ViaClaro

A Career Coach Chatbot

Abeeha Zehra
az07728@st.habib.edu.pk

Batool Ali Akbar
ba07612@st.habib.edu.pk

I. Introduction

The intricacies of career planning and job finding are quite daunting, especially for the individuals new to the job market. Individuals often struggle with effectively planning their career trajectory and locating suitable job opportunities. Even though traditional career counselling is in place, it can be quite expensive, time-consuming, and may not provide personalized support. To address these challenges, there is a need for more a accessible, efficient, and customized career support solution.

II. Problem Definition

The conventional approach to career counseling, while helpful, is often expensive and not that readily available. Many job seekers, particularly those starting out, struggle with aligning their abilities and interests to potential career paths and relevant job opportunities. Our solution, ViaClaro is a Chatbot that tries to overcome these limitations by leveraging advanced technologies like Machine Learning, Natural Language Processing, and Large Language Models. This tool will suggest potential career paths and relevant job opportunities based on user's resumes, and will offer a basic road map to help them achieve their career goals efficiently.

III. Baseline Approach

ViaClaro is structured around three key components. First, it analyzes user resumes to suggest the top five potential career options based on the individual's skills and interests. Once the predictions are made, the user can view a basic road map for the selected career path. This road map will be generated using the OpenAI API. Finally, it will suggest open job opportunities that align with user's skills and interests. However, for the baseline version, we have only implemented two components: accepting resumes as an input and using our model to suggest the top five career paths while providing a road map for the selected career.

IV. Data Description

This project uses two major datasets. The first dataset comprises of previous job postings from LinkedIn, Glassdoor and other popular job posting websites. The dataset has been downloaded from Kaggle.The second is the resume dataset that has been taken from Kaggle and Kaggle.

The resume datasets are csv format files with 2484 and 962 data entries respectively. The first CSV file, titled "Resume" consists of four columns: ID, resume in string format, resume in HTML format, and the career category or domain of the resume. The second CSV file named "UpdateResumeDataset" has two columns: the resume in string format and the career category. We have combined these two datasets to create a dataset of 3446 entires using the following function.

```python

def combine_files(file1, file2, column1, column2):

    df1 = pd.read_csv(file1)
    df2 = pd.read_csv(file2)

    df1[column1] = df1[column1].fillna('').apply(
    lambda x: cleanResume(x) if isinstance(x, str)
    else '')
    df2[column1] = df2[column1].fillna('').apply(
    lambda x: cleanResume(x) if isinstance(x, str)
    else '')

    df_combined = pd.concat([df1[[column1, column2
    ]], df2[[column1, column2]]], ignore_index=True)

    return df_combined

```

Listing 1: Combining Files

The merged file contains a dataset containing two columns: resume represented as a string and its corresponding career category or domain.

The resume string is processed by the following function to remove stop words, links, special characters, and extra spaces.

```python

def cleanResume(txt):

    cleanText = re.sub(r'http\S+\s', ' ', txt)
    cleanText = re.sub(r'RT|cc', ' ', cleanText)
    cleanText = re.sub(r'#\S+\s', ' ', cleanText)
    cleanText = re.sub(r'@\S+', '  ', cleanText)
    cleanText = re.sub(r'[!"#$%&\'()*+,\-./:;<=>?@
    [\\\]^_`{|}~]', ' ', cleanText)
    cleanText = re.sub(r'[^\x00-\x7f]', ' ',
    cleanText)
    cleanText = re.sub(r'\s+', ' ', cleanText)

    return cleanText

```

Listing 2: Data Cleaning

This cleaned string is then given to the following function to create a vocabulary and tf-idf vectors for each word.

```python

def creating_vectors(df_combined, column1):

    tfidf = TfidfVectorizer(stop_words='english')

    tfidf.fit(df_combined[column1])
    requiredText = tfidf.transform(df_combined[
     column1])

    return requiredText, tfidf

