IT TICKET CLASSIFICATION USING LLM

by

(Team C)

(Owais Shaikh;

Ashutosh Joshi;

Amulya Potluri)

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2023

ABSTRACT

This paper utilizes machine learning analysis and techniques to create model for classification of IT Tickets. The main objective is to create a robust model to classify the It tickets and assign them to the correct team utilizing the dataset having JPMorgan Chase & CO service tickets from Kaggle. Several Models were built using linear regression, Random Forest and LLM to analyze and produce the best model with better accuracy. This study aims to deliver a best model that can classify the IT tickets by analyzing the ticket descriptions. Experimental results and the analysis made demonstrate the effectiveness of the proposed models, contributing to the development of machine learning models to classify the tickets. It also helps the IT sector to come out of traditional approaches by fastening their work with this type of new advancements to deliver services on time.

LIST OF ABBREVIATIONS AND SYMBOLS

AUC Accuracy

LLM Large Language Model

LR Logistic Regression

RF Random Forest

< Less than

= Equal to

ACKNOWLEDGMENTS

The fulfillment of accomplishing the project would be incomplete without of the support of the people. Right from the Initial days of the project, we have support immense guidance and encouragement from the people around us.

Our sincerely thankfulness to **Professor Unal Sakoglu,** Project Coordinator, Data Science Department, University of Maryland Baltimore County for his support and guidance. His assistance helped us to overcome the challenges and smooth completion of the project.

We also express our gratitude towards the Project team members for their suggestions and feedback which has contributed to successful model development.

TEAM MEMBERS' CONTRIBUTIONS

Amulya Potluri worked on Data Cleaning, applying the EDA techniques and model building using Logistic regression and Random Forest. Owais Shaikh worked on EDA Techniques, model building using LLM and Model Evaluation. Ashutosh Joshi worked on EDA Techniques, model building using LLM and Model Evaluation. We all three collectively worked on the reports and presentations. We collaborated with each other on weekly basis to draw the insights from the work done, listed out the future steps and helped each other to overcome the challenges. Everyone is successful in delivering their work on time.

Name	Duties	Achievements
Amulya Potluri	Data Cleaning, EDA Techniques, Traditional Modeling, Presentations, Reports, Github Page.	Successfully Finished assigned tasks
Owais Shaikh	EDA Techniques, Traditional Modeling, LLM, Reports,Implementation, Presentations, Github Page	Successfully Finished assigned tasks
Ashutosh Joshi	EDA Techniques, Traditional Modeling, LLM Implementation, Presentations, Reports, Github Page	Successfully Finished assigned tasks

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

In IT sector, most of the issues or bugs are usually solved through a ticketing system. There are several ways to raise a ticket. It can be through call using helpline number, informing the helpdesk, or filling a ticket form. Tickets are usually raised if there is anything wrong with the functionality, existing system, missing data, name change, components missing on a page, etc. These issues may be simple or complex but both types of issues can be fixed when a ticket is raised for them. Sometimes, one issue may end up creating one more issue. Issues are sometimes more complex to deal with if they are delayed. So, most of the IT support teams are usually loaded with lots of tickets. Each ticket has a specified team to resolve and time frame to be fixed (Federico Pascual, 2019).

I.1 CURRENT APPROACHES FOR THE PROBLEM

Most of the organizations follow the below two approaches to assign the tickets that are raised.

Manual Assignment

This is the simplest way of assigning tickets to the agent or teams. The organizations hire some people and train them to assign the tickets. These trained people will have all the manuals available with them, they can either memorize or go through the manuals and assign the tickets. Once the ticket is opened, they scroll down, check the description, and then assign it to the respective agent or the team that usually solve the ticket (Helpdesk).

Automated Assignment

This is a simple assignment strategy which assigns the tickets based on the conditions. If the condition is met, then it assigns to the department. If the condition doesn't meet, it moves to the next condition. It moves till the last condition and still if doesn't satisfy then it assigns to the default team as per the organization polices and rules (Helpdesk).

I.2 THE DESCRIPTION OF THE DATA AND PROBLEM STATEMENT

Problem Statement

The problem statement here is to classify the ticket to the right team based on the description available in the target variable i.e.., 'Product'. This involves prediction of the department that can resolve the particular type of ticket among the 17 available departments.

Dataset

The data set is collected from Kaggle(Venkatasubramanian Sundara Mahadevan, 2021). https://www.kaggle.com/datasets/venkatasubramanian/automatic-ticket-classification

Classes

We have 17 different classes available in the target variable. They are listed below.

- Credit card or prepaid card
- Checking or savings account
- Mortgage
- Credit reporting, credit repair services, or other personal consumer reports
- Credit card
- Bank account or service
- Debt collection
- Money transfer, virtual currency, or money service
- Vehicle loan or lease
- Consumer Loan
- Student loan
- Money transfers
- Payday loan, title loan, or personal loan
- Credit reporting
- Other financial service
- Prepaid card
- Payday loan

Test Split: 20%
Train Split: 80%

I.3 LITERATURE SURVEY

Paramesh, Ramya, Shreedhara conducted research and developed a model for IT Ticket Classification. The model started with a collection of historical data on tickets which later followed a series of actions including pre-processing phase, vector representation, vector dimensionality reduction, model building and generating a response as output which says the team the ticket should be assigned. The dataset had 10,742 records and 18 different classes. This article also used bagging models to build the classifier model and considered AUC, precision, recall, f-score as evaluation metrics. The proposed model in the article achieved an AUC performance of LR, KNN, MNB and SVM are respectively 80.50%, 69.24%, 70.1% and 88.22% respectively (Paramesh, Ramya, Shreedhara, 2018).

