Week 7 Naive Bayes Classifer

Theory and Practice

Ying Lin, Ph.D

October 11, 2019

Outline

- Motivation: what and why?
- Probability (marginal, joint, conditional probability)
- Bayes theorem (likelihood and posterior probability)
- Naive Bayes Classifier
- Maximum Likelihood Estimator (MLE) vs. Maximum A Posterior Estimator (MAP)
- R/Python demo of NBC

Motivating Examples

- Suppose you are a medical doctor using a cancer diagnosis kit in a screening test for your patients
- A high sensitivity and a high specificity diagnostic test (95% accuracy and 0.90 specificity for cancer detection (or spam detection, or credit card fradulent detection)
- Questions: if someone is tested positive, what would be your diagnosis?

Overview of NBC

- Supervised learning method for classification problem
- probabilistic classifier
  - Given a set of attribute/feature values, a Naive Bayes Classifier (NBC) predicts a distribution over a set of outcomes
  - Above probability could be converted into discrete classification and measures the degree of certainty

Applications

- Text classification/Spam filtering/Sentiment analysis
- Recommendation system
- Real time prediction
- Multiclass prediction

Review of Probability

- Marginal probability: the probability $P(A)$ of an event A occurring. It is not conditioned on another event.
- Joint probability: the probability $P(A \cap B)$ of events A and B both occurring.
- Conditional probability: the probability $P(A|B)$ of event A occurring given that event B occurs
  - $P(A|B) = \frac{P(A \cap B)}{P(B)}$
  - when events A and B are independent, $P(A|B) = P(A)$; therefore $P(A \cap B) = P(A)P(B)$

Bayes Theorem and its Derivation

- Bayes' theorem relates the conditional and marginal probabilities of stochastic events A and B
  - $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$
- Derivation from conditional probabilities
  - $P(A|B) = \frac{P(A \cap B)}{P(B)}$
  - $P(B|A) = \frac{P(A \cap B)}{P(A)}$
  - $P(A|B)P(B) = P(A \cap B) = P(B|A)P(A)$
  - $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Bayes Theorem in Likehood and Posterior Probability

- Goal of the NBC is to the unknown parameter θ given the data $\mathcal{D}$
  - Let $\mathcal{D}$ be a set of data generated from some distribution parameterized by θ
  - θ is in the sense that it does not consider any information about $\mathcal{D}$
  - Denominator $P(Evidence)$ is independent of θ, therefore constant given the dataset and can be considered as a normalization factor and dropped in classification or estimator

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}$$

$$Posterior = \frac{Likelihood \times Prior}{Evidence}$$

8/29

NBC Algorithm

- Convert the dataset into a frequency table
- Create likelihood table by finding all the conditional probability $P(Attribute = " Value " | Class)$
- Apply Bayes theorem to calculate the posterior probability for each class and the class with the highest posterior probability is outcome of prediction
- In the case of multiple attributes dataset, apply naive Bayes classifier by taking independent assumption

Naive Assumptions

- Conditional independence assumption

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta) * P(\theta)}{P(\mathcal{D})}$$

$$= \frac{P(\theta)\prod_i^n P(d_i|\theta)}{P(\mathcal{D})}$$

- Independent Assumption
  - Reduce number of probabilities to estimate (from exponential to linear to number of attributes)
  - Reduce possibility of zero probability
  - Decent empirical performance despite violation of this assumption

Additional Issues with Probability Estimation

- Zero probability: smoothing
  - Laplace method
    - Original: $P(A_i|C) = \frac{N_{ic}}{N_c}$
    - Laplace: $P(A_i|C) = \frac{N_{ic}+1}{N_c+m}$
    - m: number of possible attribute values
- Continous attribute
  - Discretization
  - Probability density function
    - Assume a Gaussian distribution for the continuous attribute
    - $$P(A_i|C_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{\frac{-(A_i-\mu_j)^2}{2\sigma_j^2}}$$
    - Gaussian naive Bayes method

Model Parameters

- fL: add one smoothing for zero probability issue
- kernel: probability density function for continuous attributes
  - FALSE (by default): normal density function is estimated
  - TRUE: a kernel density estimate is used