```

Listing 3: Preprocessing

The second column represents the corresponding category or the career domain for each resume. The categories are HR, Designer, Information Technology, Teacher, Advocate, Business Development, Healthcare, Fitness, Agriculture, BPO, Sales, Consultant, Digital Media, Automobile, Chef, Finance, Apparel, Engineering, Accountant, Construction, Public Relations, Banking, Arts, Aviation, Data Science, Web Designing, Mechanical Engineer, Health and Fitness, Civil Engineer, Java Developer, Business Analyst, SAP Developer, Automation Testing, Electrical Engineering, Operations Manager, Python Developer, DevOps Engineer, Network Security Engineer, PMO, Database, Hadoop, ETL Developer, DotNet Developer, Blockchain, and Testing. Each category is assigned an index by the following function.

```python

def assigning_categories(df,column):

    le = LabelEncoder()

    df[column] = le.fit_transform(df[column])
    unique_values = df[column].unique()

    return le, unique_values

```

Listing 4: Indexing Categories

This data is then split into an 80 to 20 ratio for training and testing by the following line of code. stratify=df_combined['Category'] is used to ensure that each category is split in an equal ratio of 80 to 20 between the training and the testing set. This reduces the biasness caused due to unequal distribution of categories and improves the generalizability of the model over imbalanced datasets.

```python

X_train,X_test, y_train, y_test = train_test_split(
    required_text, df_combined['Category'],
    test_size=0.2, random_state=42, stratify=
    df_combined['Category'])

```

Listing 5: Indexing Categories

The count of each category in the training and testing dataset is as follows.

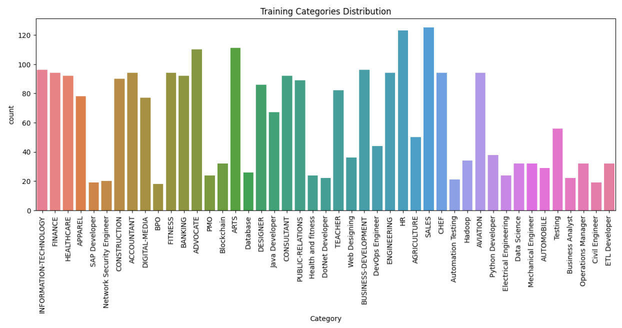

Fig. 1: Training Set Categories

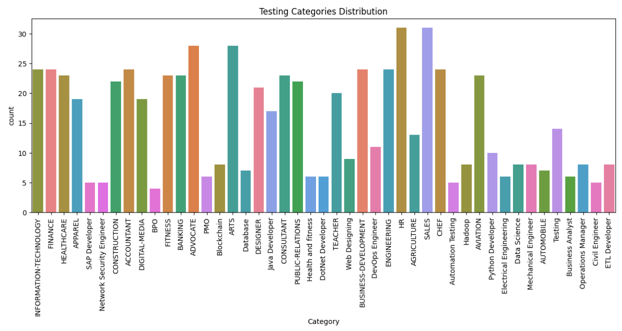

Fig. 2: Test Set Categories

V. Implementation Details

The project uses the built in models provided by the sklearn library to train and test three models. Each of these models are saved in their corresponding pkl files.

The first model is the Artificial Neural Network. The model has 1 hidden layer with 100 neurons and uses the RELU as an activation function. It also uses the Adam Optimization for better predictions.

```python

def ann(X_train,X_test,y_train,y_test):

    ann_clf = MLPClassifier(hidden_layer_sizes
    =(100,), max_iter=500, activation='relu', solver
    ='adam')

    ann_clf.fit(X_train, y_train)
    y_pred = ann_clf.predict(X_test)

    with open('ann_clf.pkl', 'wb') as ann_file:
        pickle.dump(ann_clf, ann_file)

    return y_pred

```

Listing 6: ANN Model

The second model is a Multinomial Naive Bayes Model.

```python

def naive_bayes(X_train,X_test,y_train,y_test):

    nb_clf = MultinomialNB()

    nb_clf.fit(X_train, y_train)
    y_pred = nb_clf.predict(X_test)

    with open('nb_clf.pkl', 'wb') as nb_file:
        pickle.dump(nb_clf, nb_file)

    return y_pred