Howard. J article proposed ULMFiT which is an effective transfer learning method that can be literally applied on any type of task in NLP. This primarily includes three stages discriminative fine-tuning, slanted triangular learning rates, and gradual unfreezing. These are used to retain the previous knowledge and train the model to achieve better predictions. Author utilized six widely studied datasets to deliver the significance of ULMFit. He used three common text classification tasks: sentiment analysis, question classification, and topic classification (Jeremy Howard, Sebastian Ruder, 2018).

II. METHODS

II.1 Description of Data Processing and Feature Engineering Employed

This phase was critical to ensure data quality and usability. The null values checking, and other sanity checks were done. The data did not have any null values but had a lot of complaints were blank and caused in accuracy and lack of precision in the initial modeling runs. There were 17 categories initially but were changed to final 5 categories after merging. Our dataset after merging had 19337 complaints in total for modelling.

Classes before Merging	Classes merged into
• Checking or Savings Account • Other Financial Service • Money Transfers • Money Transfer, Virtual Currency or Money Services. • Bank Account or Services	Bank Account or Services
• Payday Loan • Payday loan, title loan, or personal loan. • Student Loan • Vehicle Loan or Lease • Consumer Loan • Mortgage	Mortgage
• Credit Reporting • Credit Reporting, Credit Repair Services, or other consumer reports.	Credit Reporting, Credit Repair Services, or other consumer reports.
• Credit Card • Credit Card or Prepaid Card • Prepaid Card	Credit Card or Prepaid Card
• Debt Collection	Debt Collection

Table : 2.1 Merging classes in the target variable

EDA

Basic statistics, such as mean, standard deviation, and quantile range were computed and analyzed. The distribution of the complaints across different categories of products was analyzed and visualized. This analysis provided profound insights into the distribution and nature of the products and highlighted popular categories.

Count plot and pie charts were generated to understand data distribution across all 17 categories.

Review Text Processing

First, the complaint text was tokenized. Second, all the characters in the text were converted to lowercase to ensure uniformity, as text data is case-sensitive. Third, the punctuation and stopwords were removed. This was essential for focusing on significant words. Lastly, text contractions were expanded, and special characters were drawn to clean the data further. The list of words in the 'complaints' column was converted to a continuous string.

TF-IDF Vectorization

Each word in the complaint acts like a feature, but each complaint also has unnecessary features that might not be relevant to our model.TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that reflects that a word is essential to a document in a collection or corpus.TF-IDF gives higher weights to terms frequently used in a specific complaint but is not standard across all complaints. This helps identify words that are distinctive to a particular complaint and might carry sentiment or topic-specific information.

TF-IDF will return a sparse matrix with all the words in the complaint converted to vectors with a specific weight. Since we use ratings and the complaint length as a feature, we convert these to a sparse matrix and combine them to form one feature for modeling. Our dataset is now ready for modeling; we have our features, i.e., complaints and the target variable (product). However, TF-IDF can result in high-dimensional data, especially when dealing with large vocabularies, and this is where TSVD comes in. Truncated SVD can be particularly useful for clustering and semi-supervised learning tasks. Clustering algorithms, such as K-means or hierarchical clustering, can struggle with high-dimensional data due to the curse of dimensionality, and reducing the dimensionality of the data with TSVD can lead to better performance.

II.2 Description of Model Implementation

Traditional Model Implementation

All our models follow the typical classification setup. The features are processed and as mentioned in the TF-IDF section, we have one combined input feature of Complaints as a sparse matrix and our target variable is the Products. Random forest is a powerful ensemble technique that iteratively builds an additive model by minimizing the expected value of a given loss function, refining its predictions with a series of weak learners, typically decision trees, to achieve high accuracy and efficiency in various machine learning tasks.

Unlike decision tree-based approaches such as RF, LR is less prone to overfitting, making it advantageous for high-dimensional data or situations with limited sample sizes. Overall, LR provides a robust and interpretable framework for multi-class classification tasks in various machine learning applications.

The inherent flexibility of the LR and provides a robust foundation for classification. This is also true for RF classification. (Andreas C. Müller, Sarah Guido, 2016). We did a 80-20 split for our training and our test datasets.

Hyper-Parameter Tuning

For hyperparameter tuning on our LR model, we performed a five-fold Cross Validation. We identified the best parameters for the learning rate, number of estimators, and max depth. Using the best parameters from Cross Validation, the models. we retrained and evaluated. For model evaluation, we evaluated the models and compared the results on the following metrics, AUC, and the detailed confusion matrix for multi-class classification.

BERT Large Language Model Implementation.

Further we fine-tuned a LLM to evaluate the AUC and improve classification. In this process, the BERT based Uncased model for classification was fine-tuned on our dataset to produce a few shots learning classification. Due to the compute resources limitation, we had to choose the retrain the model over our modified dataset with 5 categories of classes instead of 17 which still required a lot of computing resources. To implement this, we used Google Colab Pro with T4 GPU. It used 96.4 compute units. The total training time was 3 hours. The model has 7 billion parameters. For this approach, we specifically chose the sequence classification model of Tensorflow python library which has been pre-trained on book corpus, a dataset consisting of 11, 038 unpublished books and English Wikipedia. We have trained the model on 30 epochs and the observed loss is negligible. The model also performed well on the test dataset(Hugging face).

