MA5810: Introduction to Data Mining

Week 2; Collaborate Session 1: Naive Bayes

Martha Cooper, PhD

JCU Masters of Data Science

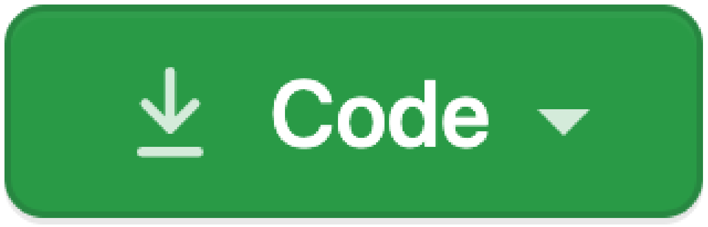
2019-21-9 (updated: 2020-11-03)

Housekeeping

- Collaborate 1 = Wednesdays 6-7pm (Martha)
- Collaborate 2 = Thursdays 7-8pm (Hongbin)

For my Collaborate Sessions, you can get the slides & R code for each week here:

https://github.com/MarthaCooper/MA8510



Subject: MA5810 Intro to Data Mining

MA5810 Learning Outcomes

1. Overview of Data Mining and Examples
2. Unsupervised data mining methods e.g. clustering and outlier detection;
3. Unsupervised and supervised techniques for dimensionality reduction;
4. Supervised data mining methods for pattern classification (Today = Naive Bayes);
5. Apply these concepts to real data sets using R (Today).

Today's Goals

- Understand the background behind Naive Bayes Classifiers
- Apply Naive Bayes Classifiers to real datasets using R
- Understand the pros and cons of Naive Bayes Classifiers

Naive Bayes Classifier

- Supervised, probabilistic classifier based on Bayes Theorem
- Strong independence assumptions
- A famous use case is spam filtering of emails
- Factor variable - Multinomial Naive Bayes Classifier
- Continuous variables - Gaussian Naive Bayes Classifier
- Given these features, does this sample belong to class X or Y?

Data

- We are going to use Class, Sex and Age data to predict Survival on the Titanic
- We first need to get the data and do a bit of wrangling

```r
data(Titanic)
str(Titanic)

#function to convert to long format (one passenger per row)
counts_to_cases <- function(x){
  #how many time to repeat each row
  inds <- rep.int(seq_len(nrow(x)), times = x[["Freq"]])
  #remove frequency column
  x <- select(x, -Freq)
  #get rows from x
  x[inds,]
}

case_titanic <- counts_to_cases(as.data.frame(Titanic))

#head(case_titanic, 3)
#dim(case_titanic)
```

Split into test and training

- We will train our Naive Bayes Classifier on 80% of the data
- We can use the caret package to split the data

```
library(caret, warn.conflicts = F, quietly = T)

set.seed(1234)
split <- createDataPartition(case_titanic$Survived, p = 0.8, list = FALS

train <- case_titanic[split, ]
test <- case_titanic[-split, ]

c(nrow(train), nrow(test)) # print number of observations in test vs. t

table(train$Survived) %>% prop.table() # Proportions of Survived Yes an
```

Naive Bayes: An Overview

- Bayes Theorem, by Reverend Thomas Bayes, is about conditional probability; the probability of A given that B occurred [deonoted as $P(A|B)$]

- In our Titanic example, the probability of a passenger having survived S_k (where S_{yes} = survived and S_{no} = did not survive), given that it's predictor values are $x_1, x_2 \ldots x_p$ [denoted as $P(S_k|x_1, \ldots x_p)$].

Naive Bayes: An Overview

Bayes Theorem looks like this:

$$P(S_k|X) = \frac{P(X|S_k) \cdot P(S_k)}{P(X)}$$

where:

- $P(S_k)$ The prior probability of the outcome - based on the training data, what is the probability of a person surviving or not? Our prior probability of survival is ~32% and the probability of not surviving is ~68%.
- $P(X)$ The probability of observing the predictor variables.
- $P(X|S_k)$ The conditional probability or likelihood, For each class (*i.e.* survived and no survived), what is the probability of observing the predictors.
- $P(S_k|X)$ The posterior probability. We update our prior probabilities with our observed information to find the posterior probability that an observation has class S_k.

