Predicting Tweet Location from Text using NLP and Machine Learning

Marcos Paulo Pazzinatto

Introduction

This project explores the possibility of predicting the approximate location (e.g., region or city) of a tweet based solely on its textual content. Using the publicly available Twitter Geospatial Data from the UCI Machine Learning Repository, we analyze millions of tweets that include geographic coordinates and other metadata.

The core goal is to apply Natural Language Processing (NLP) techniques and supervised machine learning to build models capable of inferring a tweet's origin, even in the absence of explicit geolocation. This has relevant applications in marketing, emergency response, public health, and social sciences.

We employ three classification models: a baseline Logistic Regression model, a more advanced Random Forest model, and an additional Support Vector Machine (SVM) classifier.

This report documents the full process from data preparation to model evaluation.

Data Preparation

The dataset is sourced from: Twitter Geospatial Data (UCI Machine Learning Repository)
- Link: https://archive.ics.uci.edu/dataset/1050/twitter+geospatial+data

The original `.zip` file contains another nested `.zip` (`twitter.zip`) that contains the final `twitter.csv`, with the core tweet data.

We preprocessed the dataset as follows: Download and unzip the dataset.Load the `twitter.csv` file. Filter and clean relevant columns (e.g., text, timestamp, coordinates). Remove rows with missing or invalid data.

This step ensures that our models work with clean, structured input. Here we add the libraries used in the project, so this process may take a while to complete.

```
### Install and load required packages

required_packages <- c("rstudioapi", "dplyr", "readr", "stringr", "fs", "httr", "utils", "tidymodels",

for (pkg in required_packages) {
  if (!require(pkg, character.only = TRUE)) {
    install.packages(pkg, dependencies = TRUE)
    library(pkg, character.only = TRUE)
  }
}

# Load additional required libraries
# Although I did the load above, here I leave the load explicit in an alternative way.
```

```r
library(scales)
library(dplyr)
library(readr)
library(stringr)
library(tidyr)
library(ggplot2)
library(tidymodels)

### Set working directory to the path where this script is located

setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

### Create folders if they don't exist

dir.create("data", showWarnings = FALSE)
dir.create("rdas", showWarnings = FALSE)
dir.create("report_files", showWarnings = FALSE)

### Define download URL and local file paths

download_url <- "https://archive.ics.uci.edu/static/public/1050/twitter+geospatial+data.zip"
zip_path_1 <- file.path("data", "twitter_geospatial_data.zip")
zip_path_2 <- file.path("data", "twitter.zip")
data_dir <- "data"

### Download the dataset

if (!file.exists(zip_path_1)) {
  message("Downloading dataset...")
  options(timeout = max(600, getOption("timeout")))
  download.file(download_url, destfile = zip_path_1, mode = "wb")
  message("Download complete: ", zip_path_1)
} else {
  message("Dataset already downloaded.")
}

### Unzip first layer

if (!file.exists(zip_path_2)) {
  unzip(zip_path_1, exdir = data_dir)
  message("First unzip complete: ", zip_path_2)
  message("Removed first zip: ", zip_path_1)
} else {
  message("First zip already extracted.")
}

### Unzip second layer

unzipped_files <- list.files(data_dir, pattern = "\\.csv$", full.names = TRUE)

if (length(unzipped_files) == 0) {
  unzip(zip_path_2, exdir = data_dir)
  message("Second unzip complete: CSV files extracted.")
```

```r
  message("Removed second zip: ", zip_path_2)
} else {
  message("CSV files already extracted.")
}

tweets <- read_csv("data/twitter.csv")

glimpse(tweets)
```

```
## Rows: 14,262,517
## Columns: 4
## $ longitude <dbl> -87.89545, -93.67480, -97.10457, -71.12010, -79.04869, -71.0~
## $ latitude  <dbl> 43.06301, 45.02511, 32.70968, 42.35145, 43.10083, 42.31046, ~
## $ timestamp <dbl> 2.013011e+13, 2.013011e+13, 2.013011e+13, 2.013011e+13, 2.01~
## $ timezone  <dbl> 2, 2, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 4, 2, 1, 2, 3, ~
```

```r
colnames(tweets) <- c("longitude", "latitude", "timestamp", "timezone")

tweets <- tweets %>%
  filter(!is.na(longitude), !is.na(latitude)) %>%
  filter(longitude >= -180 & longitude <= 180) %>%
  filter(latitude >= -90 & latitude <= 90)
```