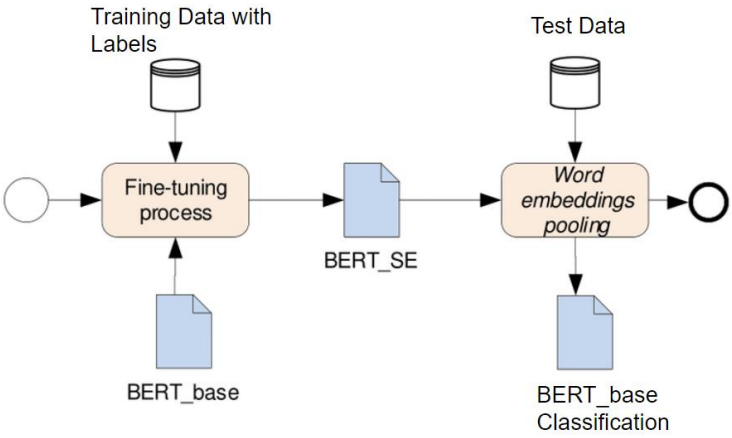


Fig: 2.1 Data Modelling with BERT LLM.

RESULTS

Data Pre-Processing

	product	complaint_what_happened
1	Debt collection	Good morning my name is XXXX XXXX and I apprec...
2	Credit card or prepaid card	I upgraded my XXXX XXXX card in XX/XX/2018 and...
10	Credit reporting, credit repair services, or o...	Chase Card was reported on XX/XX/2019. However...
11	Credit reporting, credit repair services, or o...	On XX/XX/2018, while trying to book a XXXX XX...

	Complaints	Target
1	good morning name appreciate could help put st...	Debt collection
2	upgraded card told agent upgrade anniversary d...	Credit card or prepaid card
10	chase card reported however fraudulent applica...	Credit reporting, credit repair services, or o...
11	trying book ticket came across offer applied t...	Credit reporting, credit repair services, or o...
14	grand son give check deposit chase account fun...	Bank account or service

Fig: 3.1 Data before(left) and after(right) Pre-Processing.

The above Fig: 3.1 shows the changes before and after the pre-processing of the text and before the TF-IDF vectorization. We can see that the text looks more refined after removing punctuation and the stop words. The words were then evaluated and given weights by TF-IDF vectorizer and dimensionality reduction, the result was a final feature in the form of a sparse matrix of shape (19337, 250).

EDA

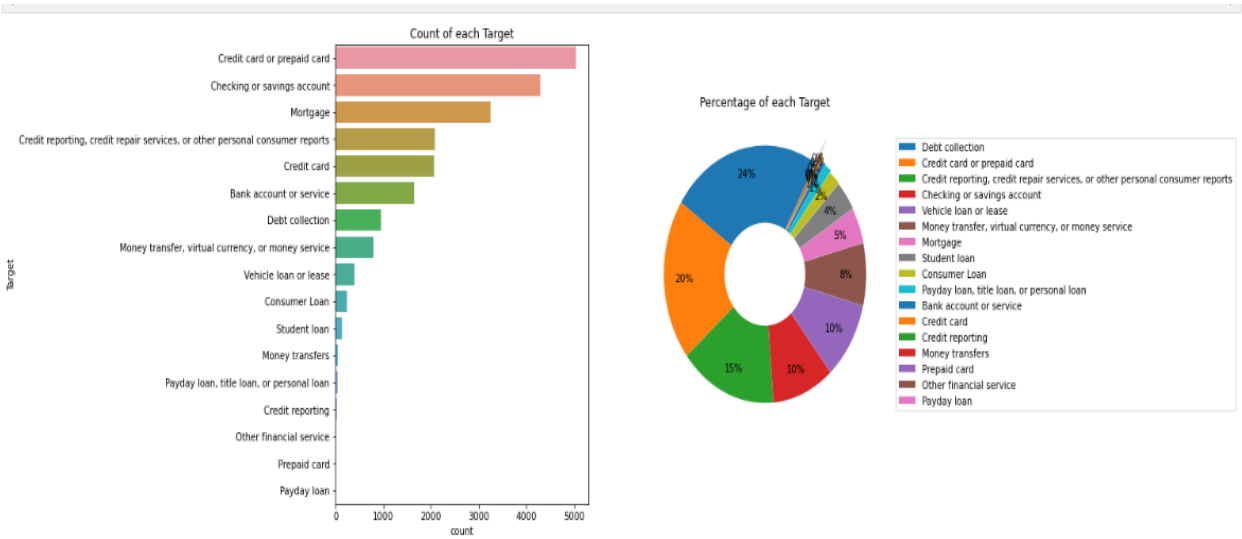


Fig: 3.2 Data distribution before merging the data.

From the Fig: 3.2, we can see the count-plot all the products, which is our target variable in the dataset. We can see that there is a data imbalance in the categories; the highest category has more than 8000 samples while the lowest only has one, also evident in the Pie chart. In order to resolve the data imbalance, we merged similar categories.

