Improving Online Ad Performance by Identifying Relevant Customer Content for Farming Community

Project Summary

This project focuses on helping a farming community website automatically identify which online advertisements are useful and which are not. Since anyone—including automated bots—can post ads, the website often receives spam, fraudulent, or irrelevant advertisements that do not serve the farming community's needs.

To solve this problem, a dataset containing **4,143 classified ads** was used. Each ad was already labeled as either **relevant** or **not relevant**. The text from these ads was cleaned and prepared so that meaningful words could be analyzed properly.

Two machine learning models were built and tested:

- **Naive Bayes**, a simple and fast method commonly used for text classification.
- **Support Vector Machine (SVM)**, a more advanced model that separates relevant and non-relevant ads based on patterns in the text.

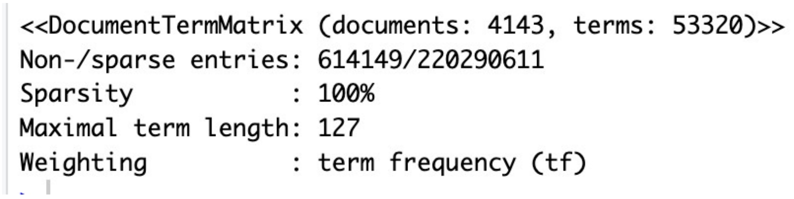
Both models were trained using most of the data and then tested on unseen ads to check how accurately they could classify new advertisements

.

1) Open the dataset FarmAds.csv and reviewing and describing the dataset

- doc_id- A unique identifier for each document (data type: integer)
- text-A text field containing phrases related to farm ads, medical terms, or symptoms (data type: object).
- Label- An integer label, likely representing categories (-1 and 1). The mean label value is close to 0, indicating a nearly equal distribution between -1 and 1.
- The dataset is fully populated with no missing values in any column. 2) Preprocess the data in R, and create a term-document matrix. (10)

- **Loading and creating corpus:** First, the text data is loaded into R, and a corpus is created. A corpus is essentially a structured collection of text documents, allowing us to apply text-processing functions consistently across all documents.
- **Tokenization**: It involves breaking down the text into individual words or "tokens." Before tokenizing, we prepare the text by removing extra whitespace and punctuation, and by converting all characters to lowercase. These steps ensure that words are consistently formatted, so variations (like "Farm" vs. "farm") are treated as the same token.
- **Stemming:** Stopwords are common words like "the," "and," or "is" that don't carry much unique meaning. Removing these words reduces noise in the data and focuses on terms that are likely to have more relevance to the analysis.
- **DTM:** Stemming reduces words to their base form by removing prefixes or suffixes. For instance, "farming" and "farmer" would both be reduced to "farm." This helps group similar words together and reduces the number of unique words in the dataset.
- **TF-IDF:** A Document-Term Matrix is created, where each row represents a document, and each column represents a unique word in the corpus. The values in the matrix indicate how frequently each word appears in each document.
- **TDM:** Finally, a Term-Document Matrix is created, which is like the DTM but transposed. Each row now represents a unique term, and each column represents a document. This format is useful for analyzing words across documents.
- **After preprocessing the text**


```
<<DocumentTermMatrix (documents: 4143, terms: 53320)>>
Non-/sparse entries: 614149/220290611
Sparsity           : 100%
Maximal term length: 127
Weighting          : term frequency (tf)
```

3) Preparing the training and test datasets.