Properties of the Algorithm

- Pros
  - Provide both classification and probability estimation
  - Robust to noise and irrelevant attributes
  - Easy and fast to predict class of test data set. Perform well in multi class prediction
  - When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression with less training data.
- Cons
  - Zero probability
  - Independent assumption

Maximum Likelihood Estimator (MLE)

- MLE estimator

$$
\begin{aligned}
h_{MLE} &= argmax_h P(D|h) \\
&= argmax_h log \prod_i P(D_i|h) \\
&= argmax_h \sum_i log(D_i|h)
\end{aligned}
$$

- MLE for some common distributions
  - Binomial distribution: $\hat{p}_{MLE} = \frac{x}{n}$
  - Gaussian distribution parameters: $\hat{\mu}_{MLE} = \frac{\sum_i^n x_i}{n}$ and $\hat{\sigma}_{MLE} = \sqrt{\frac{\sum_i^n (X_i - \mu)}{n}}$

Maximum A Posteriori (MAP)

- MAP

$$
\begin{aligned}
h_{MAP} &= argmax_h P(h|D) \\
&= argmax_h \frac{P(D|h)P(h)}{P(D)} \\
&= argmax_h \sum_i log(D_i|h)P(h)
\end{aligned}
$$

- $\theta_{MAP} = argmax_\theta \sum_i logP(X_i|\theta) + logP(\theta)$

MAP vs. MLE

- MLE can be considered as a special case of MAP with uniform distribution for $P(h)$ or uniform prior
  - The only difference between MLE and MAP is the inclusion of prior $P(\theta)$ in MAP
  - Therefore θ or prior distribution is an adjustment term to MLE in calculating MAP
- log function is applied because it is a monotonically increasing function, so $argmax_h P(D_i|h)$ is the same as $argmax_h \prod_i P(D_i|h)$
  - Avoid underflow issue for small number computation
- Example: imbalanced data challenge, such as cancer diagnosis in the motivating example, leads to wrong conclusion with MLE approach

Bayesianism vs. Frequentism

- Frequentist believes what you observe in the sample data is representative of the true distribution
- Bayesian believes the observed frequency needs to be combined with the prior belief/knowledge to infer the true distribution
  - Bayesian statistic appears to be , depending on the prior belief
- Frequentist tries to estimate MLE while Bayesian tries to estimate MAP by considering prior

Demo Dataset: SPAM

```
library(ElemStatLearn)
library(caret)
data(spam)
dim(spam)
```

```
## [1] 4601   58
```

```
names(spam)
```

```
##  [1] "A.1"  "A.2"  "A.3"  "A.4"  "A.5"  "A.6"  "A.7"  "A.8"  "A.9"  "A.10"
## [11] "A.11" "A.12" "A.13" "A.14" "A.15" "A.16" "A.17" "A.18" "A.19" "A.20"
## [21] "A.21" "A.22" "A.23" "A.24" "A.25" "A.26" "A.27" "A.28" "A.29" "A.30"
## [31] "A.31" "A.32" "A.33" "A.34" "A.35" "A.36" "A.37" "A.38" "A.39" "A.40"
## [41] "A.41" "A.42" "A.43" "A.44" "A.45" "A.46" "A.47" "A.48" "A.49" "A.50"
## [51] "A.51" "A.52" "A.53" "A.54" "A.55" "A.56" "A.57" "spam"
```

```
str(spam)
```

```
## 'data.frame':    4601 obs. of  58 variables:
##  $ A.1 : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ A.2 : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
```

Create Training and Validation Datasets

```
train_index <- createDataPartition(spam$spam, p = 0.8, list = FALSE)

spam_train <- spam[train_index, ]
spam_test <- spam[-train_index, ]

table(spam_train$spam)
```

```
##
## email  spam
##  2231  1451
```

```
table(spam_test$spam)
```

```
##
## email  spam
##   557   362
```

Train the Basic NB Model with Default Setting

```
library(klaR)
model_nb1 <- train(spam ~ ., data = spam_train, method = "nb")
model_nb1
```

```
## Naive Bayes
##
## 3682 samples
##   57 predictor
##    2 classes: 'email', 'spam'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3682, 3682, 3682, 3682, 3682, 3682, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.7059212  0.4442480
##    TRUE      0.5812787  0.2586204
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
```

