stor390final-test

Anna Fetter

2024-11-21

step 1. Load dataset & necessary packages

```
library(e1071)
#libraries for data preprocessing
library(tm)
library(SnowballC)
library(ggplot2)
#install LIME, used for explainable AI
library(lime)
library(e1071)
library(caret)
library(dplyr)
library(tidyr)
library(viridis)  # For better color palettes
library(scales)


#load data
WELFake <- read.csv("WELFake_Dataset.csv")
```

Step 2: preprocess dataset

```
preprocess_text <- function(text) {
  #convert text to all lowercase
  text <- tolower(text)
  #remove punctuation
  text <- removePunctuation(text)
  #remove numbers
  text <- removeNumbers(text)
  #remove stopwords
  text <- removeWords(text, stopwords("en"))
  text <- stripWhitespace(text)
  # Perform stemming
  text <- wordStem(text, language = "en")
  return(text)
}

# Apply preprocessing function
```

```r
WELFake$preprocessed_text <- sapply(WELFake$text, preprocess_text)



# Create side-by-side comparison of original vs preprocessed text
data.frame(
  Type = c("Original", "Preprocessed"),
  Text = c(
    substr(WELFake$text[1], 1, 300),
    substr(WELFake$preprocessed_text[1], 1, 300)
  )
) %>%
  dplyr::mutate(
    Text = paste0(Text, "...")
  ) %>%
  print(
    row.names = FALSE,
    right = FALSE,
    width = 80
  )
```

```
##  Type
##  Original
##  Preprocessed
##  Text
##  No comment is expected from Barack Obama Members of the #FYF911 or #FukYoFlag and #BlackLivesMatter
##   comment expected barack obama members fyf fukyoflag blacklivesmatter movements called lynching hang
```

Step 3: set testing/training partition

```r
# Set a seed for reproducibility
set.seed(123)

# Shuffle the data to ensure randomness
shuffled_indices <- sample(1:nrow(WELFake))

# Define the split ratio (e.g., 80% for training, 20% for testing)
split_ratio <- 0.8
train_size <- floor(split_ratio * nrow(WELFake))

# Create training and testing indices
train_indices <- shuffled_indices[1:train_size]
test_indices <- shuffled_indices[(train_size + 1):nrow(WELFake)]

# Subset the data for training and testing
train_data <- WELFake[train_indices, ]
test_data <- WELFake[test_indices, ]

# Inspect the split
cat("Training set size:", nrow(train_data), "\n")
```

```
## Training set size: 57707
```

```r
cat("Testing set size:", nrow(test_data), "\n")
```

```
## Testing set size: 14427
```

```r
# Optional: Check class distribution in both sets
cat("Class distribution in training set:\n")
```

```
## Class distribution in training set:
```

```r
print(table(train_data$Class_Label))
```

```
## < table of extent 0 >
```

```r
cat("Class distribution in testing set:\n")
```

```
## Class distribution in testing set:
```

```r
print(table(test_data$Class_Label))
```

```
## < table of extent 0 >
```