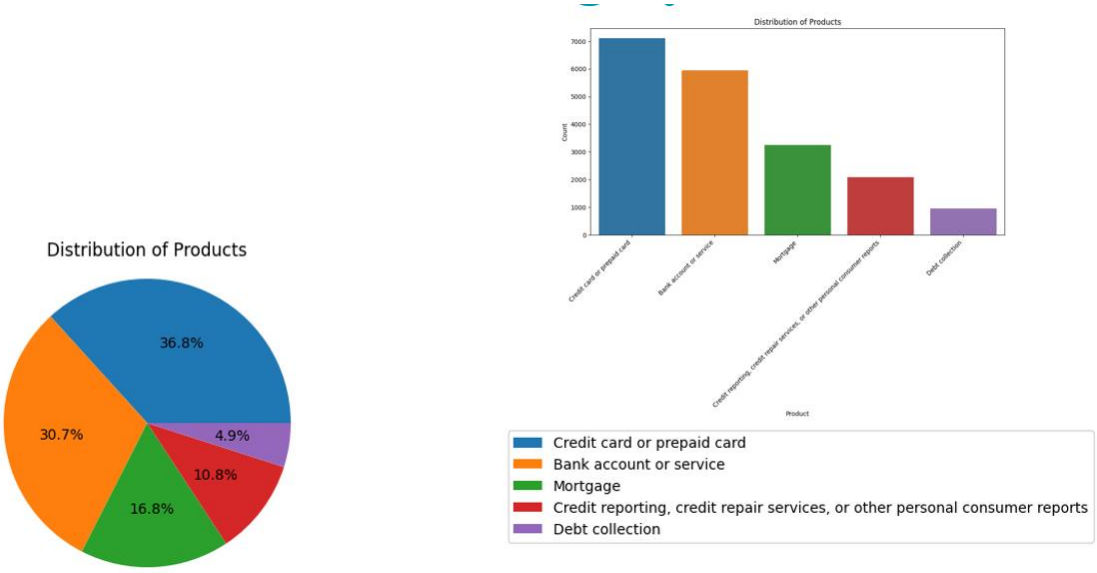
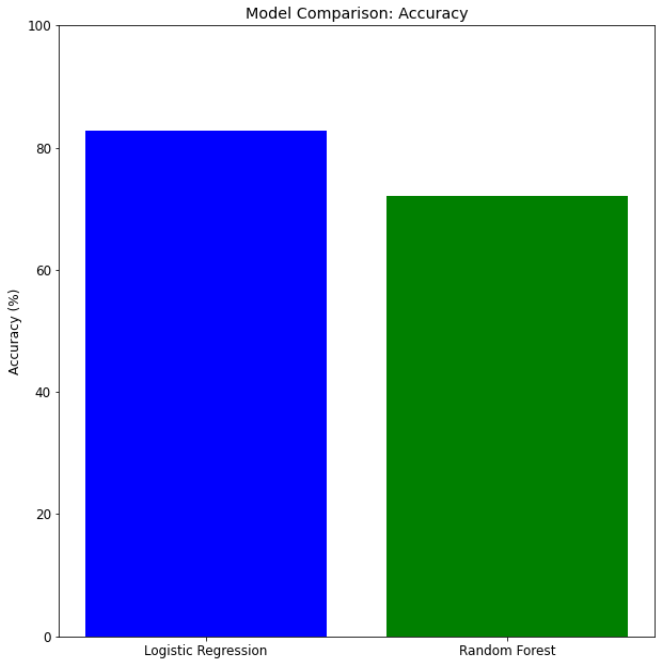


Fig: 3.3 Data distribution percentage after Category reduction.

The resultant was five major categories seen in the count plot and the pie chart in Fig: 3.3. The lowest category count is now approximately 1000, which is enough samples for our model to interpret and classify.

Traditional Model Implementation



We can see in Fig: 3.4, LR was a better performing model with an AUC of 83% after hyper-parameter tuning and cross-validation. While RF classification performed relatively well, with an AUC of 74%. This could be because LR classifies using weights while RF classifies based on voting from several decision trees thus adding bias to its output based on the samples in the decision tree and is generally seen in imbalanced datasets.**(**May Phu Paing, 2018)

Fig: 3.4 Data Modelling with Logistic Regression and Random Forest.

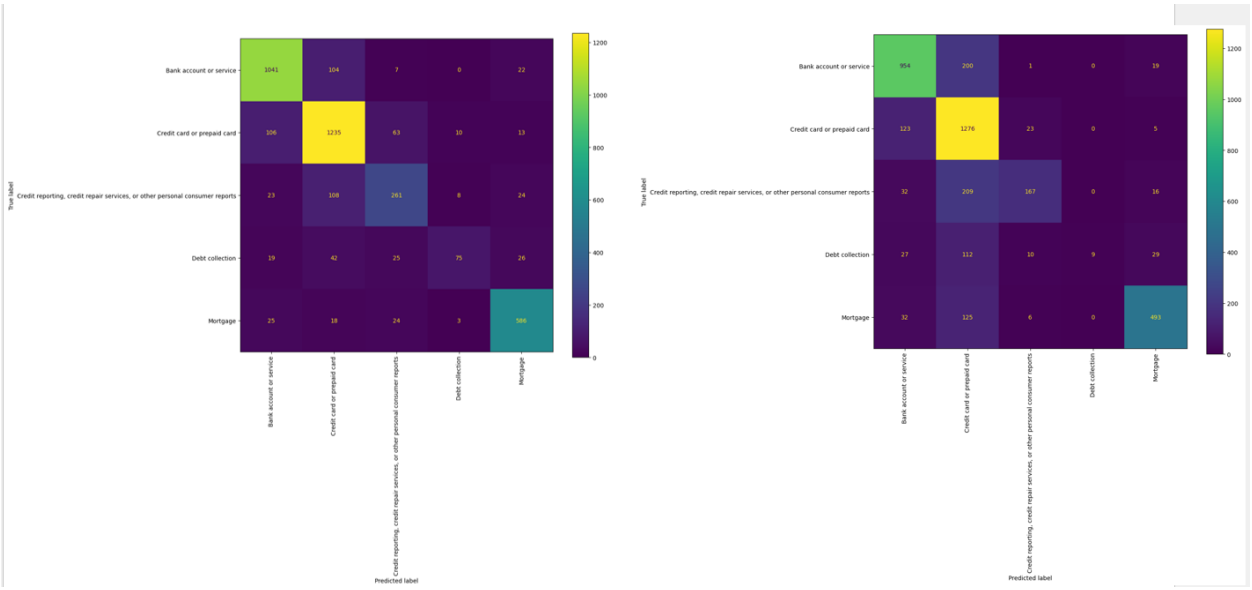


Fig: 3.5 Confusion Matrix for Logistic Regression and Random Forest.

