DEST              EWR     JFK     LGA
Flight Status
delayed        0.3793  0.1992  0.4215
ontime         0.2663  0.1558  0.5779

CARRIER            CO      DH      DL      MQ      OH      RU      UA      US
Flight Status
delayed        0.0575  0.3142  0.0958  0.2222  0.0077  0.2184  0.0153  0.0690
ontime         0.0349  0.2295  0.2040  0.1171  0.0104  0.1690  0.0170  0.2181
```

Classify New Record using Multinomial Naive Bayes

- To classify a new flight, we compute the probability that it will be delayed and the probability that it will be on time. Recall that since both probabilities will have the same denominator, we can just compare the numerators. Each numerator is computed by multiplying all the conditional probabilities of the relevant predictor values and, finally, multiplying by the proportion of that class (in this case p(delayed) = 0.2). Let us use an example: to classify a Delta flight from DCA to LGA departing between 10:00 AM and 11:00 AM on a Sunday, we first compute the numerators using the values from the pivot tables:

$$\hat{P}(\text{delayed}|\text{Carrier} = \text{DL, Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA, Origin} = \text{DCA})$$
$$\propto (0.0958)(0.1609)(0.0307)(0.4215)(0.5211)(0.2) = 0.000021,$$
$$\hat{P}(\text{ontime}|\text{Carrier} = \text{DL, Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA, Origin} = \text{DCA})$$
$$\propto (0.2040)(0.1048)(0.0519)(0.5779)(0.6478)(0.8) = 0.00033.$$

- The symbol ∝ means "is proportional to," reflecting the fact that this calculation deals only with the numerator in the naive Bayes. Comparing the numerators, it is therefore, more likely that the flight will be on time. Note that a record with such a combination of predictor values does not exist in the training set, and therefore we use the naive Bayes rather than the exact Bayes. To compute the actual probability, we divide each of the numerators by their sum:

$$\hat{P}(\text{delayed}|\text{Carrier} = \text{DL, Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA, Origin} = \text{DCA})$$
$$= \frac{0.000021}{0.000021 + 0.00033} = 0.058,$$
$$\hat{P}(\text{on time}|\text{Carrier} = \text{DL, Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA, Origin} = \text{DCA})$$
$$= \frac{0.00033}{0.000021 + 0.00033} = 0.942.$$

```
newData = pd.DataFrame({'DAY_WEEK': [7], 'CRS_DEP_TIME': [10], 'ORIGIN': ['DCA'], 'DEST': ['LGA'], 'CARRIER': ['DL']})

# One-hot encode the new record using the same process as the training set
newData_encoded = pd.get_dummies(newData, drop_first=True)

# Align data_encoded to the same feature set as the training data
for col in X.columns:
    if col not in newData_encoded.columns:
    newData_encoded[col] = 0 # Set to 0 if the column is not present

# Reorder columns to match the training data
newData_encoded = newData_encoded[X.columns]
print()
delays_nb.predict(newData_encoded)[0]
'ontime'
```

Classify New Record using Exact Naive Bayes

- The predicted probability and class for the example flight, which coincide with our manual calculation, is shown below.