step 4: train the SVM, I think I might need a document term matrix? since SVM needs numbers?, matrix that reports relatively frequency of reported terms

```r
# Create a Corpus, Corpus is a term in data science that refers to a large collection of text used to t
corpus <- Corpus(VectorSource(WELFake$preprocessed_text))

# Generate a Document-Term Matrix, counts frequency of preprocessed word use
dtm <- DocumentTermMatrix(corpus)


# Check the dimensions of the DTM
dim(dtm)
```

```
## [1]  72134 427468
```

```r
#remove infrequent terms (must appear in 5% of dataset, for sake of efficiency, was taking wayyy too lo
dtm <- removeSparseTerms(dtm, 0.95)

# Check the dimensions of the DTM to see how much this shrunk the data
dim(dtm)
```

```
## [1] 72134   761
```

```r
#convert DTM to matrix, since SVM requires numerical matrix for input
dtm_matrix <- as.matrix(dtm)

# Check that the number of rows in the DTM matches the number of labels
if (nrow(dtm_matrix) != length(WELFake$label)) {
  stop("Number of documents in DTM does not match the number of class labels!")
}

# Combine the DTM with the label column
dtm_data <- data.frame(dtm_matrix, label = WELFake$label)

# Inspect the combined dataset
# head(dtm_data)
```

```r
# Apply the training and testing partition to the DTM data
train_x <- dtm_data[train_indices, -ncol(dtm_data)]  # All columns except the last (features)
train_y <- as.factor(dtm_data[train_indices, ncol(dtm_data)])  # The last column (labels)

test_x <- dtm_data[test_indices, -ncol(dtm_data)]
test_y <- as.factor(dtm_data[test_indices, ncol(dtm_data)])

# Determine the sample size (ensure it doesn't exceed the available rows)
sample_size <- min(7000, nrow(train_x))

# Randomly sample row indices
sample_indices <- sample(1:nrow(train_x), size = sample_size, replace = FALSE)

# Subset the training data based on the sampled indices
train_x_sampled <- train_x[sample_indices, ]
train_y_sampled <- train_y[sample_indices]

# Train the SVM model on the sampled data
svm_model <- svm(x = train_x_sampled, y = train_y_sampled, kernel = "linear", cost = 1, probability = TR

# Inspect the trained model
summary(svm_model)
```

```
##
## Call:
## svm.default(x = train_x_sampled, y = train_y_sampled, kernel = "linear",
##     cost = 1, probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  859
##
##  ( 431 428 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# Predict on the test set
predictions <- predict(svm_model, test_x)

# Create a confusion matrix
svm_confusion <- table(Predicted = predictions, Actual = test_y)
print(svm_confusion)
```

```
##          Actual
## Predicted    0    1
##         0 6286  641
##         1  639 6861
```

```
# Calculate accuracy
svm_accuracy <- sum(diag(svm_confusion)) / sum(svm_confusion)
cat("Accuracy:", svm_accuracy, "\n")
```

```
## Accuracy: 0.9112775
```

```
# Precision
svm_precision <- diag(svm_confusion) / rowSums(svm_confusion)

# Recall
svm_recall <- diag(svm_confusion) / colSums(svm_confusion)

# F1-Score
svm_f1_score <- 2 * (svm_precision * svm_recall) / (svm_precision + svm_recall)

# Print metrics
cat("Precision:", svm_precision, "\n")
```

```
## Precision: 0.9074635 0.9148
```

```
cat("Recall:", svm_recall, "\n")
```

```
## Recall: 0.9077256 0.9145561
```

```
cat("F1-Score:", svm_f1_score, "\n")
```

```
## F1-Score: 0.9075946 0.914678
```

step 5: integrate FastText, using text package & downloading fasttext locally on my machine

Here's the Python version (read-only, from Facebook): https://github.com/facebookresearch/fastText

```r
# Load the package
library(text)

prepare_fasttext_file <- function(data, text_col, label_col, output_file) {
  # Format data as "__label__<label> <text>"
  formatted_data <- paste0("__label__", data[[label_col]], " ", data[[text_col]])

  # Write to the specified output file
  writeLines(formatted_data, output_file)
}

# Generate the training file
prepare_fasttext_file(
  data = WELFake[train_indices, ],        # Subset of training data
  text_col = "preprocessed_text",         # Column with preprocessed text
  label_col = "label",                    # Column with labels
  output_file = "fasttext_train.txt"      # Training file name
)

# Generate the testing file
prepare_fasttext_file(
  data = WELFake[test_indices, ],         # Subset of testing data
  text_col = "preprocessed_text",
  label_col = "label",
  output_file = "fasttext_test.txt"       # Testing file name
)
```

```r
system("/Users/annafetter/fastText/fasttext supervised -input fasttext_train.txt -output fasttext_model
```

files created during this step: fasttext_model.bin is the binary model created during training, fasttext_test.txt is the test dataset in FastText format, predictions.txt is the output file where predictions are stored

```r
system("/Users/annafetter/fastText/fasttext predict fasttext_model.bin fasttext_test.txt > predictions.t

# Read predictions from the file
predictions <- readLines("predictions.txt")

# Preview predictions
head(predictions)
```

```
## [1] "__label__0" "__label__0" "__label__0" "__label__0" "__label__1"
## [6] "__label__0"
```