Fig: 3.5 show the confusion matrix of LR and RF multi-class classification respectively. The count in each block, represents the prediction count between predicted category on X-axis vs predicted label on Y axis. The models are said to be more efficient if the count is higher diagonally across the confusion matrix, indicating that each category is getting predicted accurately with high precision. We can see that LR classifies all classes relatively well. In contrast, RF is better in classifying categories with high samples like ‘Credit card or prepaid card’ and ‘Bank Accounts and Services’.

BERT Large Language Model Implementation

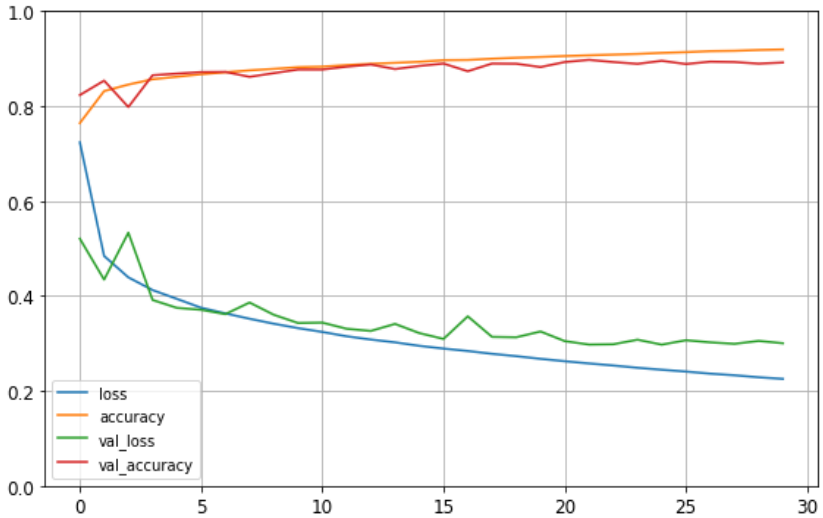


Fig: 3.6 Accuracy for BERT.

Fig: 3.6 shows the AUC results of our fine-tuned model with the validation test dataset over 30 epochs. We can see that the model shows an increase in AUC from 80% initially to 87% in the final epoch, showing improvement in performance over both LR and our RF model. The loss, which is the

error in predicting the true label decreases over the epochs as our model learns to improve classification from the loss function.

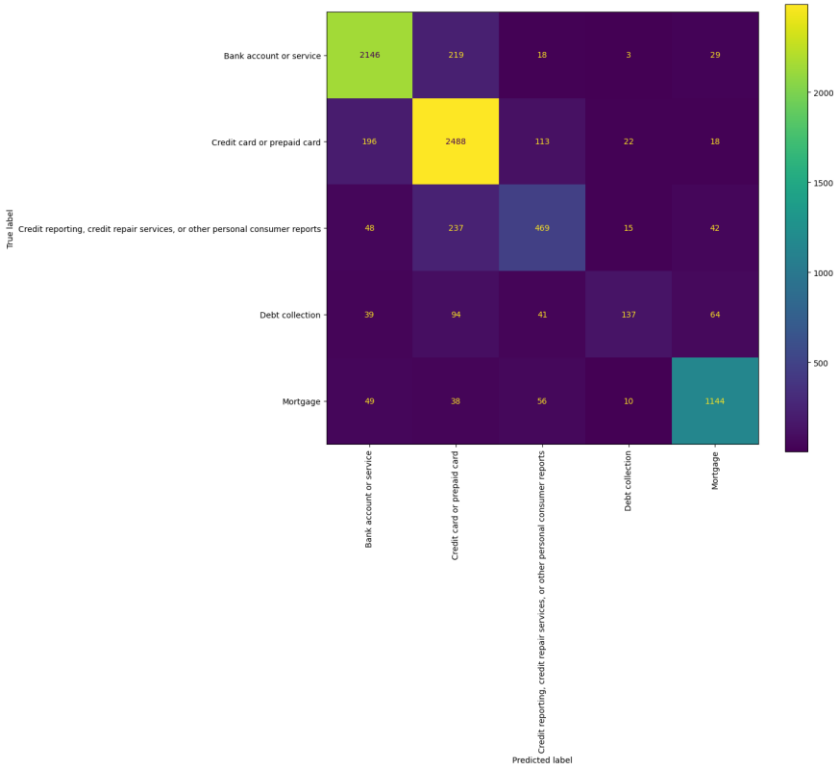


Fig: 3.7 Confusion Matrix for BERT.

Fig: 3.7 shows confusion matrix from the results of our BERT model predictions. We can see the difference between using a LLM and traditional models for multi-class classification. BERT is able to classify all 5 categories efficiently. The categories like ‘Debt Collection’ that have a lower sample size are predicted correctly with minimal misclassification as compared to LR and RF Fig:3.5.

III. DISCUSSIONS AND CONCLUSIONS

As a result of our findings, BERT showed improvement in accuracy and performance by 7% Moving to the advantages of BERT, BERT provides contextual understanding by considering the entire context of the input text. This is particularly advantageous in understanding the nuanced language often present in IT tickets, leading to improved classification accuracy.The other advantage is BERT captures semantic representations of words and phrases, allowing the model to understand the meaning behind the text. This is crucial for handling IT tickets where specific terms and phrases may carry different meanings based on the context.

On other hand, BERT models are computationally expensive, both in terms of training and inference. Logistic Regression, being a simpler model, is computationally more efficient. BERT models require substantial resources, including powerful GPUs, for efficient training and inference. Logistic Regression is more lightweight in terms of resource requirements.