```

Listing 7: Naive Bayes Model

The third model is the Logistic Regression which runs on a maximum of 1000 iterations.

```python

def logistic_regression(X_train, X_test, y_train,
    y_test):

    lr_clf = LogisticRegression(max_iter=1000)

    lr_clf.fit(X_train, y_train)
    y_pred = lr_clf.predict(X_test)

    with open('lg_clf.pkl', 'wb') as lr_file:
        pickle.dump(lr_clf, lr_file)

    return y_pred

```

Listing 8: Logistic Regression Model

Once the training is done, we utilize the best-performing model i.e. Artificial Neural Network to make predictions. The following chunk of code depicts how the final prediction process works.

```python

pdf = select_pdf_file()
if pdf:
    myresume = pdf_to_string(pdf)

with open('tfidf.pkl', 'rb') as tfidf_file:
    tfidf = pickle.load(tfidf_file)
with open('ann_clf.pkl', 'rb') as ann_file:
    ann_clf = pickle.load(ann_file)
with open('label_encoder.pkl', 'rb') as le_file:
    le_loaded = pickle.load(le_file)

cleaned_resume = cleanResume(myresume)
input_features = tfidf.transform([cleaned_resume])
predicted_probabilities = ann_clf.predict_proba(
    input_features)

top_n = 5
top_n_indices = np.argsort(predicted_probabilities
    [0])[-top_n:][::-1]
top_categories = le_loaded.inverse_transform(
    top_n_indices)
top_probabilities = predicted_probabilities[0][
    top_n_indices]

for category, probability in zip(top_categories,
    top_probabilities):
    print(f"Category: {category}, Probability: {
    probability:.4f}")

```

```python

index = int(input("Choose profession to view the
    career path: "))
print(top_categories[index-1])
print(generate_roadmap(top_categories[index-1]))


```

Listing 9: Predicting Careerpath and Generating Roadmap

The user is first prompted with a window to upload a PDF file, typically a resume, which is then converted into a single string and passed to the cleanResume function (Listing 2).

```python

def select_pdf_file():

    root = tk.Tk()

    root.withdraw()
    file_path = filedialog.askopenfilename(filetypes
    =[("PDF files", "*.pdf")])

    return file_path

```

Listing 10: PDF Input

```python

def pdf_to_string(file_path):

    with open(file_path, "rb") as file:
        reader = PyPDF2.PdfReader(file)
        text = ""
        for page in reader.pages:
            text += page.extract_text() + "\n"

    return text

```

Listing 11: PDF to String

This cleaned resume is then transformed into numerical features using a tf-idf vectorizer.

```python

with open('tfidf.pkl', 'rb') as tfidf_file:
    tfidf = pickle.load(tfidf_file)

input_features = tfidf.transform([cleaned_resume])

```

Listing 12: tf-idf Vectorization

The best of the three models i.e. Arificial Neural Network is then used to predict the probabilities for each career category and only the top 5 are shown to the user. The user can then choose to explore a career road map by inputting its corresponding index.

```python

with open('lg_clf.pkl', 'rb') as ann_file:
    lg_clf = pickle.load(ann_file)

predicted_probabilities = lg_clf.predict_proba(
    input_features)

top_n = 5
top_n_indices = np.argsort(predicted_probabilities
    [0])[-top_n:][::-1]

top_categories = le.inverse_transform(top_n_indices)
```

```
top_probabilities = predicted_probabilities[0][
    top_n_indices]

index = int(input("Choose profession to view the
    career path: "))

```

Listing 13: Prediction

The road map is generated by sending a prompt to the GPT-3.5-turbo model via OpenAPI.

```

def generate_roadmap(domain):

    openai.api_key = ''

    if (openai.api_key != ''):

        prompt = f"Create a comprehensive roadmap
    for ${domain}. The roadmap should contain all
    the key skills to excel in this profession. Also
     mention the dependencies between these skills."

        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "user", "content": prompt}
            ]
        )

        roadmap = response['choices'][0]['message'][
    'content']

        return roadmap
    else:
        return "No API Key."

