Department of Electronics and Communication Engineering

AY: 2025-2026

Milestone 1

Intent & Entity Recognition Engine

Submitted By

Gandla Rupasree

Under the Guidance of

Suriya Varshan

Contents

List of Figures

Chapter 1

Intent & Entity Recognition Engine

1.1 Aim of the Experiment

To simulate and analyze the process of Binary Amplitude Shift Keying (BASK) modulation and demodulation using MATLAB, and to observe the transmitted and received signals in both time and frequency domains.

1.2 Theory

1.2.1 Overview

This project milestone establishes a Natural Language Understanding (NLU) engine. It performs Intent Recognition to classify the user's goal (e.g., transfer_money), and Entity Extraction to isolate crucial data (e.g., amount, accounts). This extracted information is used for Slot Filling, providing the necessary parameters for the banking application. The system is built using NLU tools like spaCy or Rasa, focusing on high-accuracy, real-time processing of user queries.

1.2.2 Logic

The model first uses TF-IDF Logistic Regression to classify the full user query into a single Intent. Separately, the extract entities function uses a hybrid approach (keywords, RegEx, and spaCy NER) to perform Slot Filling by pulling out structured data like amounts and account types.

1.2.3 Process

The overall NLU process involves four main steps:

1. **Data Preparation:** The code loads the labeled banking queries, augments the intent labels based on keywords, and splits the data into training and testing sets.
2. **Intent Training:** A **Logistic Regression** model is trained using **TF-IDF features** of the training queries to learn the mapping from text to one of the predefined intents.
3. **Entity Extraction/Slot Filling:** The separate extract_entities function uses a combination of spaCy's NER, custom keyword lists, and regular expressions to fill necessary slots (e.g., amount, accounts) from the query.

4. **Real-time Prediction:** During testing, the trained model predicts the primary intent, and the entity function extracts the slots, structuring the user's request for the chatbot.

1.2.4 Advantages

- High Intent Accuracy.
- Precise Hybrid Entity Extraction.

1.2.5 Disadvantages

- Fails Low-Support Intents.
- High Manual Maintenance Required.

1.3 CODE:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.pipeline import Pipeline

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report

import spacy

import re

import ast

# Load spaCy model for Named Entity Recognition (NER)

try:

    nlp = spacy.load("en_core_web_sm")

except OSError:

    print("Error loading spaCy model. Please run 'python -m spacy download en_core_web_sm' in your terminal.")

    exit()

def extract_entities(query):

