Learning from Big Data: Tutorial 1.1

Tayyip Altan

September 2024

Introduction

This tutorial illustrates the application of the Naive Bayes Classifier (NBC) and VADER using the review data. Furthermore, we will also be covering performance measures using a confusion matrix.

1. Loading libraries

Before starting the tutorial, make sure you have all the required libraries properly installed. Simply run this chunk of code below.

```
# Required packages. P_load ensures these will be installed and loaded.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tm, nnet, dplyr, tidyr, ggplot2, reshape2,latex2exp, vader, caret, png, knitr)
```

2. Load the reviews and prepare the data

```
# Load the review data. Note that we are now using the fileEncoding parameter when
#      calling read.csv() - this helps reading the review text correctly for further
#      processing (by correctly interpreting the non-ASCII symbols)
reviews_raw <- read.csv('Reviews_tiny.csv', fileEncoding="ISO-8859-1")

reviews_raw <- reviews_raw %>%
               select(movie_name,review_code,   reviewer,   review_date, num_eval,
                      prob_sentiment,words_in_lexicon_sentiment_and_review, ratio_helpful,
                      raters,
                      prob_storyline,   prob_acting,    prob_sound_visual,
                      full_text,    processed_text,
                      release_date, first_week_box_office,MPAA, studio, num_theaters )
```

The code chunk below loads training data from three content lexicons related to storyline, acting, and visual aspects. Furthermore, content/sentiment likelihood data is loaded. Finally, the parameters capturing the priors for our NBC models are defined

```
# training data
dictionary_storyline <- read.csv2("storyline_33k.txt")
dictionary_acting <- read.csv2("acting_33k.txt")
```

```r
dictionary_visual <- read.csv2("visual_33k.txt")

# Load fake likelihoods data
likelihoods <- read.csv("example_100_fake_likelihood_topic.csv")

lexicon_content  <- as.character(likelihoods[ ,1])

# Load fake likelihoods data
likelihoods_sentim <-read.csv2("example_100 fake_likelihood_sentiment.csv",header=TRUE,
                               sep=",", quote="\"",dec=".",fill=FALSE)

lexicon_sentiment <-  as.character(likelihoods_sentim$words)

# Set the parameters
prior_topic <- 1/3
prior_sent  <- 1/2
total_reviews <- nrow(reviews_raw)
```

3. Supervised Learning: Naive Bayes Classifier (NBC)

Below, the functions Compute_posterior_sentiment and Compute_posterior_content are displayed. These functions apply the Bayes rule, which was covered in the lecture, to calculate posteriors. The first function estimates the probability that the review expresses positive or negative sentiment, while the second function determines the probability that the review pertains to each specific topic.

Compute posterior sentiment function

```r
Compute_posterior_sentiment <- function(prior, corpus_in , dict_words, p_w_given_c,TOT_DIMENSIONS ){

  output <-  capture.output (word_matrix <-
                               inspect(DocumentTermMatrix(corpus_in,
                                                          control=list(stemming=FALSE,
                                                                       language = "english",
                                                            dictionary=as.character(dict_words)))))

  # Check if there are any relevant words in the review, if there are, treat them. if not, use prior
  if (sum(word_matrix) == 0) {
    posterior<-prior ; words_ <- c("")
  } else{

    # Positions in word matrix that have words from this review
    word_matrix_indices <- which(word_matrix>0)
    textual_words_vec   <- colnames(word_matrix)[word_matrix_indices]

    # Loop around words found in review
    WR <- length(word_matrix_indices) ;word_matrix_indices_index=1

    for (word_matrix_indices_index in 1: WR){
      word <- colnames(word_matrix)[word_matrix_indices[word_matrix_indices_index]]
      p_w_given_c_index <- which(as.character(p_w_given_c$words) == word)
```

```
      # Loop around occurrences  | word
      occ_current_word <- 1
      for (occ_current_word in 1: word_matrix[1,word_matrix_indices[word_matrix_indices_index]])
      {
        # initialize variables
        posterior     <- c(rep(0, TOT_DIMENSIONS))
        vec_likelihood <- as.numeric(c(p_w_given_c$pos_likelihood[p_w_given_c_index],
                                      p_w_given_c$neg_likelihood[p_w_given_c_index]))

        # positive - this is the first element in the vector
        numerat         <-  prior[1] *  as.numeric(p_w_given_c$pos_likelihood[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[1]    <- numerat / denomin

        # negative - this is the second element in the vector
        numerat         <-  prior[2] *  as.numeric(p_w_given_c$neg_likelihood[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[2]    <- numerat /  denomin


        if (sum(posterior)>1.01) {
          ERROR <- TRUE
        }

        prior <- posterior
      } # close loop around occurrences
    } # close loop around words in this review
    words_ <-  colnames(word_matrix)[word_matrix_indices]
  } # close if review has no sent words

  return(list(posterior_=posterior, words_=words_) )
}
```