Create a confusion matrix based on predicted vs actual labels for the SVM

```r
actual_labels <- WELFake$label[test_indices]
# Create a confusion matrix
fasttext_confusion <- table(Predicted = predictions, Actual = paste0("__label__", actual_labels))
print(fasttext_confusion)
```

```
##             Actual
```

```
## Predicted    __label__0 __label__1
##   __label__0       6731        196
##   __label__1        194       7306
```

```
# Calculate accuracy
fasttext_accuracy <- sum(diag(fasttext_confusion)) / sum(fasttext_confusion)
cat("Accuracy:", fasttext_accuracy, "\n")
```

```
## Accuracy: 0.9729674
```

Generate additional metrics similar to those in presented in research paper

```
# Precision
fasttext_precision <- diag(fasttext_confusion) / rowSums(fasttext_confusion)

# Recall
fasttext_recall <- diag(fasttext_confusion) / colSums(fasttext_confusion)

# F1-Score
fasttext_f1_score <- 2 * (fasttext_precision * fasttext_recall) / (fasttext_precision + fasttext_recall)

# Print metrics
cat("Precision:", fasttext_precision, "\n")
```

```
## Precision: 0.9717049 0.9741333
```

```
cat("Recall:", fasttext_recall, "\n")
```

```
## Recall: 0.9719856 0.9738736
```

```
cat("F1-Score:", fasttext_f1_score, "\n")
```

```
## F1-Score: 0.9718452 0.9740035
```

include necessary libraries

```
#install LIME, used for explainable AI
library(lime)
library(e1071)
library(ggplot2)
library(caret)
library(dplyr)
library(tidyr)
library(viridis)  # For better color palettes
library(scales)
```

Step 6: Implement LIME

```r
# Load minimal packages
library(lime)
library(ggplot2)

# Define wrapper function
wrap_svm <- function(model) {
  class(model) <- c("wrapped_model", class(model))
  return(model)
}

# Wrap existing SVM model
wrapped_svm <- wrap_svm(svm_model)

# Define required methods
model_type.wrapped_model <- function(x, ...) "classification"
predict_model.wrapped_model <- function(x, newdata, ...) {
  preds <- attr(predict(x, newdata, probability = TRUE), "probabilities")
  return(as.data.frame(preds))
}

# Sample small subset of test data
set.seed(123)
test_sample_idx <- sample(1:nrow(test_x), 1000)
test_x_sample <- test_x[test_sample_idx, ]

# Create explainer
explainer <- lime(train_x, wrapped_svm)

# Generate explanations with reduced parameters
explanation <- lime::explain(
  x = test_x_sample,
  explainer = explainer,
  n_features = 3,
  n_labels = 1,
  n_permutations = 50      # Reduced from 500
)

# Plot results
# plot_features(explanation)

# Summarize feature importance
importance <- explanation %>%
  group_by(feature) %>%
  summarise(
    importance = mean(abs(feature_weight)),
    n = n()
  ) %>%
  arrange(desc(importance)) %>%
  head(10)

# print(importance)
```

