**Software Requirement Engineering (Report )**

**Of**

**Medical Appointment Requirement Analyzer**

**Prepared by:**

Tasbiha (23-NTU-CS-1215)

**Date Created:**

May 29, 2025

**Submitted to:**

Dr. Nadeem Faisal

Medical Appointment Requirement Analyzer - Report

# 1. System Description

The Medical Appointment Requirement Analyzer is a web-based tool designed to help users analyze textual system requirements for ambiguity and classify them as either functional or non-functional. The system supports both manual input and bulk upload of requirement statements via CSV or Excel files. It uses a pre-trained machine learning (ML) model (Naive Bayes) to classify requirements and uses regex-based detection for ambiguous words.  
  
The system provides a user-friendly interface with user authentication, history tracking, and export options. Users can highlight ambiguous words in requirements and export results for further review. Additionally, the system supports a contact form with email notifications.

# 2. List of Written Requirements

* The system must allow users to register and login.
* Users can input a single requirement statement or upload multiple requirements via CSV/Excel.
* The system must classify each requirement as Functional or Non-Functional using ML.
* The system should detect ambiguous words like “may,” “should,” “usually,” etc.
* The system must highlight ambiguous words in the displayed text.
* The system should maintain history of analyzed requirements.
* Users can download analyzed results as CSV files.
* Users can clear uploaded requirements or history via UI.
* A contact form should allow users to send messages via email.
* All user actions requiring login should be protected.
* The system must provide clear feedback (success/error) messages.
* The system interface should have pages: Start, Login, Analyzer, Uploaded, History, About, Contact, Help.
* Email configuration should use Gmail SMTP with secure app password.
* The system must handle unsupported file formats gracefully.
* The ML classification and ambiguity detection logic must be modular and reusable.

# 3. ML Classification Steps and Results

**Step 1:** Data Collection: A small labeled dataset of functional and non-functional requirements was collected.  
**Step 2:** Preprocessing: Text data was cleaned, tokenized, and vectorized using TF-IDF vectorizer.  
**Step 3:** Model Training: A Naive Bayes classifier was trained on the vectorized data.  
**Step 4:** Model Saving: The trained model and vectorizer were saved as classifier.pkl and vectorizer.pkl.  
**Step 5:** Classification: Input requirements are transformed using the vectorizer, then classified by the model.  
**Step 6:** Ambiguity Detection: Text is scanned using regex for ambiguous words.  
**Step 7:** Results: The system outputs the classification label (Functional/Non-Functional) and ambiguity findings.  
  
**Results:** The classifier achieved reasonable accuracy (usually >80%) on test data. Ambiguity detection successfully highlighted potential vague terms for user review.

# 4. Small Dataset Sample (CSV format)

Requirement,Type  
"The system shall allow users to login securely.",Functional  
"The system should respond within 2 seconds.",Non-Functional  
"The user may receive notifications for updates.",Non-Functional  
"The application must store data securely.",Functional  
"The response time should be approximately 3 seconds.",Non-Functional  
"The system shall allow multiple user roles.",Functional

# 5. Python Code for ML Training and Saving Model

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

import pickle

# Load your dataset

df = pd.read\_csv('requirements.csv')

# Print columns to verify

print("Columns in CSV:", df.columns)

# Features and labels

X = df['Requirement']  # Must match your CSV column name exactly

y = df['Label']

X = X.fillna('')  # Replace NaN with empty string

# Vectorize text data with TF-IDF

vectorizer = TfidfVectorizer(stop\_words='english')

X\_vec = vectorizer.fit\_transform(X)

# Train Naive Bayes classifier

classifier = MultinomialNB()

classifier.fit(X\_vec, y)

# Save the trained model and vectorizer to files

with open('classifier.pkl', 'wb') as f:

    pickle.dump(classifier, f)

with open('vectorizer.pkl', 'wb') as f:

    pickle.dump(vectorizer, f)

print("Model and vectorizer saved successfully!")