Naive Bayes: An Overview

$$P(S_{yes}|X) < P(S_{no}|X)$$

$$P(S_{yes}|X) = P(S_{no}|X)$$

$$P(S_{yes}|X) > P(S_{no}|X)$$

- Because we are comparing between classes, we don't need the denominator $P(X)$ as it will be the same in both classes.
- So what we actually end up calculating is proportional (not equal to) to $P(S_k|X)$

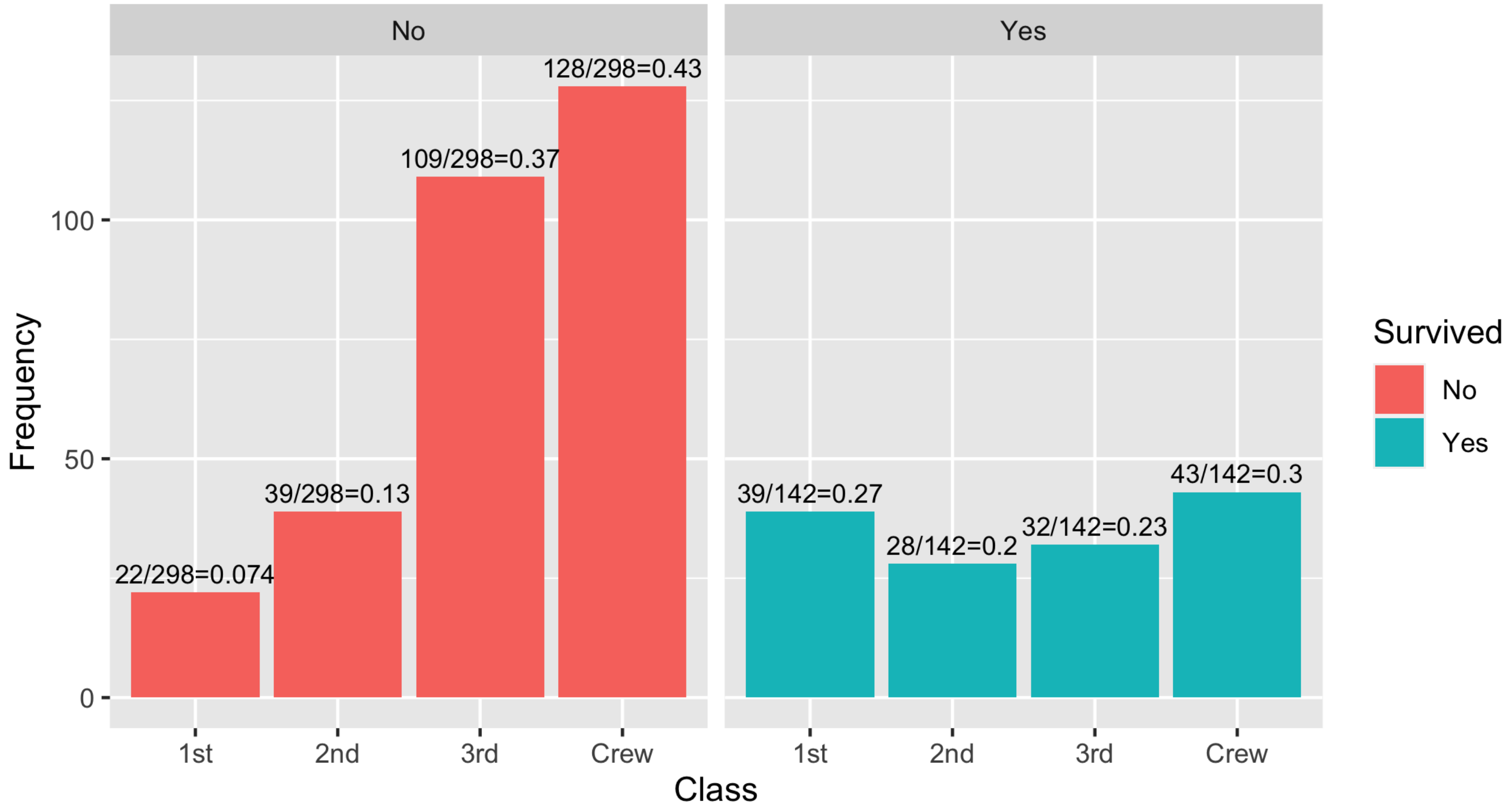
Naive Bayes: Why Naive?