Compute posterior content function

```
Compute_posterior_content <- function(prior,  word_matrix, p_w_given_c , BIGRAM, TOT_DIMENSIONS){

  # Check if there are any relevant words in the review, if there are, treat them.If not, use prior
  if (sum(word_matrix) == 0) {
    posterior<-prior
  } else{

    # Positions in word matrix that have words from this review
    word_matrix_indices <- which(word_matrix>0)
    textual_words_vec   <- colnames(word_matrix)[word_matrix_indices]

    # Loop around words found in review
    WR <- length(word_matrix_indices) ;word_matrix_indices_index=1

    for (word_matrix_indices_index in 1: WR) {
      word <- colnames(word_matrix)[word_matrix_indices[word_matrix_indices_index]]
```

```r
      p_w_given_c_index <- which(as.character(p_w_given_c$words) == word)

      # Loop around occurrences  | word
      occ_current_word <- 1
      for (occ_current_word in 1:word_matrix[1,word_matrix_indices[word_matrix_indices_index]])
      {
        # initialize variables
        posterior     <- c(rep(0, TOT_DIMENSIONS))
        vec_likelihood <-as.numeric(c(p_w_given_c$storyline[p_w_given_c_index],
                                      p_w_given_c$acting[p_w_given_c_index],
                                      p_w_given_c$visual[p_w_given_c_index]) )

        # storyline - this is the first element in the vector
        numerat         <-  prior[1] *  as.numeric(p_w_given_c$storyline[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[1]    <-  numerat /  denomin

        # acting - this is the second element in the vector
        numerat         <-  prior[2] *  as.numeric(p_w_given_c$acting[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[2]    <-  numerat /  denomin

        # visual - this is the third element in the vector
        numerat         <-  prior[3] *  as.numeric(p_w_given_c$visual[p_w_given_c_index])
        denomin         <-  prior %*% vec_likelihood
        posterior[3]    <-  numerat /  denomin

        if (sum(posterior)>1.01) {
          ERROR <- TRUE
        }
        prior <- posterior
      } # close loop around occurrences
    } # close loop around words in this review

  } # close if review has no sent words

  return (posterior_= posterior  )
}
```

NBC Sentiment Analysis Loop

Now that we have defined the functions for calculating posteriors, we can loop over the reviews and apply these functions to determine the sentiment and content posteriors for each review. Using the 'Compute_posterior_sentiment' function we defined, we can calculate the posteriors for sentiment for each review using NBC.

```r
# Loop over each review
for (review_index in 1:total_reviews) {

  # Print progress every 100th review
  if (review_index %% 100 == 0) {
    cat("Computing content of review #", review_index, "\n", sep="")
  }
```

```r
  # If the review is not empty, continue and calculate posterior
  if ( reviews_raw$full_text[review_index] != ""){

    # Assign the processed text of the non-empty review to text_review
    text_review    <- as.character(reviews_raw$processed_text[review_index])

    # Reset the prior every iteration as each review is looked at separately
    prior_sent_reset <- c(prior_sent, 1 - prior_sent)

    # Pre-process the review to remove punctuation marks and numbers.
    # Note that we are not removing stopwords here (nor elsewhere - a point for improvement)
    corpus_review <- tm_map(tm_map(VCorpus(VectorSource(text_review)), removePunctuation),
                            removeNumbers)

    # Compute posterior probability the review is positive
    TOT_DIMENSIONS <- 2
    sent.results <- Compute_posterior_sentiment(prior = prior_sent_reset,
                                                corpus_in  = corpus_review,
                                                dict_words = lexicon_sentiment,
                                                p_w_given_c = likelihoods_sentim,
                                                TOT_DIMENSIONS)

    words_sent  <- sent.results$words_
    posterior_sent <- sent.results$posterior_
    reviews_raw$prob_sentiment[review_index] <- posterior_sent[1]
    reviews_raw$words_in_lexicon_sentiment_and_review[review_index] <-paste(words_sent,collapse =" ")
  }
}
```

