**Step 1: Text Preprocessing**

Preprocess the text to remove unnecessary variations like punctuation, case differences, and abbreviations.

**Example:**

R

CopyEdit

library(stringr)

# Sample dataset

universities <- c(

"UNIVERSITY OF MUMBAI", "SAVITRIBAI PHULE PUNE UNIVERSITY",

"PUNE UNIVERSITY", "MUMBAI UNIVERSITY", "UNIVESITY OF MUMBAI",

"ITM UNIVERITY", "S.R.T.M.U", "RANI CHNNAMMA UNIVERSITY BELAGAVI",

"SOLAPUR UNIVERITY", "SRTMU NANDED"

)

# Preprocess function

preprocess <- function(name) {

name <- tolower(name) # Convert to lowercase

name <- str\_replace\_all(name, "\\.", "") # Remove periods

name <- str\_replace\_all(name, "\\s+", " ")# Replace multiple spaces with single space

name <- str\_trim(name) # Trim leading/trailing spaces

return(name)

}

# Apply preprocessing

universities\_cleaned <- sapply(universities, preprocess)

print(universities\_cleaned)

**Output:**

arduino

CopyEdit

"university of mumbai" "savitribai phule pune university" "pune university" "mumbai university" "univesity of mumbai" "itm univerity" "srtmu" "rani chnnamma university belagavi" "solapur univerity" "srtmu nanded"

**Step 2: Standardize Names Using Rules**

You can create a dictionary or mapping of standardized names and replace inconsistent names with the correct ones.

**Example:**

R

CopyEdit

# Standardized dictionary

standard\_names <- c(

"university of mumbai" = "University of Mumbai",

"mumbai university" = "University of Mumbai",

"univesity of mumbai" = "University of Mumbai",

"savitribai phule pune university" = "Savitribai Phule Pune University",

"pune university" = "Savitribai Phule Pune University",

"itm univerity" = "ITM University",

"srtmu" = "SRTMU",

"srtmu nanded" = "SRTMU",

"rani chnnamma university belagavi" = "Rani Channamma University",

"solapur univerity" = "Solapur University"

)

# Replace using dictionary

universities\_standardized <- standard\_names[universities\_cleaned]

print(universities\_standardized)

**Output:**

arduino

CopyEdit

"University of Mumbai" "Savitribai Phule Pune University" "Savitribai Phule Pune University"

"University of Mumbai" "University of Mumbai" "ITM University" "SRTMU"

"Rani Channamma University" "Solapur University" "SRTMU"

**Step 3: Fuzzy Matching for Similar Names**

If the dataset is large and manually creating a dictionary is impractical, use **fuzzy string matching** with the stringdist or fuzzyjoin package.

**Example with stringdist:**

R

CopyEdit

library(stringdist)

# Standard names list

standard\_names\_list <- c(

"University of Mumbai", "Savitribai Phule Pune University",

"ITM University", "SRTMU", "Rani Channamma University", "Solapur University"

)

# Fuzzy match function

fuzzy\_match <- function(name, choices) {

dist <- stringdist(name, choices, method = "jw") # Jaro-Winkler distance

return(choices[which.min(dist)]) # Closest match

}

# Apply fuzzy matching

universities\_matched <- sapply(universities\_cleaned, fuzzy\_match, choices = standard\_names\_list)

print(universities\_matched)

**Step 4: Machine Learning (Optional)**

For larger datasets, you can use a supervised machine learning approach to classify university names into categories. This requires labeled data with consistent names.

1. **Create Labeled Data**:
   * Map each inconsistent name to a standard name (as in Step 2).
2. **Train a Model**:
   * Use a text classification algorithm like **Naive Bayes** or **Support Vector Machines (SVM)** with caret or text packages.
3. **Predict and Replace**:
   * Use the trained model to classify inconsistent names and replace them with the predicted standard names.

**Example:**

R

CopyEdit

library(caret)

# Example dataset

data <- data.frame(

Name = universities\_cleaned,

Label = c(

"University of Mumbai", "Savitribai Phule Pune University",

"Savitribai Phule Pune University", "University of Mumbai",

"University of Mumbai", "ITM University", "SRTMU",

"Rani Channamma University", "Solapur University", "SRTMU"

)

)

# Split data into training and testing sets

set.seed(123)

train\_index <- createDataPartition(data$Label, p = 0.8, list = FALSE)

train <- data[train\_index, ]

test <- data[-train\_index, ]

# Train a Naive Bayes model

model <- train(Label ~ Name, data = train, method = "nb")

# Predict on the test set

predictions <- predict(model, test)

# View predictions

print(predictions)

**Step 5: Final Cleanup**

After standardization, review the results for accuracy and adjust rules or training data as needed.

This approach balances manual intervention, automated processing, and flexibility for small or large datasets.

For handling a dataset with over 20,000 entries, using **machine learning algorithms** is a robust approach. Here's how you can systematically approach this problem in R:

**1. Define the Problem**

You want to classify inconsistent university names into standardized categories.

* Input: Inconsistent university names.
* Output: A standardized university name for each entry.

This is a **multi-class text classification problem**.