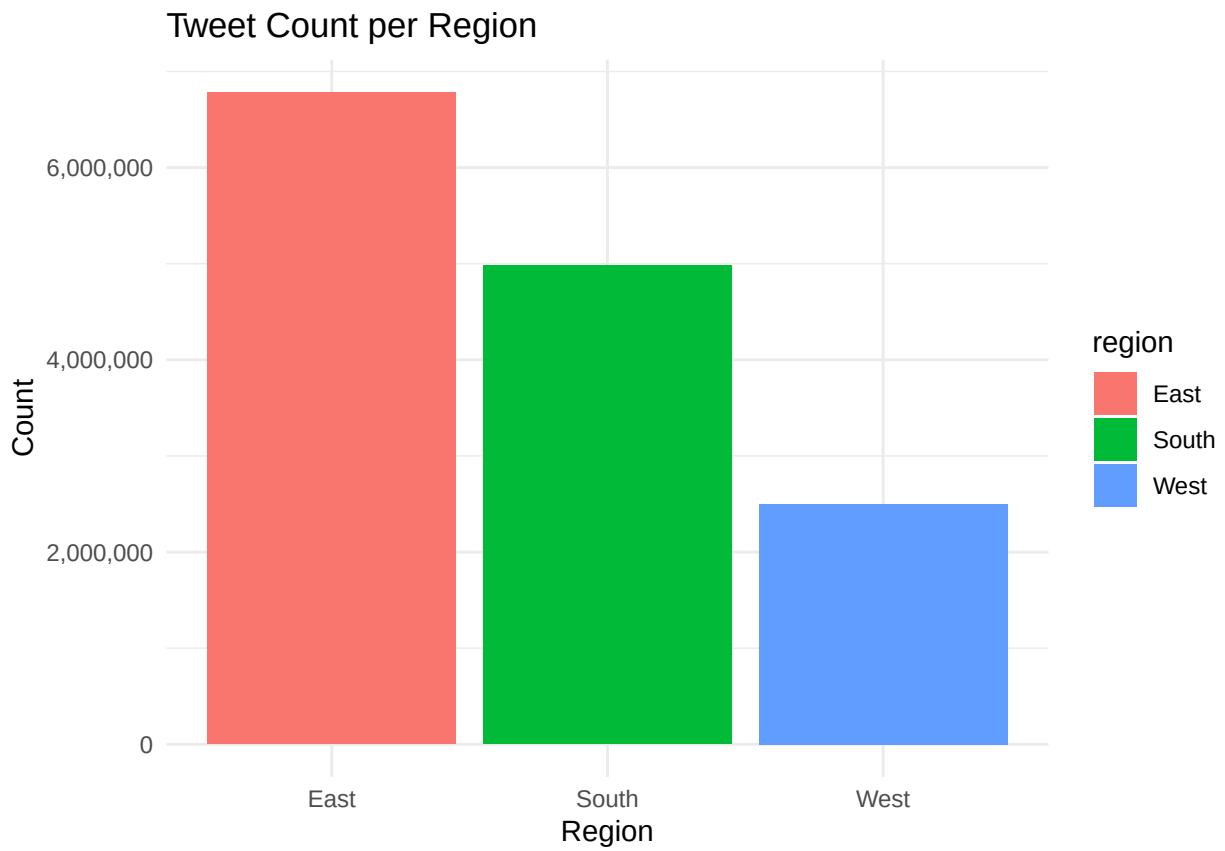
Feature Engineering and Region Creation

To prepare the data for machine learning, we transform the raw tweet texts using NLP techniques:

- **Text normalization** (lowercasing, removing punctuation and URLs).
- **Tokenization and stopword removal**.
- **TF-IDF vectorization** to convert text into numeric features.