NBC Content Analysis Loop

We also calculate the posteriors for the content each review using NBC.

```r
# Loop over each review
for (review_index in 1: total_reviews) {

  # Print progress every 100th review
  if (review_index %% 100 == 0) {
    cat("Computing content of review #", review_index, "\n", sep="")
  }

  # If the review is not empty, continue and calculate posterior
  if ( reviews_raw$full_text[review_index]!=""){

    # Assign the processed text of the non-empty review to text_review
    text_review   <- reviews_raw$processed_text[review_index]

    # Pre-process the review to remove numbers and punctuation marks.
    # Note that we are not removing stopwords here (nor elsewhere - a point for improvement)

    # put in corpus format and obtain word matrix
    corpus_review <- VCorpus(VectorSource(text_review))
```

```
    output <-capture.output(content_word_matrix <-
                              inspect(DocumentTermMatrix(corpus_review,
                                        control = list(stemming=FALSE,
                                                      language = "english",
                                                      removePunctuation=TRUE,
                                                      removeNumbers=TRUE,
                                                 dictionary=as.character(lexicon_content)))))

    # Compute posterior probability the review is about each topic
    TOT_DIMENSIONS <- 3
    posterior <- Compute_posterior_content(prior= matrix(prior_topic, ncol=TOT_DIMENSIONS),
                                          content_word_matrix,
                                          p_w_given_c=likelihoods,
                                          TOT_DIMENSIONS)

    # Store the posteriors
    reviews_raw$prob_storyline[review_index]    <- posterior[1]
    reviews_raw$prob_acting[review_index]       <- posterior[2]
    reviews_raw$prob_sound_visual[review_index] <- posterior[3]
  }
}
Processed_reviews <- reviews_raw
View(Processed_reviews)

# Saves the updated file, now including the sentiment and content/topic posteriors.
# write.csv(Processed_reviews,"TestProcessed_reviews.csv" , row.names = FALSE )
```

4. Supervised Learning: VADER

Next, we use VADER, a lexicon-based sentiment analysis tool, to assess the sentiment of our reviews. Following a similar approach to the Naive Bayes Classifier, we calculate the sentiment for each review and add the results to our dataframe.

```
# Loop over each review
for (review_index in 1:total_reviews) {

  # Print progress every 100th review
  if (review_index %% 100 == 0) {
    cat("Computing VADER sentiment of review #", review_index, "\n", sep="")
  }

  # If the review is not empty, continue and apply VADER
  if (reviews_raw$full_text[review_index] != ""){

    # Assign the processed text of the non-empty review to text_review
    # Note that we have not removed punctuation, numbers, and stopwords (a point for improvement)
    text_review   <- as.character(reviews_raw$processed_text[review_index])

    # Apply VADER
    vader <- get_vader(text_review)

    # store the VADER results in the dataframe
```

```r
    reviews_raw$vader_pos[review_index] <- vader[["pos"]]
    reviews_raw$vader_neg[review_index] <- vader[["neg"]]
    reviews_raw$vader_neu[review_index] <- vader[["neu"]]
    reviews_raw$vader_compound[review_index] <- vader[["compound"]]
  }


}
Processed_reviews <- reviews_raw
View(Processed_reviews)
#write.csv(Processed_reviews,"VADER_Processed_reviews.csv" , row.names = FALSE )
```

5. Performance Measurement: Confusion matrix

A confusion matrix is a valuable tool for evaluating classification models. A confusion matrix helps us compare our model's predictions with the true values by summarizing the counts of correct and incorrect predictions across different classes. In classification problems, we typically focus on the positive class—the one we're interested in predicting—and the negative class, which represents all other outcomes. Below, you'll find an example of a confusion matrix.

		Actual Class	
		Positive (**P**)	Negative (**N**)
Predicted Class	Positive (**P**)	True Positive (**TP**)	False Positive (**FP**)
	Negative (**N**)	False Negative (**FN**)	True Negative (**TN**)

A confusion matrix consists of:

1. True Positives (TP): Correctly predicted positive cases.
2. True Negatives (TN): Correctly predicted negative cases.
3. False Positives (FP): Incorrectly predicted as positive (Type I error).
4. False Negatives (FN): Incorrectly predicted as negative (Type II error).

Using a confusion matrix we can calculate metrics to calculate the performance of our classification model. A commonly used metric is the specificity The formula for the specificity is given below. A more comprehensive overview of metrics can be found here: https://en.wikipedia.org/wiki/Confusion_matrix#Table_of_confusion

$$\text{Specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$

Below is an example of how to use a confusion matrix in R and calculate specificity:

```r
# Artificial example with actual and predicted values for a classification problem with classes 1 and 0
actual <- factor(c(1, 0, 1, 1, 0, 0, 1, 0, 0, 1), levels = c(1, 0))
predicted <- factor(c(1, 0, 0, 1, 0, 0, 1, 1, 0, 0), levels = c(1, 0))

# Create a confusion matrix
conf_matrix <- confusionMatrix(predicted, actual)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 0
##          1 3 1
##          0 2 4
##
##                Accuracy : 0.7
##                  95% CI : (0.3475, 0.9333)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.1719
##
##                   Kappa : 0.4
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.6000
##             Specificity : 0.8000
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.6667
##              Prevalence : 0.5000
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.4000
##       Balanced Accuracy : 0.7000
##
##        'Positive' Class : 1
##
```

```r
# Calculating the specificity
specificity(predicted, actual)
```

```
## [1] 0.8
```

You might have noticed that our supervised models above do not output class labels, but rather a probability. At this stage, we distinguish between soft predictions and hard predictions. Soft predictions are the probabilities that an observation belongs to the positive class, while hard predictions are the final class labels assigned to the observations. To convert soft predictions into hard predictions, we use a decision rule, such as applying a threshold or selecting the class with the highest probability.