BERT Language Models exhibit superior performance in IT ticket classification due to their contextual understanding and semantic representations, they come with computational challenges.

IV. FUTURE WORK

To take this project to next step, we can develop a chatbot using this model. This makes the project more accessible to the people especially who work at IT desk. As it minimizes the organization cost and increases efficiency in resolving the tickets by meeting client requirements.

In terms of cloud-based technologies like salesforce, we can develop queues and use this analysis to assign the tickets to the right team in improve the team performance.

We can also create a GUI or website and extend the project to the organization-wide by collecting more data. In such case, based on their project they can select the project and add the complaint description to get the team details.

V. REFERENCES

(Venkatasubramanian Sundara Mahadevan, 2021). Kaggle. Retrieved on September,2023. https://www.kaggle.com/datasets/venkatasubramanian/automatic-ticket-classification

(Paramesh, Ramya, Shreedhara, 2018). "*Classifying the Unstructured IT Service Desk Tickets Using Ensemble of Classifier*s". IEEE. https://arxiv.org/pdf/2103.15822

(Jeremy Howard, Sebastian Ruder, 2018). "*Universal Language Model Fine-tuning for Text Classification"*. ArXiv. https://arxiv.org/pdf/1801.06146.pdf

(May Phu Paing, 2018)Comparison of Sampling Methods for Imbalanced Data Classification in Random Forest https://ieeexplore.ieee.org/abstract/document/8609946/

(Andreas C. Müller**,** Sarah Guido, 2016**)**. Introduction to Machine Learning with Python: A Guide for Data Scientists

(Hugging face).Bert-base-uncased. https://huggingface.co/bert-base-uncased

(Federico Pascual, 2019). *Automate Ticket Classification with AI*.

https://monkeylearn.com/blog/ticket-classification-with-ai/

(Helpdesk). The Best Three Customer Ticket Assignment Strategies

https://www.helpdesk.com/learn/customer-support-essentials/customer-ticket-assignment-strategies/

APPENDICES

Programming Codes:

Python 3.11.0 – *Google Collaboratory*

Github Link: https://github.com/Mohd-Owais-Shaikh/Data606-IT-ticket-classification-using-LLM/tree/main

Kaggle Link : *https://www.kaggle.com/code/karna107/umbc-rockers*

Problem Statement - The main objective of this study is to build a classification model for support tickets in IT.

Importing Libraries

```
#importing all the necessary libraries
from google.colab import drive
drive.mount('/content/drive')
!pip install light-the-torch >> /.tmp
!ltt install torch torchvision >> /.tmp
!pip install fastai --upgrade >> /.tmp
!pip3 install tqdm
!pip3 install scikit-learn
!pip3 install numpy
!pip3 install genism
!pip install transformers
!pip install torch
!pip3 install pandas
!pip3 install regex
!pip3 install contractions
import pandas as pd
import json
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import nltk
import string
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
import regex as re
import contractions
import contractions
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
import tensorflow as tf
import timeit
import seaborn as sns
import matplotlib.pyplot as plt
```

Dataset

```
#Loading the file
file_path = '/content/drive/MyDrive/606/complaints-2021-05-14_08_16_.json'
with open(file_path, 'r') as file:
    # Opening the file from the path and loading it into data object
    data = json.load(file)
#creating a dataframe
df = pd.DataFrame(data)

data_ = pd.read_json('/content/drive/MyDrive/606/complaints-2021-05-14_08_16_.json')
Initial_df: pd.DataFrame = data_["_source"].apply(pd.Series)
Initial_df.head()#displays the top rows in dataset

selected_columns = ['complaint_id', 'issue', 'date_received', 'product', 'company_response', 'company',
'sub_product', 'timely', 'complaint_what_happened', 'sub_issue', 'consumer_consent_provided']#filtering
the columns
Initial_df= Initial_df[selected_columns]#adding the filtered columns to the initial_df
Initial_df.head()#displays the modified the dataframe

product_counts = Initial_df['product'].value_counts()#counting the records in product
plt.figure(figsize=(8, 4))
plt.pie(product_counts, labels=None, autopct='%1.1f%%') #plotting a pie chart
plt.title('Distribution of Products')#assigning title to plot
plt.axis('equal')
plt.legend(labels=product_counts.index, loc='upper right', bbox_to_anchor=(2, 0.5))
plt.show()#displays plot

top_product_categories = Initial_df['product'].value_counts().nlargest(7).index
Initial_df = Initial_df[Initial_df['product'].isin(top_product_categories)]
Initial_df['product'] = Initial_df['product'].replace('Checking or savings account', 'Bank account or
service')
Initial_df['product'] = Initial_df['product'].replace('Credit card', 'Credit card or prepaid card')
Initial_df.head()
```

Data Cleaning:

```
Initial_df.isnull().sum()
#checking the null values
null_values = Initial_df['product'].isnull().sum()
#printing the null values count in product column
print("Number of null values in the 'product' column:", null_values)
#listing out unique complaints
Initial_df['complaint_what_happened'].nunique()
filtered_df = Initial_df[Initial_df['complaint_what_happened'] != '']
filtered_df
```

Feature and Target Variable:

```
#displays the target and feature
print('Complaint:', filtered_df['complaint_what_happened'][14])
print('\nProduct category for complaint:',filtered_df['product'][14])
```