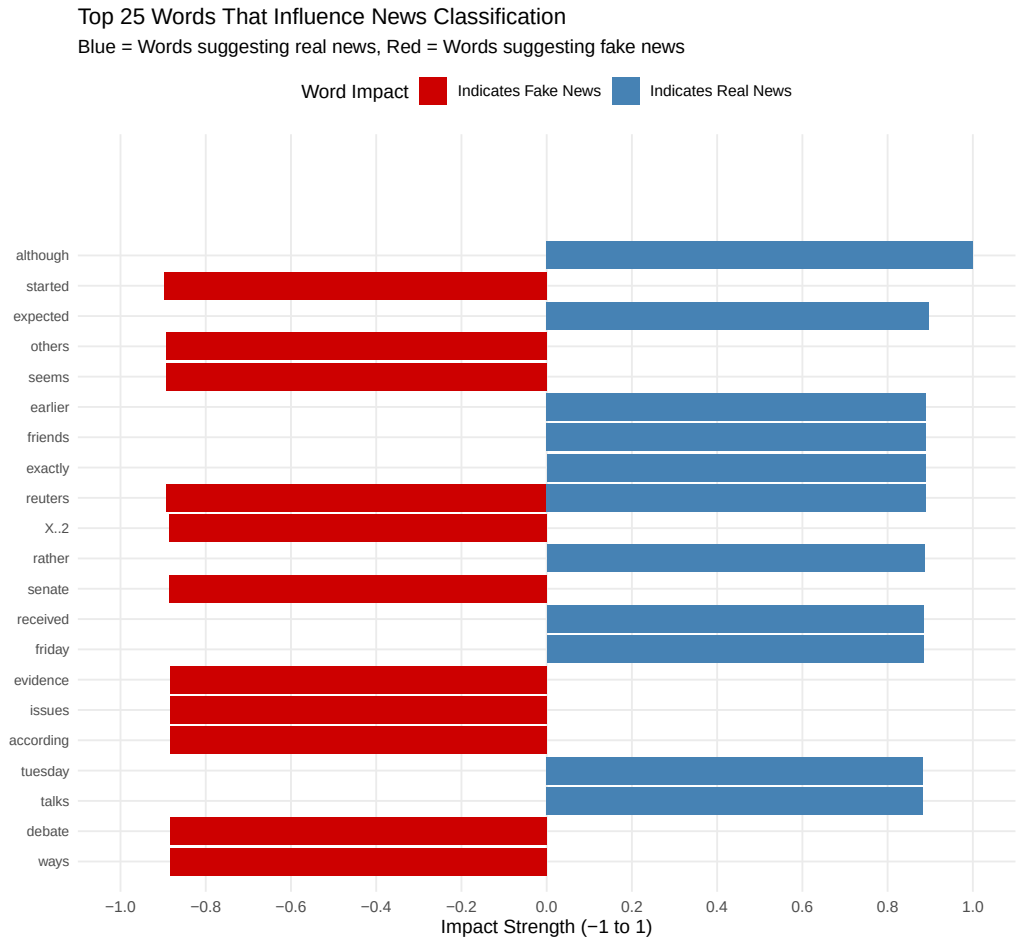
```r
# Create detailed importance table
importance_summary <- explanation %>%
  group_by(feature) %>%
  summarise(
    avg_impact = round(mean(feature_weight), 4),
    abs_impact = round(mean(abs(feature_weight)), 4),
    consistency = round(sd(feature_weight), 4),
    frequency = n()
  ) %>%
  mutate(
    direction = ifelse(avg_impact > 0, "Real News", "Fake News"),
    strength = case_when(
      abs_impact > 0.75 ~ "Very Strong",
      abs_impact > 0.5 ~ "Strong",
      abs_impact > 0.25 ~ "Moderate",
      TRUE ~ "Weak"
    )
  ) %>%
  arrange(desc(abs_impact)) %>%
  head(10) %>%
  select(
    Word = feature,
    `Impact Direction` = direction,
    `Impact Strength` = strength,
    `Average Impact` = abs_impact,
    `Times Used` = frequency
  )

# Print formatted table
print(importance_summary, n = 10, width = Inf)
```

```
## # A tibble: 10 x 5
##    Word     `Impact Direction` `Impact Strength` `Average Impact` `Times Used`
##    <chr>    <chr>              <chr>                        <dbl>        <int>
##  1 team     Real News          Very Strong                  0.867            1
##  2 business Fake News          Very Strong                  0.863            1
##  3 risk     Real News          Very Strong                  0.862            1
##  4 source   Fake News          Very Strong                  0.860            1
##  5 meeting  Real News          Very Strong                  0.858            1
##  6 X.m      Fake News          Very Strong                  0.856            1
##  7 four     Real News          Very Strong                  0.854            1
##  8 chief    Fake News          Very Strong                  0.853            2
##  9 didn.t   Real News          Very Strong                  0.853            1
## 10 decision Real News          Very Strong                  0.852            1
```

```r
# intuitive visualization of LIME
ggplot(head(arrange(explanation, desc(abs(feature_weight))), 25),
       aes(x = feature_weight / max(abs(feature_weight)), # Normalize to -1 to 1
           y = reorder(feature, abs(feature_weight)),
           fill = feature_weight > 0)) + # Color code by positive/negative
  geom_col() +
  theme_minimal() +
  theme(
```

```
  axis.text.y = element_text(size = 8),
  axis.text.x = element_text(angle = 0),
  plot.margin = margin(t = 20, r = 20, b = 20, l = 120),
  panel.grid.minor = element_blank(),
  legend.position = "top"
) +
scale_fill_manual(values = c("red3", "steelblue"),
                  labels = c("Indicates Fake News", "Indicates Real News")) +
labs(
  title = "Top 25 Words That Influence News Classification",
  subtitle = "Blue = Words suggesting real news, Red = Words suggesting fake news",