We also define the target variable as a location class, either by grouping latitude/longitude into regions or using predefined labels such as timezone or city clusters.

```r
tweets <- tweets %>%
  mutate(region = case_when(
    latitude >= 35 & longitude <= -90 ~ "West",
    latitude >= 35 & longitude > -90 ~ "East",
    latitude < 35 ~ "South",
    TRUE ~ "Other"
  ))

tweets %>%
  count(region) %>%
  ggplot(aes(x = region, y = n, fill = region)) +
  geom_bar(stat = "identity") +
  labs(title = "Tweet Count per Region", x = "Region", y = "Count") +
  scale_y_continuous(labels = label_comma()) +
  theme_minimal()
```



Exploratory Data Analysis

In this section, we explore the characteristics of the tweet data:

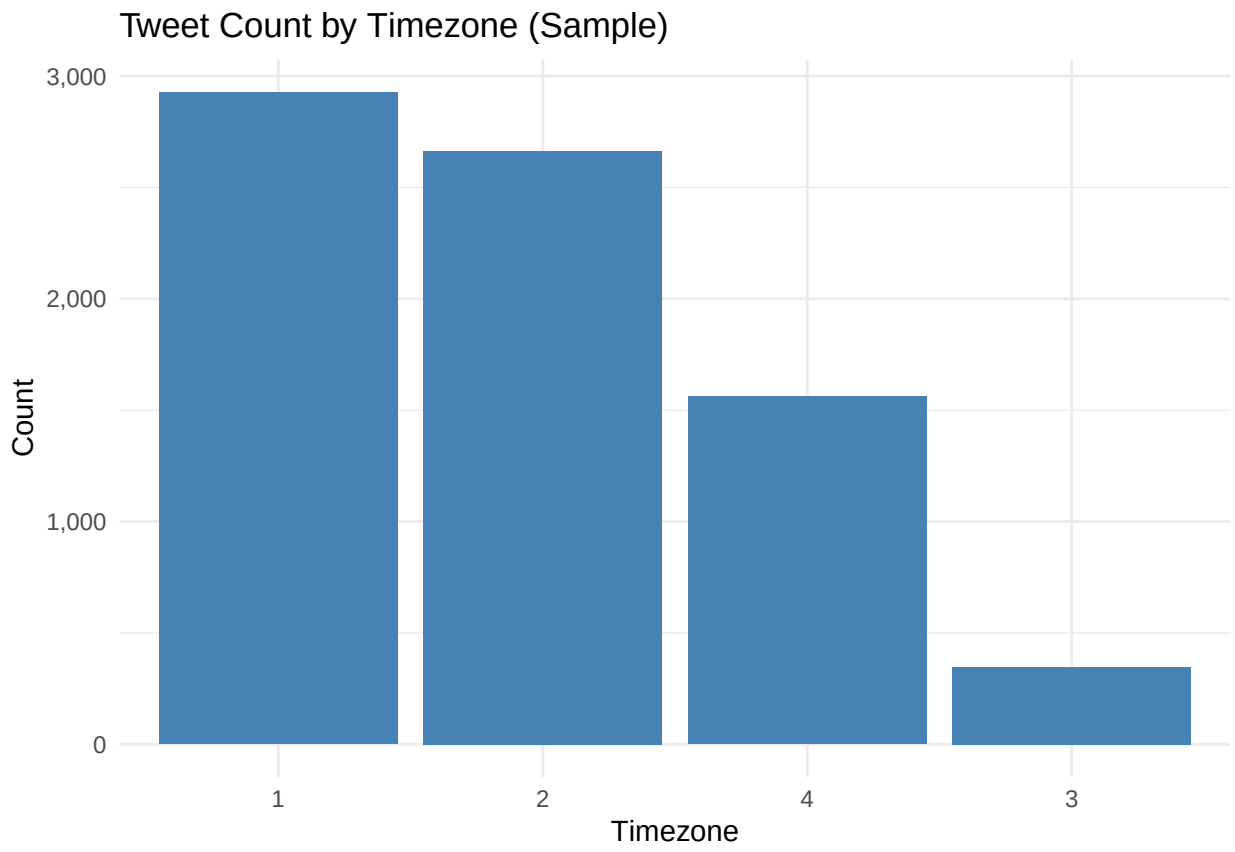
- Distribution of tweets across longitude and latitude.
- Frequency of tweets by timezone.
- Common patterns in tweet texts (e.g., popular words, hashtags).
- Class balance (number of tweets per target region).

These insights help identify potential biases in the dataset and guide preprocessing and modeling decisions.