Target Variable Distribution

```
product_counts = filtered_df['product'].value_counts()
plt.figure(figsize=(8, 4))
#displays pie plot
plt.pie(product_counts, labels=None, autopct='%1.1f%%')
#assigning title to pie charts
plt.title('Distribution of Products')
plt.axis('equal')
plt.legend(labels=product_counts.index, loc='upper right', bbox_to_anchor=(2, 0.5))
#displays plot
plt.show()
plt.figure(figsize=(12, 6))
sns.countplot(x='product', data=filtered_df, order=filtered_df['product'].value_counts().index)
plt.title('Distribution of Products')
plt.xlabel('Product')#assigning x label
plt.ylabel('Count') #assigning y label
plt.xticks(rotation=45, ha='right')
#displays plot
plt.show()
```

Data Pre-Processing:

```
Cleaned_df= filtered_df.loc[:, ['product', 'complaint_what_happened']]
Cleaned_df
```

Tokenizing

```
# importing necessary libraries

import nltk
nltk.download('punkt')
nltk.download('stopwords')
Cleaned_df ['lower'] = Cleaned_df ['tokenized'].apply(lambda x: [word.lower() for word in x])
Cleaned_df.head()
# cleaning the data using Lambda function
punc = string.punctuation
Cleaned_df ['no_punc'] = Cleaned_df ['lower'].apply(lambda x: [word for word in x if word not in
punc])
Cleaned_df.head()
```

#Removing Stopwords

```
stop_words = set(stopwords.words('english'))
Cleaned_df['stopwords_removed'] = Cleaned_df['no_punc'].apply(lambda x: [word for word in x if word not in stop_words])
# cleaning the data using Lambda function
Cleaned_df.head()
Cleaned_df['cleaned'] = Cleaned_df['stopwords_removed'].apply(lambda row: ' '.join(row))
Cleaned_df['cleaned'][1]

#removing the stopwords
from tqdm import tqdm
from nltk.corpus import stopwords
processed_body = []
# Data Cleaning
for i in tqdm(Cleaned_df['cleaned']):
    i=re.sub('(<[\w\s]*/?>)',' ',i)
    i=contractions.fix(i)
    i=re.sub('[^a-zA-Z0-9\s]+',' ',i)
    i=re.sub('\d+',' ',i)
    i=re.sub(' +', ' ', i)
    i=re.sub(r'(xxxx\s*)+', ' ', i)
    processed_body.append(' '.join([j.lower().translate(str.maketrans('', '', string.punctuation)) for j in i.split()

Final dataset
#creating a final dataset
final_dataset = pd.DataFrame({'Complaints': processed_body, 'Target':Cleaned_df['product']})
final_dataset

IF-IDF Vectorization
# importing necessary libraries
from sklearn.preprocessing import MaxAbsScaler
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.pipeline import Pipeline
from sklearn.decomposition import TruncatedSVD

pipeline = Pipeline([
    ('vectorizer', CountVectorizer(binary=True)),
    ('tfidf', TfidfTransformer())
     ,('scaler', MaxAbsScaler())
    ,('TSVD', TruncatedSVD(n_components=250))
])
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

Test-Train Split for Logistic Regression
```

```
final_dataset['Target_encoded'] = label_encoder.fit_transform(final_dataset['Target'])
x_train = pipeline.fit_transform(final_dataset['Complaints'], final_dataset['Target'])
x_train.shape
#displays shape of the train dataset

# importing necessary libraries

from sklearn.model_selection
import train_test_split
dX_train, dX_test, dy_train, dy_test = train_test_split(x_train,
                                        final_dataset['Target'],
                                        test_size=0.4,
                                        random_state=42
                                       )
dy_test.value_counts()
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver='liblinear')
log_reg = log_reg.fit(dX_train, dy_train)

log_reg_score = log_reg.score(dX_test, dy_test)
print(f'Training Accuracy Score: {log_reg_score:.2%}') # printing accuracy

#confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
y_pred = rf.predict(dX_test)
classes = np.unique(dy_test)
fig, ax = plt.subplots(figsize=(12, 10))
cm = confusion_matrix(dy_test, y_pred, labels=classes)
cmp = ConfusionMatrixDisplay(cm, display_labels=classes)
cmp.plot(ax=ax)
plt.xticks(rotation='vertical')
plt.show()

Random Forest
# importing necessary libraries

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=13)
rf.fit(dX_train, dy_train)
rf_score = rf.score(dX_test, dy_test)
print(f' Training Score: {rf_score:.2%}')
```

Cross Validation

```
# importing necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

def generate_estimates(comp=5):

    modeling_pipeline = Pipeline([
        ('model', LogisticRegression())
       ]
    )

    return modeling_pipeline

# importing necessary libraries
from sklearn.model_selection import cross_validate

clf = generate_estimates()
cv_results = cross_validate(clf, dX_train, dy_train,
                            scoring=['accuracy', 'f1_macro'],
                            cv=5,
                            error_score='raise')

cv_results
```

Grid Search

```
# importing necessary libraries

from sklearn.model_selection import GridSearchCV
log_reg = LogisticRegression(solver='liblinear')
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100]
}
grid_search = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy')
grid_search.fit(dX_train, dy_train)
print(f'Best parameters: {grid_search.best_params_}')
print(f'Best score: {grid_search.best_score_:.4f}')
#printing Score and parameters
```

Hyperparameter Tuned Logistic Regression

```
log_reg_hyper = LogisticRegression(C=1, penalty='l1')
log_reg_hyper = log_reg.fit(dX_train, dy_train)
log_reg_score = log_reg_hyper.score(dX_test, dy_test)
print(f'Training Accuracy Score: {log_reg_score:.2%}')
```