- Naive Bayes makes the assumption that the predictor variables are all *conditionally independent* of each other
- So $P(S_k|X)$ is proportional to $P(S_k)$ times the product of conditional probabilities of all predictors in the class.

$$P(S_k|X) \propto P(S_k) \cdot \prod_{i=1}^{n} P(x_i|S_k)$$

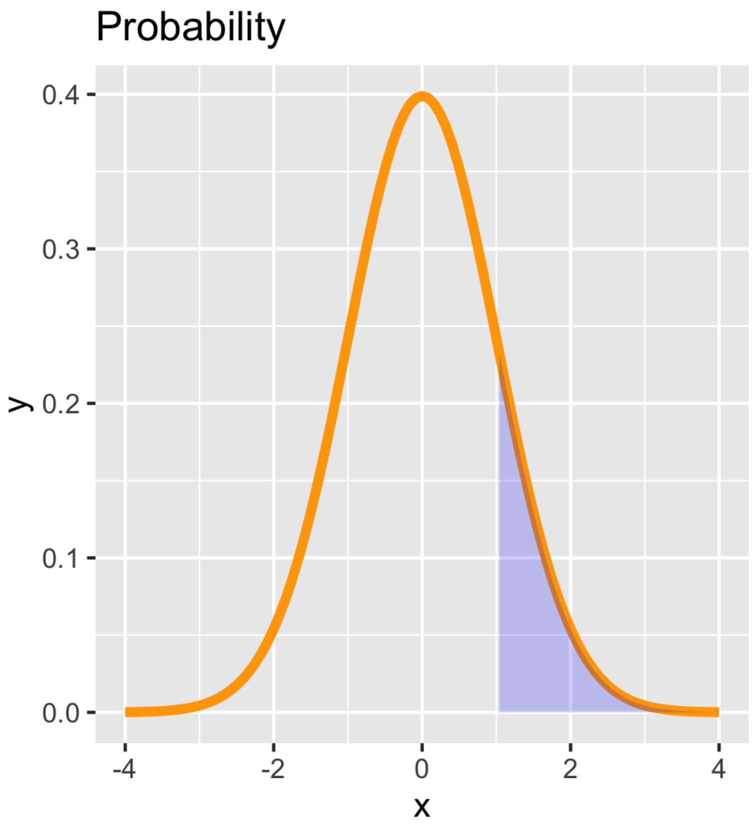
Calculating Likelihoods

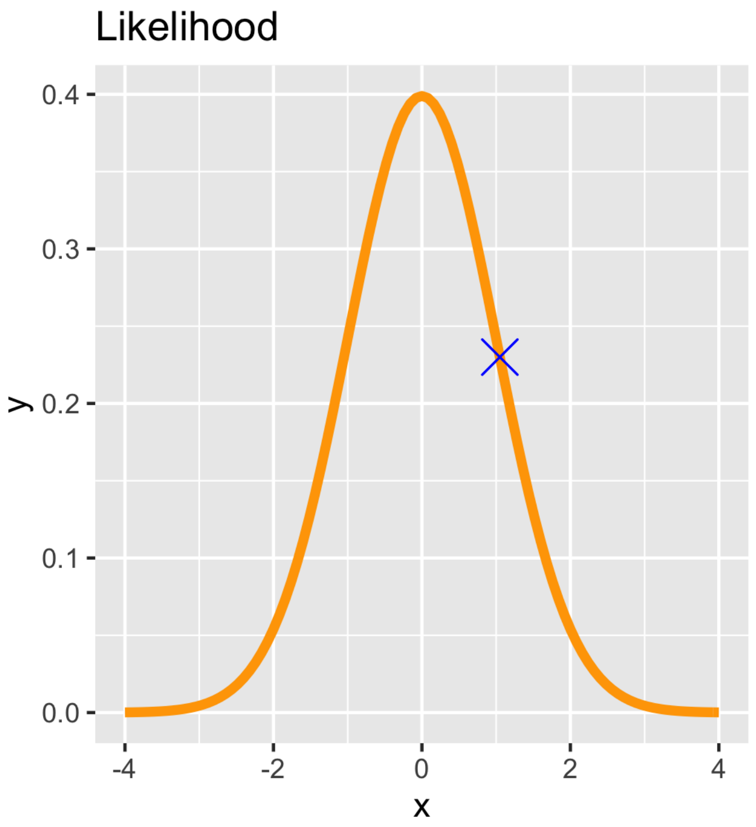
- Factor Variables - use *frequencies from the training data, or multinomial or Bernoulli distribution* (distribution for discrete variables), to calculate probabilities of the each predictor in each class (like our Titanic data).
- For factor variables, probabilities = likelihoods.



Calculating Likelihoods

- Continuous Variables - use a *Gaussian or Kernal (non-parametric) density estimate*, to calculate the likelihood of each predictor in a class.
- n.b. For continuous variable, probabilities and likelihoods are different





Naive Bayes Classifier with factor variables - Titanic Survival

There are lots of R packages to apply naive bayes (e1071, klaR, naivebayes etc...) but I like caret - common framework for many algorithms.

```r
predictors <- names(train)[1:3] #Create response and predictor data

x <- train[,predictors] #predictors
y <- train$Survived #response

train_control <- trainControl(method = "cv", number = 10) #set up 10 fo

survival_mod1 <- train( #train the model
  x = x,
  y = y,
  method = "nb",
  trControl = train_control
)

confusionMatrix(survival_mod1) #results of model on training data

pred <- predict(survival_mod1, newdata = test) #make predictions

confusionMatrix(pred, test$Survived) #assessing the classifier
```