Predict with NBC

```
predict_nb1 <- predict(model_nb1, newdata = spam_test, type = "raw")
confusionMatrix(predict_nb1, spam_test$spam)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   340   15
##      spam    217  347
##
##                Accuracy : 0.7476
##                  95% CI : (0.7182, 0.7754)
##     No Information Rate : 0.6061
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5183
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6104
##             Specificity : 0.9586
##          Pos Pred Value : 0.9577
##          Neg Pred Value : 0.6152
##              Prevalence : 0.6061
```

Fine Tune the Models

```
model_nb2 <- train(spam ~ ., data = spam, method = "nb",
             trControl = trainControl(method = "cv", number = 3),
             tuneGrid = expand.grid(fL = 1:3, usekernel = c(TRUE, FALSE), adjust = 1:3))
model_nb2
```

```
## Naive Bayes
##
## 4601 samples
##   57 predictor
##    2 classes: 'email', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 3067, 3067, 3068
## Resampling results across tuning parameters:
##
##   fL  usekernel  adjust  Accuracy   Kappa
##   1   FALSE      1       0.7072053  0.4478853
##   1   FALSE      2       0.7072053  0.4478853
##   1   FALSE      3       0.7072053  0.4478853
##   1    TRUE      1       0.5663962  0.2366004
##   1    TRUE      2       0.5340137  0.1903944
##   1    TRUE      3       0.5203209  0.1710591
```

Performance Issues with Package

- train() function runs slower than the original ML function
  - Parameter tuning: by default, three parameter levels are tested to identify the best performing models
  - Resampling for model performance evaluation. By default, 25 repeats of Bootstrap resampling are tried
  - Therefore, by default, 3 * 25 = 125 times of model training/evaluation which significantly slows down the modeling process
- Solution
  - Specify number of parameters to tune with tuneLength and tuneGrid arguments
  - Reduce or remove resampling by decreasing number of repeats or setting trainControl(method = “none”)

Comparison of Execution Time for Various NB Models by (1)

```
start1 <- Sys.time()
default_model <- train(spam ~ ., data = spam_train, method = "nb")
Sys.time() - start1
```

```
## Time difference of 2.230981 mins
```

```
start2 <- Sys.time()
model2 <- train(spam ~ ., data = spam_train, method = "nb",
                trControl = trainControl(method = "cv", number = 3))
Sys.time() - start2
```

```
## Time difference of 14.80887 secs
```

Comparison of Execution Time for Various NB Models by (2)

```
start3 <- Sys.time()
model3 <- train(spam ~ ., data = spam_train, method = "nb",
                trControl = trainControl(method = "none"),
                tuneGrid = expand.grid(fL = 1, usekernel = T, adjust = 1))
Sys.time() - start3
```

```
## Time difference of 0.384975 secs
```

Comparison of Multiple Naive Bayes Packages

- package: naiveBayes()
  - Gaussian distribution assumption for continuous attributes in probability estimation
- package: NaiveBayes()
  - Based on David Meyer's package
  - Wrapper function for naiveBayes()
  - Extended for kernel estimated densities and user specified probabilities
- \+ packages: train(method = "nb")
  - Standardize train function interface
  - Allow resampling for performance evaulation and automatic parameter tuning, among many functionalities from package

Import Python Packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, ShuffleSpl
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
```

Data Preparation

```
spam = r.spam
X = spam.drop('spam', axis=1)
y = spam['spam']
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=0.3, random_state=16)
X_train.shape
```

```
## (3220, 57)
```

```
y_train.shape
```

```
## (3220,)
```

Gaussian NB Model

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
## GaussianNB(priors=None, var_smoothing=1e-09)
```

```
gnb_pred = gnb.predict(X_test)
print(f"Accuracy: {round(metrics.accuracy_score(y_test, gnb_pred)*100, 2)}%")
```

```
## Accuracy: 82.77%
```

```
gnb_pred_prob = gnb.predict_proba(X_test)
pred_tab = pd.DataFrame(np.concatenate((gnb_pred_prob, gnb_pred[:, None], np.array(y_test)[:, N
pred_tab.head(5)
```

```
##        P(email)      P(spam) Prediction   Real
## 0  5.60664e-15            1       spam   spam
## 1            1  2.97918e-31      email  email
## 2  7.80239e-18            1       spam   spam
## 3            1  5.61007e-57      email  email
## 4  1.58776e-26            1       spam   spam
```

29/29