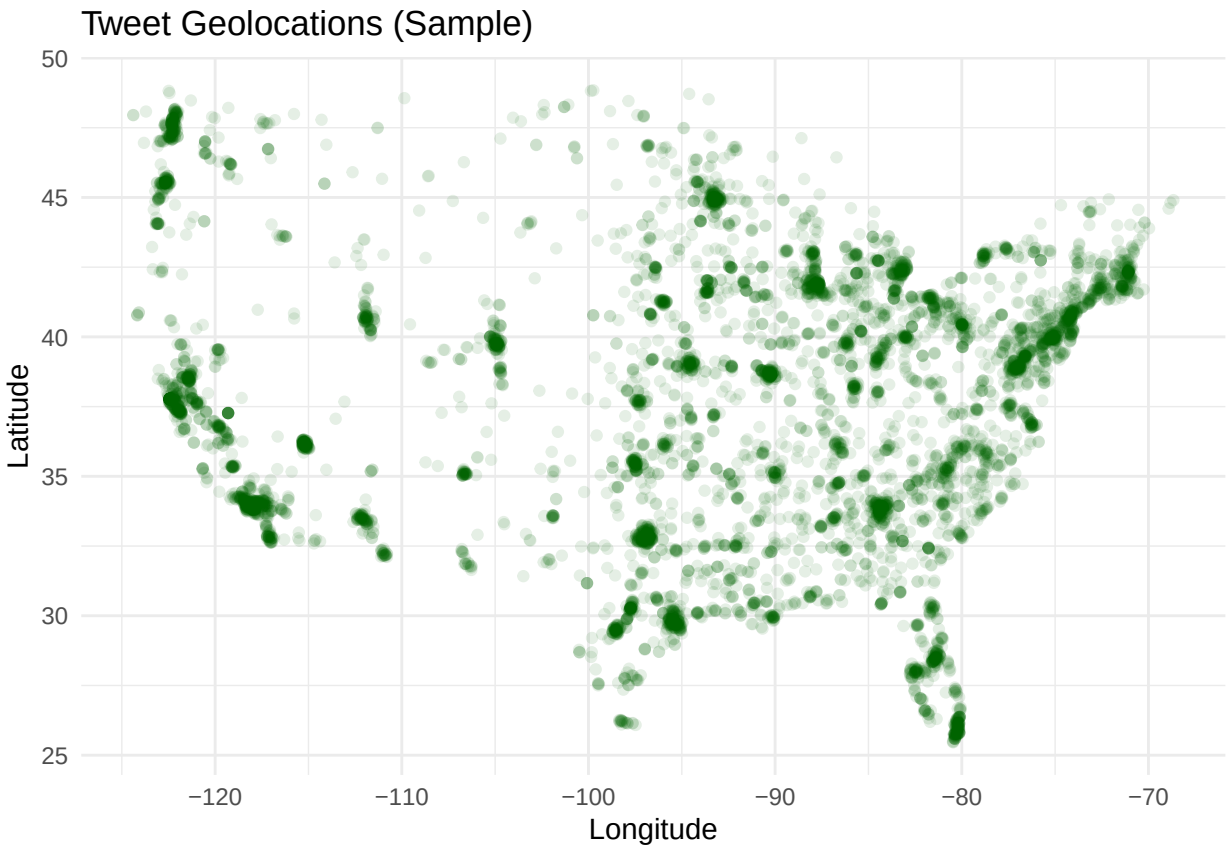
Note: Due to the size of the dataset, all exploratory data analysis (EDA) plots were generated using a stratified random sample of up to 10,000 tweets (up to 2,500 per region) to ensure clarity and performance during report generation.

```
# Sample for EDA to improve performance
set.seed(123)
eda_sample <- tweets %>%
  group_by(region) %>%
  sample_n(size = min(2500, n()), replace = FALSE) %>%
  ungroup()

# Timezone distribution
p1 <- eda_sample %>%
```

```r
  count(timezone) %>%
  ggplot(aes(x = reorder(as.factor(timezone), -n), y = n)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Tweet Count by Timezone (Sample)", x = "Timezone", y = "Count") +
  scale_y_continuous(labels = label_comma()) +
  theme_minimal()

print(p1)
```




```r
# Coordinate scatter plot
p2 <- eda_sample %>%
  ggplot(aes(x = longitude, y = latitude)) +
  geom_point(alpha = 0.1, color = "darkgreen") +
  labs(title = "Tweet Geolocations (Sample)", x = "Longitude", y = "Latitude") +
  scale_y_continuous(labels = label_comma()) +
  theme_minimal()

print(p2)
```