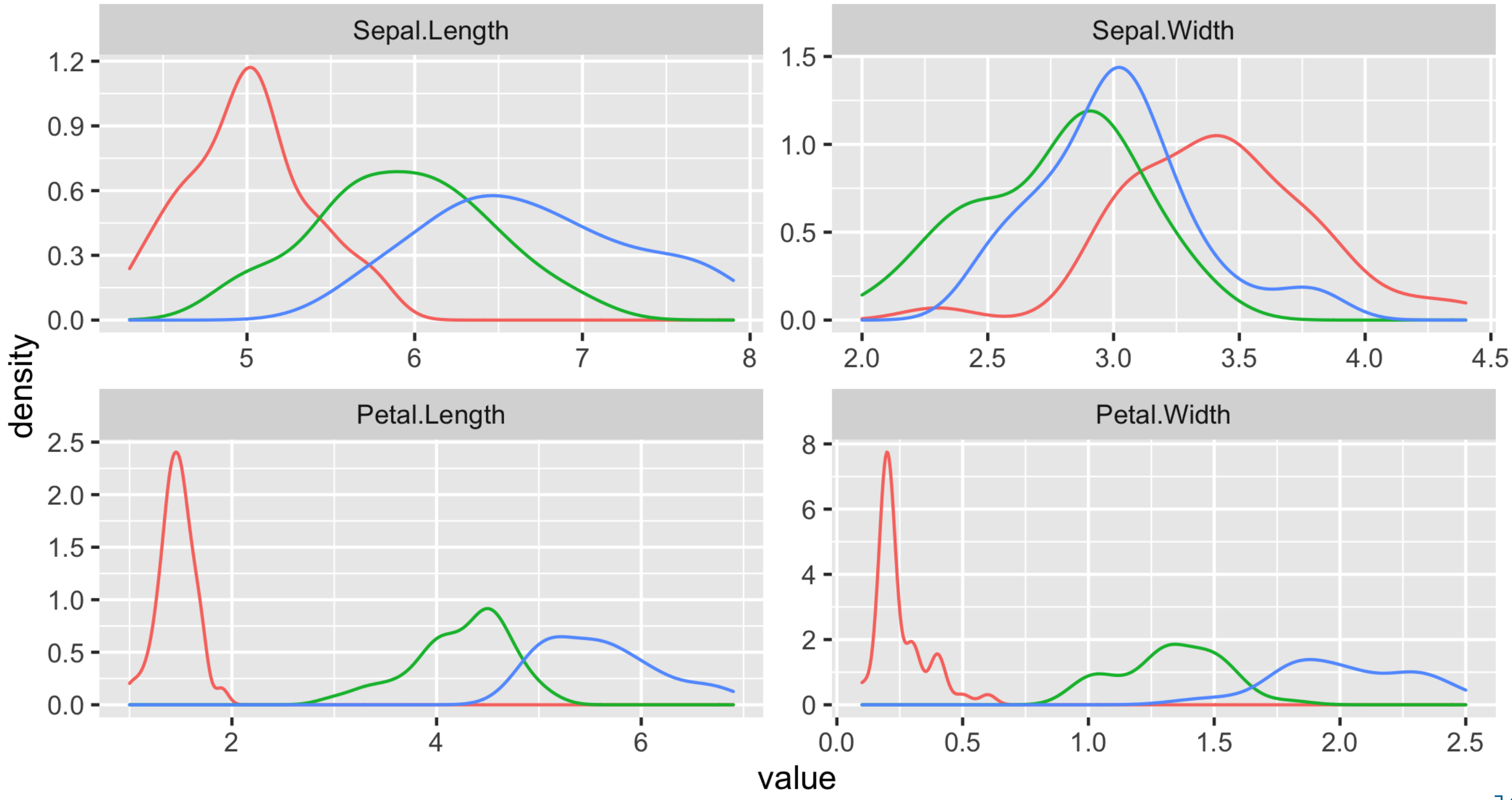
Gaussian Naive Bayes - Iris dataset

- Predict Species based on sepal length, sepal width, petal length and petal width.

```
set.seed(1234)
split <- createDataPartition(iris$Species, p = 0.8, list = FALSE)

train <- iris[split, ]
test <- iris[-split, ]

c(nrow(train), nrow(test)) # print number of observations in test vs. t

table(train$Species) %>% prop.table() # Proportions of Species
```

Gaussian Naive Bayes - Iris dataset

- Checking assumption of gaussian distribution
- Are data normally distributed? Do we need to transform? Or should we use non-parametric kernel density distribution instead? (Actually we will try both by tuning the model)



Gaussian Naive Bayes - Iris dataset

- First, fit a model with 10 fold cross validation

```r
predictors <- names(train)[1:4] #Create response and predictor data

x <- train[,predictors] #predictors
y <- train$Species #response

train_control <- trainControl(method = "cv", number = 10) #set up 10 fo

species_mod1 <- train( #train the model
  x = x,
  y = y,
  method = "nb",
  trControl = train_control
)

confusionMatrix(species_mod1) #results of model on training data
```