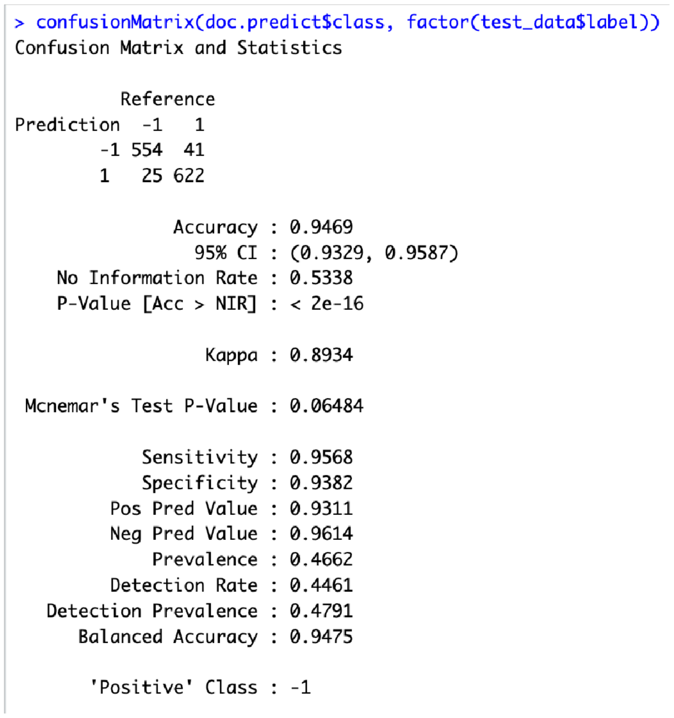
- Data Cleaning: Remove duplicates, handle missing values, and preprocess text (e.g., remove punctuation, lowercase).
- Feature Engineering: Create relevant features (e.g., a Document-Term Matrix for text) and convert categorical variables to numeric (e.g., one-hot encoding).
- Set an 80-20 Split Ratio: Use 80% of the data for training and 20% for testing.

- Shuffle Data: Randomly shuffle the dataset to ensure balanced distribution.
- Split: Separate the data into training (80%) and testing (20%) sets, including both text and labels.

4) Developing a model to classify the documents as "relevant" or "non-relevant" using NaiveBayes.

To classify documents as "relevant" or "non-relevant," a Naive Bayes model can be used due to its effectiveness with text data and simplicity. The model calculates the probability of each document belonging to either class based on feature probabilities. Its efficacy can be evaluated using metrics like accuracy, sensitivity, and specificity, which assess how well the model correctly classifies relevant and non-relevant documents.

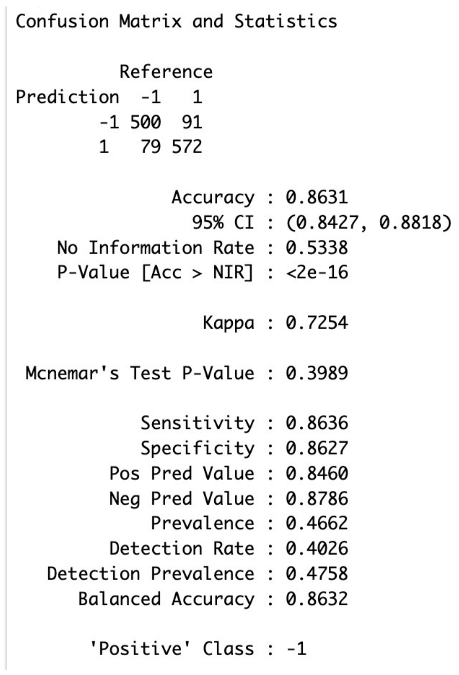
The Naive Bayes model for classifying documents as "relevant" or "non-relevant" is highly effective, achieving an accuracy of 94.69% and a strong Kappa of 0.8934, indicating excellent agreement with true labels. The high sensitivity (95.68%) and specificity (93.82%) showiing it accurately identifies both classes, making it a reliable classifier for this task.



```
> confusionMatrix(doc.predict$class, factor(test_data$label))
Confusion Matrix and Statistics

          Reference
Prediction  -1   1
        -1 554  41
        1   25 622

               Accuracy : 0.9469
                 95% CI : (0.9329, 0.9587)
    No Information Rate : 0.5338
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8934

 Mcnemar's Test P-Value : 0.06484

            Sensitivity : 0.9568
            Specificity : 0.9382
         Pos Pred Value : 0.9311
         Neg Pred Value : 0.9614
             Prevalence : 0.4662
         Detection Rate : 0.4461
   Detection Prevalence : 0.4791
      Balanced Accuracy : 0.9475

       'Positive' Class : -1
```

5) Developing a model to classify the documents as "relevant" or "non-relevant" using SVMs.