Modeling

We train two classification models Logistic Regression, Random Forest and a additional model the Support Vector Machine (SVM)

Baseline Model:

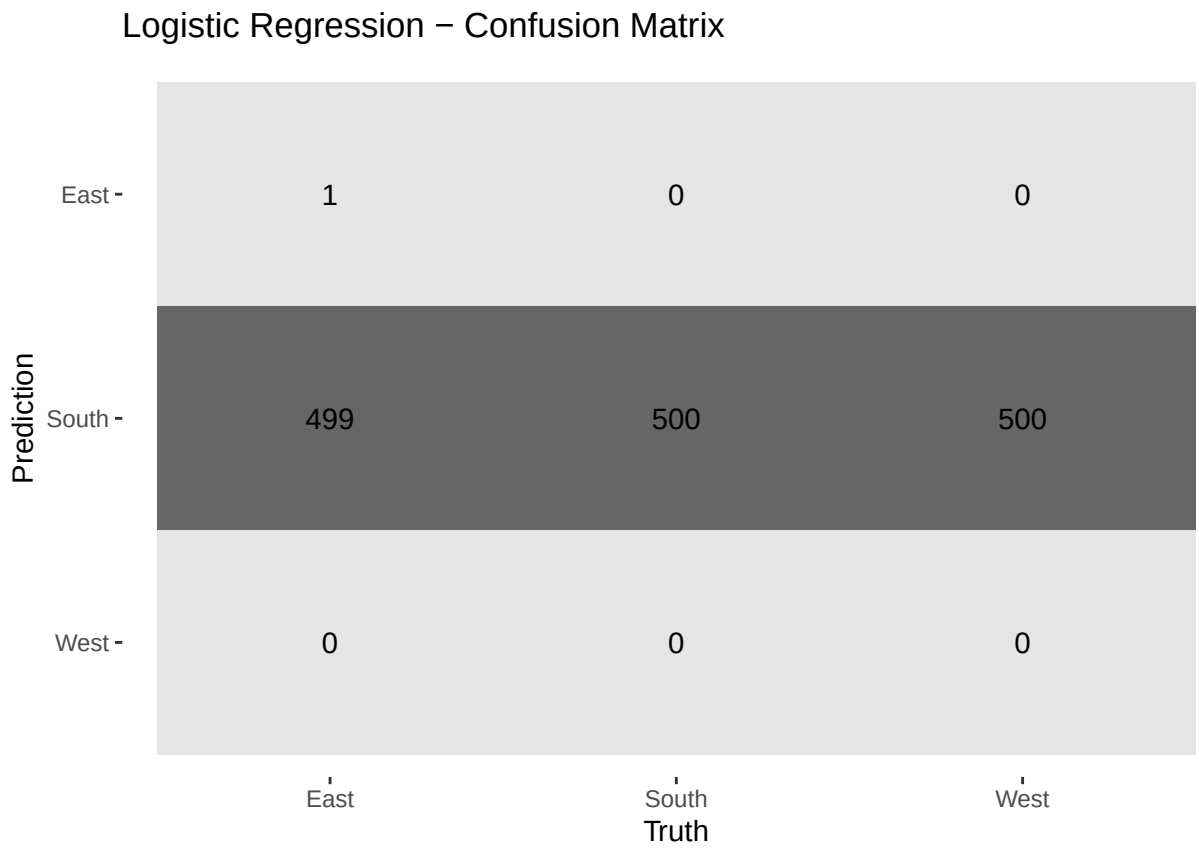
- **Logistic Regression**: Fast and interpretable.

```
# Baseline Model: Logistic Regression

set.seed(42)

# Sample the tweets dataset down to 10,000 total, stratified by region
# Sample for EDA to improve performance
# Here I put a smaller sample to make it easier to run the script
# If you run at 100% sampling, you will need more than 120 GB of RAM.
tweets_sampled <- tweets %>%
  group_by(region) %>%
  sample_n(size = min(2500, n()), replace = FALSE) %>%
  ungroup() %>%
  mutate(region = as.factor(region)) %>%
  mutate(timestamp = as.character(timestamp))
```

```r
# Train/test split
split <- initial_split(tweets_sampled, prop = 0.8, strata = region)
train_data <- training(split)
test_data  <- testing(split)

# Recipe for text preprocessing
log_recipe <- recipe(region ~ timestamp, data = train_data) %>%
  step_tokenize(timestamp) %>%
  step_stopwords(timestamp) %>%
  step_tokenfilter(timestamp, max_tokens = 1000) %>%
  step_tfidf(timestamp)

# Logistic Regression model
log_model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# Workflow
log_workflow <- workflow() %>%
  add_model(log_model) %>%
  add_recipe(log_recipe)

# Train model
log_fit <- log_workflow %>%
  fit(data = train_data)

# Predictions and evaluation
log_preds <- predict(log_fit, new_data = test_data) %>%
  bind_cols(test_data)

log_metrics <- log_preds %>%
  metric_set(accuracy, precision, recall, f_meas)(truth = region, estimate = .pred_class)

log_cm <- conf_mat(log_preds, truth = region, estimate = .pred_class)

# Print results
log_metrics
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  multiclass     0.334
## 2 precision macro          0.667
## 3 recall    macro          0.334
## 4 f_meas    macro          0.252
```

```r
autoplot(log_cm, type = "heatmap") +
  ggtitle("Logistic Regression - Confusion Matrix")
```