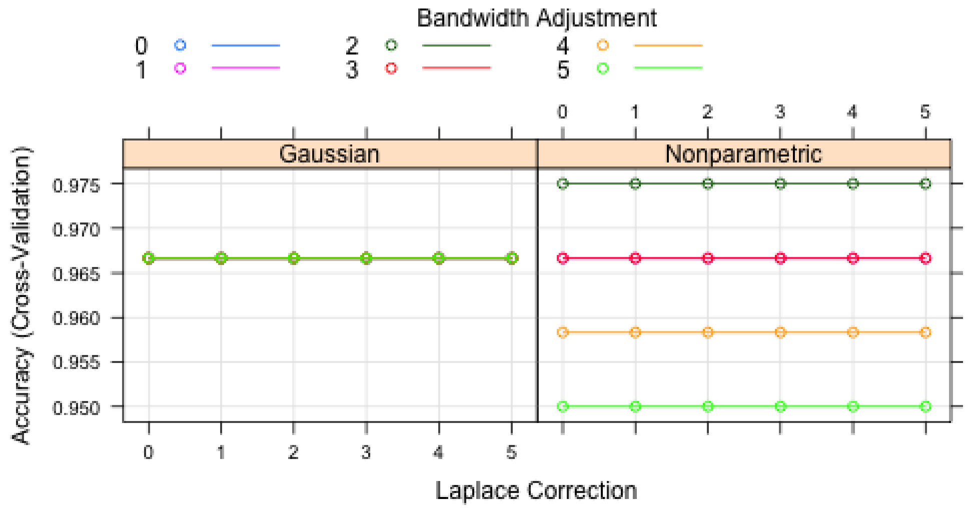
Gaussian Naive Bayes - Iris dataset

- Can we do any better?
- Tuning parameters = usekernal, adjust, fL

```
tune_params <- expand.grid( #define tuning parameters
  usekernel = c(TRUE, FALSE),
  fL = 0:5,
  adjust = 0:5
)

# train model
species_mod2 <- train(
  x = x,
  y = y,
  method = "nb",
  trControl = train_control,
  tuneGrid = tune_params
  )
```

Gaussian Naive Bayes - Iris dataset

```
plot(species_mod2)
```



- Laplace smoother makes no difference
- Non parametric Kernel with Bandwidth = 2 is best

Gaussian Naive Bayes - Iris dataset

- Results for best model

```
confusionMatrix(species_mod2) #results of model on training data
                              # we have improved accuracy a tiny bit

pred <- predict(species_mod2, newdata = test) #make predictions

confusionMatrix(pred, test$Species) #assessing the classifier
```

Overcoming randomness of splitting data

- Split data into test and training 10 times, and perform tuning & cross validation within each one

```
n <- nrow(iris) #number of observations in iris

splits <- createDataPartition(iris$Species, p = 0.8, list = FALSE, times
train_control <- trainControl(method = "cv", number = 10) #set up 10 fo
tune_params <- expand.grid( #define tuning parameters
  usekernel = c(TRUE, FALSE),
  fL = 0:5,
  adjust = 0:5
)
```

Overcoming randomness of splitting data

- Split data into test and training 10 times, and perform tuning & cross validation within each one

```r
all_mods <- lapply(seq_len(ncol(splits)), function(i){

  train <- iris[splits[,i], ] #split into i'th  test and train
  test <- iris[-splits[,i], ]

  predictors <- names(train)[1:4] #Create response and predictor data

  x <- train[,predictors] #predictors
  y <- train$Species #response

  species_mod <- train( #fit models with tuning and cv
      x = x,
      y = y,
      method = "nb",
      trControl = train_control,
      tuneGrid = tune_params
      )

  pred <- predict(species_mod, newdata = test) #make predictions

  confusionMatrix(pred, test$Species)$overall["Accuracy"] #assessing th
})
```

Overcoming randomness of splitting data

- Split data into test and training 10 times, and perform tuning & cross validation within each one

```
mean(unlist(all_mods))
sd(unlist(all_mods))
```

Naive Bayes Pros and Cons

Pros

- Simple
- Fast
- Scales well

Cons

- Assumes independence of variables
- Assumes all variables are equally important
- Not as accurate as other methods e.g. Random Forests

Extra reading/listening

Confused? Go here:

- A great conceptual overview StatQuest

Feeling confident? Try this:

- Use the caret *preProc* argument in train() to see if preprocessing the iris dataset improves model accuracy
- Try using the h2o package instead

Just for fun:

- Listen to this Data Skeptic podcast

Important!

- Accuracy is not the best or only test for algorithm performance, explore more!

References

- UC Business Analytics Tutorial
- Zhang, 2016