  x = "Impact Strength (-1 to 1)",
  y = NULL,
  fill = "Word Impact"
) +
scale_x_continuous(limits = c(-1, 1),
                   breaks = seq(-1, 1, 0.2)) +
scale_y_discrete(expand = expansion(mult = c(0.05, 0.2)))
```



```r
# Exploring misclassification examples
misclassified_examples <- data.frame(
  Actual = WELFake$label[test_indices],
  Predicted = as.numeric(gsub("__label__", "", predictions)),
  Text = substr(WELFake$text[test_indices], 1, 200)  # First 200 chars
) %>%
  # Keep only misclassified cases
  filter(Actual != Predicted) %>%
  # Split into false positives and false negatives
  mutate(
    Error_Type = case_when(
      Actual == 0 & Predicted == 1 ~ "Real News Classified as Fake",
      Actual == 1 & Predicted == 0 ~ "Fake News Classified as Real"
    )
  ) %>%
  # Get sample of each type
  group_by(Error_Type) %>%
  slice_head(n = 2) %>%
  select(Error_Type, Text) %>%
  mutate(
    Text = paste0(Text, "...")  # Add ellipsis to show truncation
  )

# Save to CSV
write.csv(misclassified_examples,
          "results/tables/misclassification_examples.csv",
          row.names = FALSE)
```

```r
# At end of first RMD (final-test-code.Rmd):

# Create directories
dir.create("results", showWarnings = FALSE)
dir.create("results/plots", showWarnings = FALSE)
dir.create("results/tables", showWarnings = FALSE)
dir.create("results/matrices", showWarnings = FALSE)

# Create metrics dataframe using existing FastText calculations
fasttext_metrics <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1-Score"),
  Score = c(fasttext_accuracy, fasttext_precision[1], fasttext_recall[1], fasttext_f1_score[1])
)

# Save confusion matrix
write.csv(as.data.frame.matrix(fasttext_confusion),
          "results/matrices/fasttext_confusion.csv")

# Save metrics
write.csv(fasttext_metrics, "results/tables/fasttext_metrics.csv")

# Create metrics dataframe using existing svm calculations
svm_metrics <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1-Score"),
  Score = c(svm_accuracy, svm_precision[1], svm_recall[1], svm_f1_score[1])
```

```
)

# Save confusion matrix
write.csv(as.data.frame.matrix(svm_confusion),
          "results/matrices/svm_confusion.csv")

# Save metrics
write.csv(svm_metrics, "results/tables/svm_metrics.csv")

# Save LIME plot
ggsave(
  "results/plots/lime_features.pdf",
  plot = last_plot(),
  width = 10,
  height = 8,
  dpi = 300
)

# Save text preprocessing example
write.csv(
  data.frame(
    Original = substr(WELFake$text[1], 1, 300),
    Preprocessed = substr(WELFake$preprocessed_text[1], 1, 300)
  ),
  "results/tables/text_preprocessing_example.csv"
)

write.csv(importance_summary,
          "results/tables/lime_importance_summary.csv",
          row.names = FALSE)
```