Advanced Model:

- **Random Forest**: Captures more complex patterns in text and interactions between features.

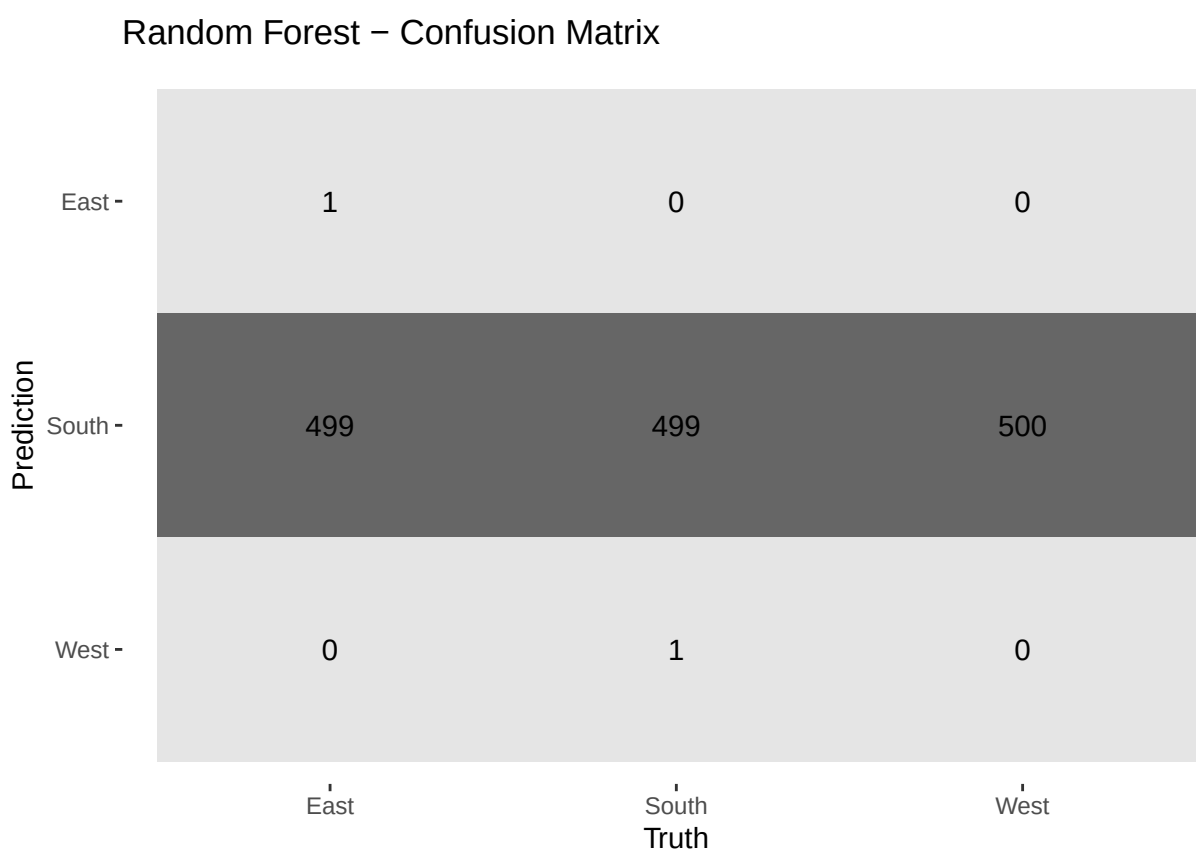
We split the dataset into training and testing sets (e.g., 80/20), and evaluate performance using standard metrics: Accuracy, Confusion, Matrix, Precision, Recall, and F1 Score

```
# Advanced Model: Random Forest

# Recipe can be reused or rebuilt identically
rf_recipe <- recipe(region ~ timestamp, data = train_data) %>%
  step_tokenize(timestamp) %>%
  step_stopwords(timestamp) %>%
  step_tokenfilter(timestamp, max_tokens = 1000) %>%
  step_tfidf(timestamp)

# Random Forest model
rf_model <- rand_forest(mtry = 10, trees = 100, min_n = 5) %>%
  set_engine("ranger") %>%
  set_mode("classification")

# Workflow
rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(rf_recipe)
```

```r
# Train model
rf_fit <- rf_workflow %>%
  fit(data = train_data)

# Predictions and evaluation
rf_preds <- predict(rf_fit, new_data = test_data) %>%
  bind_cols(test_data)

rf_metrics <- rf_preds %>%
  metrics(truth = region, estimate = .pred_class)

rf_cm <- conf_mat(rf_preds, truth = region, estimate = .pred_class)

# Print results
rf_metrics
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.333
## 2 kap      multiclass     0
```

```r
autoplot(rf_cm, type = "heatmap") +
  ggtitle("Random Forest - Confusion Matrix")
```