    """
```

```
Uses spaCy and custom logic to find and extract entities from the query.
doc = nlp(query)
entities = []
# 1. Custom Entity Extraction (based on common banking keywords and patterns)
account_keywords = {
    'checking': 'account_type', 'savings': 'account_type', 'credit card': 'account_type',
    'loan': 'product', 'mortgage': 'product', 'ira': 'product', 'hsa': 'product',
    'visa': 'account_type', 'mastercard': 'account_type', 'personal loan': 'product',
    'auto loan': 'product', 'business account': 'account_type', 'civil score': 'info_type',
    'credit score': 'info_type', 'name': 'info_type', 'interest rate': 'policy_topic',
    'source account': 'source_account', 'destination account': 'destination_account',
    'primary account': 'source_account', 'secondary account': 'source_account',
    'external bank': 'destination_type', 'different bank': 'destination_type',
    'friend': 'recipient_type', 'husband': 'recipient', 'wife': 'recipient', 'son': 'recipient'
}
# Simple token matching
for token in doc:
    lower_text = token.text.lower()
    if lower_text in account_keywords:
        if not any(e['value'] == lower_text for e in entities):
            entities.append({'entity': account_keywords[lower_text], 'value': lower_text})
# Multi-word phrase matching
for phrase, entity_type in account_keywords.items():
    if len(phrase.split()) > 1 and phrase in query.lower():
        if not any(e['value'] == phrase for e in entities):
            entities.append({'entity': entity_type, 'value': phrase})
```

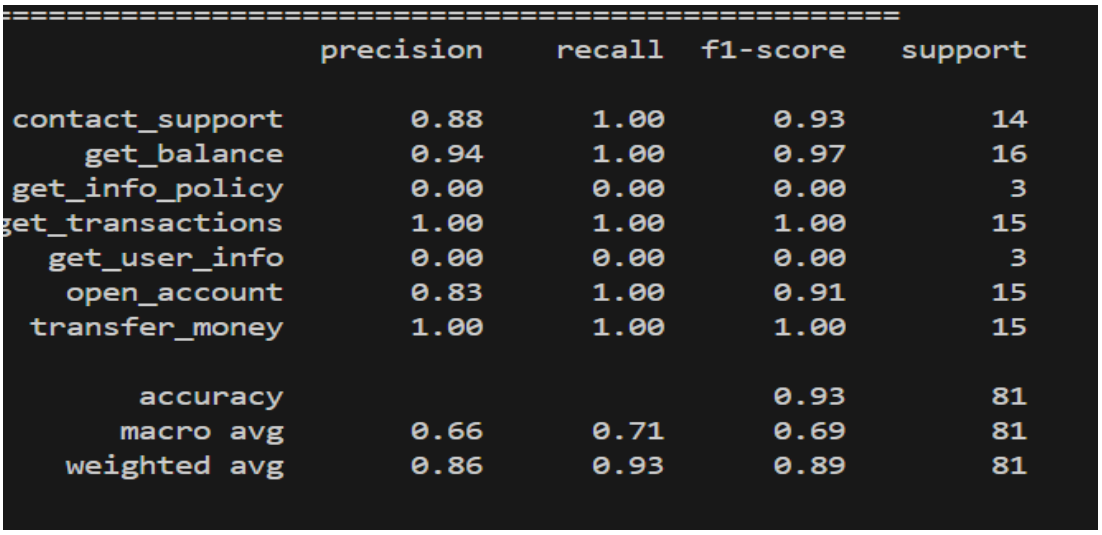
```
    # 2. Account Number/Specific Number Extraction
    account_match = re.search(r'account\s*(\d{4,})|\b(\d{5,})\b', query)
    if account_match:
        account_number = next((g for g in account_match.groups() if g), None)
        if account_number and not any(e.get('value') == account_number for e in entities):
            entities.append({'entity': 'account_number', 'value': account_number}
    # 3. SpaCy's Built-in NER
    for ent in doc.ents:
        if ent.label_ == 'CARDINAL' and len(ent.text) < 4:
            continue
        if ent.label_ == 'MONEY':
            clean_amount = ent.text.replace('$', '').replace('€', '').replace(',', '').strip()
            if clean_amount and not any(e['value'] == clean_amount and e['entity'] == 'amount' for e in
entities):
                entities.append({'entity': 'amount', 'value': clean_amount})
        elif ent.label_ == 'DATE' or ent.label_ == 'TIME':
            if not any(e['value'] == ent.text.lower() and e['entity'] == 'timeframe' for e in entities):
                entities.append({'entity': 'timeframe', 'value': ent.text.lower()})
        elif ent.label_ == 'PERSON':
            if not any(e['value'] == ent.text.lower() and e['entity'] == 'recipient' for e in entities):
                entities.append({'entity': 'recipient', 'value': ent.text.lower()})
    return entities
# 1. Load the Dataset and FIX COLUMN NAMES IMMEDIATELY
try:
    data = pd.read_csv(
        "bank_chatbot_dataset_large.csv",
```