- Accuracy: The percentage of correctly classified documents.
- Precision, Recall, F1-Score: These metrics are particularly useful in cases where the class distribution is imbalanced, helping to evaluate the model's ability to correctly identify relevant vs. non-relevant documents.
- Cross-validation: Use cross-validation techniques to ensure the model generalizes well to unseen data.
- The SVM model performs well, achieving an accuracy of 87.6% with a balanced accuracy of 87.58%, showing strong classification ability. The Kappa of 0.751 suggests good agreement beyond chance. Sensitivity (87.22%) and specificity (87.93%) shows the balanced performance in identifying both relevant and non-relevant classes effectively.



```
Confusion Matrix and Statistics

          Reference
Prediction  -1   1
        -1 500  91
        1   79 572

               Accuracy : 0.8631
                 95% CI : (0.8427, 0.8818)
    No Information Rate : 0.5338
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7254

 Mcnemar's Test P-Value : 0.3989

            Sensitivity : 0.8636
            Specificity : 0.8627
         Pos Pred Value : 0.8460
         Neg Pred Value : 0.8786
             Prevalence : 0.4662
         Detection Rate : 0.4026
   Detection Prevalence : 0.4758
      Balanced Accuracy : 0.8632

       'Positive' Class : -1
```

Conclusion:

This project shows that computers can be taught to **automatically filter online ads**, saving time and effort for website owners. Instead of manually checking every advertisement, the system can quickly decide whether an ad is useful for farmers or should be ignored.

Among the two approaches tested, the **Naive Bayes model performed exceptionally well**, correctly identifying ads almost **95% of the time**. This means it is both reliable and efficient for real-world use. The SVM model also worked well but was slightly less accurate.

Overall, the project proves that using data and simple machine learning techniques can greatly improve the quality of online content. For a farming website, this means **less spam, more relevant ads, and a better experience for users**, without requiring technical knowledge from the people running the site.

CODES:

```
## Required Libraries library(tm)
library(SnowballC)
doc<- read.csv("/Volumes/WORK/Fall 2024/R /FarmAds.csv",  stringsAsFactors =
FALSE) summary(doc)

# convert sentences into a corpus corp <-
VCorpus(VectorSource(doc$text))
inspect(corp)



#Tokenizationand #Removing Stopword, punctuation # Stemming
corp <- tm_map(corp, stripWhitespace)     corp <- tm_map(corp,
removePunctuation) corp <- tm_map(corp,
content_transformer(tolower)) corp <- tm_map(corp, removeWords,
stopwords("english"))  corp <- tm_map(corp, stemDocument)
```

```
##DTM
dtm <- DocumentTermMatrix(corp) dtm
### Term Frequency--Inverse Document Frequency (TF-IDF) tfidf <-
weightTfIdf(dtm)

# Convert the DTM with TF-IDF  tdm <-
as.data.frame(as.matrix(tfidf))
tdm
tdm$label <-doc$label

# Convert the label column to a factor for classification
tdm$label <- factor(tdm$label) tdm$label

###Building Predictive models
library(caret) library(klaR)
tdm$label <- factor(tdm$label)
tdm$text <- factor(tdm$text)

# Split data into training and testing sets
set.seed(177) trainIndex <- createDataPartition(tdm$label, p = 0.7, list
= FALSE) train_data <- doc[trainIndex, ] test_data <- doc[-trainIndex, ]
train_data$label <- as.factor(train_data$label)
test_data$label <- as.factor(test_data$label)
##table distribution proportions(table(doc$label))
proportions(table(train_data$label))
proportions(table(test_data$label))

##NB modl using klar pckg doc.nb <- NaiveBayes(label ~ .,
data = train_data, fL=0.6)  doc.nb
```

```
# Predict on the test data doc.predict <-
predict(doc.nb, test_data) doc.predict

# Performance Evaluation for Naive Bayes confusionMatrix(doc.predict$class,
factor(test_data$label))



##SVM model
library(caret) library(kernlab)
library(e1071)

set.seed(177) train_data$label <-
factor(train_data$label) test_data$label <-
factor(test_data$label)

# Train the SVM model  doc.svm <- ksvm(label ~ ., data = train_data,
kernel = "vanilladot") doc.ksvm
# Predict on the test data svm_predictions <-
predict(doc.svm, test_data)

# Evaluate the performance
svm_confusion_matrix <- confusionMatrix(svm_predictions, test_data$label)
svm_confusion_matrix)
```