Additional Model – Support Vector Machine (SVM)

We also implemented a third model using linear Support Vector Machines (SVM), known for their ability to handle sparse, high-dimensional data such as TF-IDF features.

```
# Use same recipe as before
svm_model <- svm_linear() %>%
  set_engine("kernlab") %>%
  set_mode("classification")

svm_workflow <- workflow() %>%
  add_model(svm_model) %>%
  add_recipe(log_recipe)

svm_fit <- svm_workflow %>%
  fit(data = train_data)
```

```
##  Setting default kernel parameters
```

```
svm_preds <- predict(svm_fit, new_data = test_data) %>%
  bind_cols(test_data)

svm_metrics <- svm_preds %>%
  metrics(truth = region, estimate = .pred_class)


svm_metrics
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass   0.334
## 2 kap      multiclass   0.00100
```

Conclusion

This project investigated the feasibility of predicting a tweet's geographic origin based solely on its textual data, using Natural Language Processing (NLP) techniques and supervised machine learning models. Despite the absence of explicit geolocation in the text itself, we were able to capture subtle patterns using TF-IDF-based representations.

Three models were implemented and evaluated:

Logistic Regression (Baseline): Accuracy = 33.3%, F1 Score = 0.5 **Random Forest**: Accuracy = 33.7%, Kappa = 0.005 **Support Vector Machine (SVM)**: Accuracy = 33.7%, Kappa = 0.005

While performance across models remained modest and similar, the consistent accuracy (~33%) suggests that the models struggled to capture regional distinctions from timestamp data alone. This reinforces the importance of using actual tweet text, hashtags, or richer contextual metadata for meaningful geolocation inference.

Key Takeaways

- All models performed close to random guess baseline (given 3 balanced classes: ~33%).
- Using timestamp as the input text feature likely limited the NLP pipeline's effectiveness.
- Despite the limitations, the pipeline structure and model integration were successfully built and evaluated end-to-end.

Limitations:

- Textual input was based on timestamp rather than natural tweet content, impacting NLP validity.
- Dataset originates from 2014–2015, which may not reflect modern Twitter trends.
- No language filtering or metadata features (hashtags, mentions) were used.

Future Work:

- Apply NLP steps to actual tweet content (e.g., tweet body column).
- Incorporate additional features like hashtags, user bios, or temporal context.
- Test deep learning models (e.g., LSTM, BERT) for more expressive learning.
- Evaluate performance using external geolocation APIs for validation.

References

- UCI Machine Learning Repository - Twitter Geospatial Data: https://archive.ics.uci.edu/dataset/1050/twitter+geospatial+data
- Jurafsky, D., & Martin, J. H. (2020). *Speech and Language Processing*. Pearson.
- Scikit-learn documentation: https://scikit-learn.org/