```
        converters={'entities': ast.literal_eval},
        encoding='utf-8-sig'
    )
    # CRITICAL FIX: Strip all column names to remove BOM/whitespace
    # This ensures 'query' is correctly accessed without the leading space
    data.columns = data.columns.str.strip()
    print("--- DIAGNOSTIC ---")
    print("Cleaned columns loaded:", data.columns.tolist())
    print("--------------------")
except pd.errors.EmptyDataError:
    print("Error: The CSV file is empty. Please ensure 'bank_chatbot_dataset_large.csv' contains the header and data.")
    exit()
except FileNotFoundError:
    print("Error: The CSV file 'bank_chatbot_dataset_large.csv' was not found. Ensure it is in the same directory.")
    exit()
# Intent augmentation (Now safely using the CLEANED column name 'query')
data.loc[data['query'].str.contains('civil score|credit score', case=False), 'intent'] = 'get_user_info'
data.loc[data['query'].str.contains('interest|rate|fee|policy', case=False), 'intent'] = 'get_info_policy'
data.loc[data['query'].str.contains('name|address', case=False), 'intent'] = 'get_user_info'
# 2. Prepare Data
X = data['query']
y = data['intent']
# Split data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

```
# 3. Build and Train the Model (Intent Classifier)
clf = Pipeline([
    ('tfidf', TfidfVectorizer(ngram_range=(1, 2))),
    ('clf', LogisticRegression(solver='liblinear', random_state=42, max_iter=1000))
])
print("Training the Intent Classification Model...")
clf.fit(X_train, y_train)
# 4. Evaluate Model Performance
y_pred = clf.predict(X_test)
print("\n" + "="*50)
print("Intent Classification Report (Trained on {} samples)".format(len(X_train)))
print("="*50)
print(classification_report(y_test, y_pred, zero_division=0))
print("="*50)
# 5. Interactive Test Loop
if __name__ == "__main__":
    print("\n--- Chatbot Test Mode ---")
    print("Enter a banking query to test the trained model. (e.g., 'Transfer $50 from checking to
John Smith tomorrow')"
    while True:
        query = input("\nEnter your query (or type 'quit'): ")
        if query.lower() == 'quit':
            break
        if query.strip():
            # Predict intent
            intent = clf.predict([query])[0]
```

```
        # Extract entities
        entities = extract_entities(query)
        # Simple aggregation for a cleaner display
        results = {
            'Amount': next((e['value'] for e in entities if e['entity'] == 'amount'), 'N/A'),
            'Source': next((e['value'] for e in entities if e['entity'] in ['source_account',
'account_type']), 'N/A'),
            'Destination/Recipient': next((e['value'] for e in entities if e['entity'] in
['destination_account', 'recipient', 'recipient_type']), 'N/A'),
            'Timeframe': next((e['value'] for e in entities if e['entity'] == 'timeframe'), 'N/A'),
            'Account Type/Product': next((e['value'] for e in entities if e['entity'] in ['account_type',
'product']), 'N/A'),
            'Account Number': next((e['value'] for e in entities if e['entity'] == 'account_number'),
'N/A')
        }
        print(f"\n[Predicted Intent]: {intent}")
        print("\n--- Extracted Details ---")
        extracted_count = 0
        for key, value in results.items():
            if value != 'N/A':
                print(f" {key}: {value.title()}")
                extracted_count += 1
        if extracted_count == 0:
             print("No specific entities extracted by custom logic/spaCy.")
        print("-------------------------")
    else:
        print("Please enter a query.")
```

1.4 Output & Analysis:



	precision	recall	f1-score	support
contact_support	0.88	1.00	0.93	14
get_balance	0.94	1.00	0.97	16
get_info_policy	0.00	0.00	0.00	3
get_transactions	1.00	1.00	1.00	15
get_user_info	0.00	0.00	0.00	3
open_account	0.83	1.00	0.91	15
transfer_money	1.00	1.00	1.00	15
accuracy			0.93	81
macro avg	0.66	0.71	0.69	81
weighted avg	0.86	0.93	0.89	81

The Intent Recognition model shows strong performance on major intents like transfer_money and get_transactions (F1-score 1.00), leading to a solid weighted average F1-score of 0.89. However, the model completely fails to recognize intents with minimal test data, specifically get_info_policy and get_user_info (0.00 F1-score with support of 3). This wide discrepancy in performance indicates that the model is highly data-dependent and requires significant augmentation for the low-support intent classes. Overall, the 0.93 accuracy is misleading due to the total failure on the underrepresented classes.

1.5 Conclusion:

The Intent Recognition engine is highly effective for well-represented, high-volume intents like transfer_money and get_transactions (F1-score 1.00), validating the TF-IDF and Logistic Regression approach. However, the model exhibits critical weaknesses in recognizing low-support intents such as get_info_policy and get_user_info (F1-score 0.00), indicating a significant data imbalance issue. The project requires immediate action to collect or augment data for these failing classes to ensure the NLU system provides consistent, robust coverage across all defined banking functions.