```
# classify a specific flight by searching in the dataset
# for a flight with the same predictor values
df = pd.concat([pd.DataFrame({'actual': y_valid, 'predicted': y_valid_pred}),
                pd.DataFrame(predProb_valid, index=y_valid.index)], axis=1)
mask = ((X_valid.CARRIER_DL == 1) & (X_valid.DAY_WEEK_7 == 1) &
        (X_valid.CRS_DEP_TIME_10 == 1) &  (X_valid.DEST_LGA == 1) &
        (X_valid.ORIGIN_DCA == 1))
df[mask]
```

	actual	predicted	0	1
1225	ontime	ontime	0.057989	0.942011

Naive Bayes Remarks

- The alpha parameter is a smoothing (regularization) hyperparameter. It is used to avoid zero probabilities in cases where a particular feature value has not been observed in a specific class during training. The purpose of smoothing is to prevent the model from becoming too confident about probabilities when there is little or no evidence in the training data.
- A smaller alpha value, such as 0.01, indicates stronger smoothing. It means the model is more conservative and assigns non-zero probabilities to feature values even if they are rare or unseen in the training data.
- In machine learning, we distinguish between "parameters" and "hyperparameters" because they have different roles and purposes in the modeling process:
- Parameters are internal values that the model learns directly from the training data. They are adjusted during the model's training process to optimize performance. Examples:
  - In linear regression, the coefficients (slopes) of the model are the parameters.
  - In a Neural Network, the weights and biases of the network are parameters.
- Hyperparameters are external settings that must be defined before the learning process begins. They are not learned from the data but are instead manually chosen and control the overall behavior of the model. Examples:
  - The number of neighbors in a K-Nearest Neighbors (KNN) model.
  - The number of layers or neurons in a Neural Network.

Advantages

- Handles purely categorical data well
- Works well with very large data sets
- Simple & computationally efficient
- Ability to handle categorical variables directly
- Often outperforms more sophisticated classifiers (even when the underlying assumption of independent predictors is far from true)

Shortcomings

- Requires large number of records
- Problematic when a predictor category is not present in training data
  - Assigns 0 probability of response, ignoring information in other variables

On the other hand …

- Good performance is obtained when the goal is classification or ranking of records according to their probability of belonging to a certain class. However, when the goal is to estimate the probability of class membership (propensity), this method provides very biased results.
- Probability rankings are more accurate than the actual probability estimates
  - Good for applications using lift (e.g. response to mailing), less so for applications requiring probabilities (e.g. credit scoring)

SPAM Filtering

Filtering spam in e-mail has long been a widely familiar application of data mining. Spam filtering, which is based in large part on natural language vocabulary, is a natural fit for a naive Bayesian classifier, which uses exclusively categorical variables. Most spam filters are based on this method, which works as follows:

Humans review a large number of e-mails, classify them as “spam” or “not spam,” and from these select an equal (also large) number of spam e-mails and non-spam e-mails. This is the training data.

These e-mails will contain thousands of words; for each word, compute the frequency which it occurs in the spam dataset, and the frequency which it occurs in the non-spam dataset. Convert these frequencies into estimated probabilities (i.e., if the word “free” occurs in 500 out of 1000 spam e-mails, and only 100 out of 1000 non-spam e-mails, the probability that a spam e-mail will contain the word “free” is 0.5, and the probability that a non-spam e-mail will contain the word “free” is 0.1).

If the only word in a new message that needs to be classified as spam or not spam is “free,” we would classify the message as spam, since the Bayesian posterior probability is 0.5/(0.5 + 0.1) or 5/6 that, given the appearance of “free,” the message is spam.

Of course, we will have many more words to consider. For each such word, the probabilities described are calculated, and multiplied together, and the Bayesian formula is applied to determine the naive Bayes probability of belonging to the classes. In the simple version, class membership (spam or not spam) is determined by the higher probability.

In a more flexible interpretation, the ratio between the “spam” and “not spam” probabilities is treated as a score for which the operator can establish (and change) a cutoff threshold—anything above that level is classified as spam.

Users have the option of building a personalized training database by classifying incoming messages as spam or not spam, and adding them to the training database. One person’s spam may be another person’s substance.

It is clear that, even with the “Naive” simplification, this is an enormous computational burden. Spam filters now typically operate at two levels—at servers (intercepting some spam that never makes it to your computer) and on individual computers (where you have the option of reviewing it). Spammers have also found ways to “poison” the vocabulary-based Bayesian approach, by including sequences of randomly selected irrelevant words. Since these words are randomly selected, they are unlikely to be systematically more prevalent in spam than in non-spam, and they dilute the effect of key spam terms such as “Viagra” and “free.” For this reason, sophisticated spam classifiers also include variables based on elements other than vocabulary, such as the number of links in the message, the vocabulary in the subject line, determination of whether the “From:” e-mail address is the real originator (anti-spoofing), use of HTML and images, and origination at a dynamic or static IP address (the latter are more expensive and cannot be set up quickly).