**2. Prepare the Dataset**

To train a machine learning model, you need:

1. A labeled dataset: A subset of your data where each university name is mapped to a standardized name.
2. A cleaned dataset: Preprocess the text to remove noise.

**Preprocess the Data**

R

CopyEdit

library(dplyr)

library(stringr)

# Preprocessing function

preprocess <- function(name) {

name <- tolower(name) # Convert to lowercase

name <- str\_replace\_all(name, "\\.", "") # Remove periods

name <- str\_replace\_all(name, "\\s+", " ")# Replace multiple spaces with single space

name <- str\_trim(name) # Trim leading/trailing spaces

return(name)

}

# Apply preprocessing to the dataset

df <- data.frame(

UniversityName = c(

"UNIVERSITY OF MUMBAI", "SAVITRIBAI PHULE PUNE UNIVERSITY",

"PUNE UNIVERSITY", "MUMBAI UNIVERSITY", "UNIVESITY OF MUMBAI",

"ITM UNIVERITY", "S.R.T.M.U", "RANI CHNNAMMA UNIVERSITY BELAGAVI",

"SOLAPUR UNIVERITY", "SRTMU NANDED"

) # Replace this with your actual dataset

)

df$CleanedName <- preprocess(df$UniversityName)

**3. Train-Test Split**

Split the labeled data into training and testing sets.

R

CopyEdit

library(caret)

# Create a labeled dataset (manually create labels for a subset of data)

labeled\_data <- data.frame(

CleanedName = c(

"university of mumbai", "savitribai phule pune university",

"savitribai phule pune university", "university of mumbai",

"university of mumbai", "itm university", "srtmu",

"rani chnnamma university belagavi", "solapur university", "srtmu"

),

Label = c(

"University of Mumbai", "Savitribai Phule Pune University",

"Savitribai Phule Pune University", "University of Mumbai",

"University of Mumbai", "ITM University", "SRTMU",

"Rani Channamma University", "Solapur University", "SRTMU"

)

)

# Split into training and testing sets

set.seed(123)

train\_index <- createDataPartition(labeled\_data$Label, p = 0.8, list = FALSE)

train\_data <- labeled\_data[train\_index, ]

test\_data <- labeled\_data[-train\_index, ]

**4. Train a Machine Learning Model**

Use text classification algorithms like Naive Bayes or Random Forest.

**Example: Naive Bayes**

R

CopyEdit

library(e1071)

# Train a Naive Bayes model

model <- naiveBayes(Label ~ CleanedName, data = train\_data)

# Predict on the test set

test\_data$Predicted <- predict(model, test\_data)

# Evaluate the model

confusionMatrix(test\_data$Predicted, test\_data$Label)

**5. Use the Model to Predict on the Full Dataset**

Once the model performs well on the test set, apply it to the entire dataset.

R

CopyEdit

# Predict standardized names for the full dataset

df$PredictedLabel <- predict(model, newdata = data.frame(CleanedName = df$CleanedName))

**6. Advanced: Use TF-IDF and a Multinomial Model**

For better results on large datasets, use **TF-IDF (Term Frequency-Inverse Document Frequency)** to create features from text.

**Example with TF-IDF and Random Forest**

R

CopyEdit

library(tm)

library(randomForest)

# Create a corpus and TF-IDF matrix

corpus <- Corpus(VectorSource(train\_data$CleanedName))

dtm <- DocumentTermMatrix(corpus, control = list(weighting = weightTfIdf))

# Convert to a data frame

train\_tfidf <- as.data.frame(as.matrix(dtm))

train\_tfidf$Label <- train\_data$Label

# Train a Random Forest model

rf\_model <- randomForest(Label ~ ., data = train\_tfidf, ntree = 100)

# Preprocess and predict on new data

corpus\_test <- Corpus(VectorSource(test\_data$CleanedName))

dtm\_test <- DocumentTermMatrix(corpus\_test, control = list(dictionary = Terms(dtm)))

test\_tfidf <- as.data.frame(as.matrix(dtm\_test))

test\_tfidf$Label <- test\_data$Label

# Predict

test\_data$Predicted <- predict(rf\_model, newdata = test\_tfidf)

# Evaluate

confusionMatrix(test\_data$Predicted, test\_data$Label)

**7. Scale Up with Parallel Processing**

For large datasets, use parallel processing libraries like doParallel.

**Example:**

R

CopyEdit

library(doParallel)

# Set up parallel backend

cl <- makeCluster(detectCores() - 1) # Use all but one core

registerDoParallel(cl)

# Train Random Forest with parallel processing

rf\_model <- randomForest(Label ~ ., data = train\_tfidf, ntree = 100, do.trace = TRUE)

stopCluster(cl) # Stop parallel backend

**8. Save and Deploy the Model**

Save the trained model for reuse and deploy it on unseen data.

R

CopyEdit

saveRDS(rf\_model, "university\_model.rds")

# Load the model later

model <- readRDS("university\_model.rds")

**Key Considerations**

* Ensure the labeled data covers all possible categories.
* Regularly evaluate the model with new data for accuracy.
* Use fuzzy matching or clustering as a fallback for low-confidence predictions.

This approach will help you classify large datasets efficiently and ensure scalability.