LLM

```
# print the GPU availability

print("GPU Available:", tf.test.is_gpu_available())
tpu_available = False
devices = tf.config.list_logical_devices()
for device in devices:
    if device.device_type == 'TPU':
        tpu_available = True
        break
# print the TPU availability

print("TPU Available:", tpu_available)
final_dataset['Target'].value_counts()
possible_labels = final_dataset.Target.unique()

label_dict = {}
for index, possible_label in enumerate(possible_labels):
    label_dict[possible_label] = index
label_dict
final_dataset['label'] = final_dataset.Target.replace(label_dict)
final_dataset

# importing necessary libraries
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(final_dataset['Complaints'],
                                                  final_dataset['label'],
                                                  test_size=0.3,
                                                  random_state=42 )

final_dataset['data_type'] = ['not_set']*final_dataset.shape[0]
```

BERT Tokenizer

```
# importing necessary libraries

from transformers import BertTokenizer
import torch
```

```
from torch.utils.data import TensorDataset
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
                                          do_lower_case=True)
# Train data
encoded_data_train = tokenizer.batch_encode_plus(
    final_dataset[final_dataset.data_type=='train'].Complaints.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    return_tensors='pt'
)
# validate data
encoded_data_val = tokenizer.batch_encode_plus(
    final_dataset[final_dataset.data_type=='val'].Complaints.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    return_tensors='pt'
)
input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(final_dataset[final_dataset.data_type=='train'].label.values)

input_ids_val = encoded_data_val['input_ids']
attention_masks_val = encoded_data_val['attention_mask']
labels_val = torch.tensor(final_dataset[final_dataset.data_type=='val'].label.values)
# Training data
dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_masks_val, labels_val)

# importing necessary libraries

from transformers import BertForSequenceClassification, BertTokenizer
# Bert Model
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
                                                      num_labels=len(label_dict),
                                                      output_attentions=False,
                                                      output_hidden_states=False)
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# labeling encoder
final_dataset['Target_encoded'] = label_encoder.fit_transform(final_dataset['Target'])
```

```
class_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
print("Class Mapping:", class_mapping)
# importing necessary libraries

import torch
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
# training models

dx_train = X_train.to_list()
dy_train = y_train.to_list()

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=17).to('cuda')  # Move the model to GPU

optimizer = AdamW(model.parameters(), lr=5e-5)
num_epochs = 1

for epoch in range(num_epochs):
  with tf.device('/device:GPU:0'):
    for i in tqdm(range(len(dx_train))):
        complaint = dx_train[i]
        target = dy_train[i]

        inputs = tokenizer(complaint, padding=True, truncation=True, return_tensors="pt").to('cuda')

        label = torch.tensor(target).unsqueeze(0).to('cuda')

        outputs = model(**inputs, labels=label)
        loss = outputs.loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

model.save_pretrained("bert_model")  # model save
tokenizer.save_pretrained("bert_model")
!zip -r /content/bert_file.zip /content/fine_tuned_bert_model
!mv /content/bert_file.zip /content/drive/MyDrive/606
```

BERT Model Training

```
# importing necessary libraries
```

```
import zipfile
import os
import torch
from transformers import BertTokenizer, BertForSequenceClassification

dx_val = X_val.to_list()

#defining model path
model_path = "/content/fine_tuned_bert_model"
model = BertForSequenceClassification.from_pretrained(model_path)
tokenizer = BertTokenizer.from_pretrained(model_path)
# model evaluation
model.eval()
predictions = []
with torch.no_grad():
  with tf.device('/device:GPU:0'):
    for i in tqdm(range(len(dx_val))):
        text = dx_val[i]
        inputs = tokenizer(text, padding=True, truncation=True, return_tensors="pt")
        outputs = model(**inputs)
        probs = torch.nn.functional.softmax(outputs.logits, dim=-1).squeeze().tolist()
        predicted_label = torch.argmax(outputs.logits).item()
        predictions.append(predicted_label)

# accuracy of the model
accuracy = accuracy_score(y_val, predictions)
f1_macro = f1_score(y_val, predictions, average='macro')
print(f'Accuracy: {accuracy:.4f}')
print(f'f1_score: {f1_macro:.4f}')
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
y_pred = log_reg.predict(dX_test)
classes = np.unique(dy_test)

# confusion matrices

fig, ax = plt.subplots(figsize=(12, 10))
cm = confusion_matrix(dy_test, y_pred, labels=classes)
cmp = ConfusionMatrixDisplay(cm, display_labels=classes)
cmp.plot(ax=ax)
plt.xticks(rotation='vertical')
plt.show()



# declare epoch value
```

```
epochs = 10
seed_val = 17
for epoch in tqdm(range(1, epochs+1)):
  with tf.device('/device:GPU:0'):
    model.train()  # model training
    loss_train_total = 0
    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch), leave=False,
disable=False)
    for batch in progress_bar:
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)
        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels': batch[2],
                 }
        outputs = model(**inputs)
        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch))})

# validating test and train losses
    torch.save(model.state_dict(), f'data_volume/finetuned_BERT_epoch_{epoch}.model')

    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'loss: {loss_train_avg}')

    val_loss, predictions, true_vals,accuracy = evaluate(dataloader_validation)
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'accuracy: {accuracy}')
    tqdm.write(f'val_loss: {val_loss}')
    tqdm.write(f'val_accuracy: {val_f1}')

accuracy = cv_results['val_accuracy']

all_scores = [accuracy]
metric_names = ['Accuracy']
# Plotting figures
plt.figure(figsize=(10, 6))
plt.boxplot(all_scores)
plt.title('Modeling Scores from Cross Validation')
```

```
plt.xticks(range(1, len(metric_names) + 1), metric_names)
plt.ylabel('Score')
plt.show()

# importing necessary libraries


import pandas as pd

pd.DataFrame(model.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
save_fig("keras_learning_curves_plot")
plt.show()
```