```

VI. Evaluation

For the evaluation of the above mentioned models, we have used a confusion matrix and calculated the recall, precision, accuracy and f-1 scores for each model. The results are as follows:

Evaluation	Score
Accuracy	0.75
Precision	0.82
Recall	0.80
F1 score	0.80

Table 1: Artificial Neural Network

Evaluation	Score
Accuracy	0.57
Precision	0.58
Recall	0.51
F1 score	0.51

Table 2: Naive Bayes

Evaluation	Score
Accuracy	0.76
Precision	0.82
Recall	0.80
F1 score	0.79

Table 3: Logistic Regression

VII. Analysis and Insights

To achieve the goals of this project, we tried three different models; Artificial Neural Network, Multinomial Naive Bayes, and Logistic Regression.

An Artificial Neural Network (ANN) is a machine learning model that mimics the brain's structure to identify complex patterns. The ANN typically takes encoded word vectors as inputs and passes them through multiple layers of interconnected neurons. For this project we have used a model with one hidden layer of 100 neurons to learn relationships between words and professions. The model is given the tf-idf encoded vectors as an input and it predicts the probabilities of each category. The ReLU activation function is applied to introduce non-linearity, allowing the network to capture more complex patterns, while the Adam optimizer is used for efficient training.

Multinomial Naive Bayes is a classification algorithm that applies Bayes' Theorem, assuming that features are conditionally independent given the class. In predicting career pathways from resumes, the algorithm uses the frequency of words to calculate the likelihood of each profession. It uses the training data to predict which words are the most indicative of certain professions and assigns probabilities accordingly.

Logistic Regression is a linear classification algorithm that predicts the probability of a data point belonging to a specific class. In the current context, it works by fitting a linear model to the training data, where each resume is represented by a feature vector of tf-idf values. The algorithm estimates the likelihood that a resume belongs to each profession category based on these features.

The following graph depicts a comparison between the three models

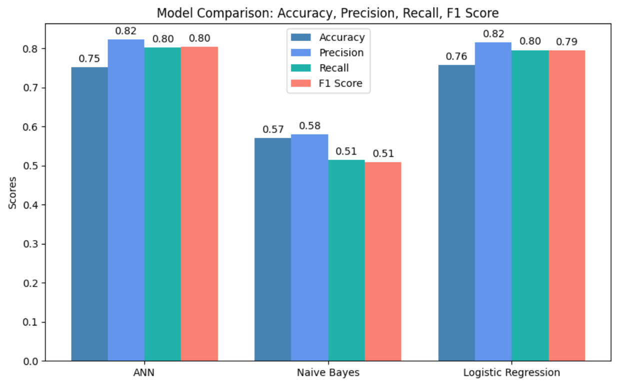

Fig. 3: Comparison between different models

From the above graph, it can be seen that Logistic Regression achieved an accuracy of 0.76, with a precision of 0.82, recall of 0.80, and an f-1 score of 0.79. While its accuracy is slightly higher than that of the ANN, the model falls behind in the f-1 score, which reflects its balance between precision and recall. This slight dip suggests that Logistic Regression might not be as effective in handling the nuances of classifying the most relevant career paths, even though it provides slightly better overall accuracy.Regression.

ANN performed slightly better in terms of the f-1 score, with an accuracy of 0.75, precision of 0.82, recall of 0.80, and an f-1 score of 0.80. Although it has marginally lower accuracy, its higher f-1 score indicates that ANN strikes a better balance between precision and recall. This improvement suggests that ANN is better at capturing complex relationships in the data, potentially leading to more accurate predictions of the top career paths.

In contrast, Naive Bayes lagged significantly, recording an accuracy of 0.57, precision of 0.58, and recall and an f-1 score of 0.51. Its under-performance can be attributed to its assumption of feature independence, which does not adequately reflect the inter-dependencies present in resume data, resulting in a higher number of mis-classifications.

Thus, we can conclude that ANN is the more effective model for this classification task. Despite Logistic Regression having slightly higher accuracy, ANN's superior F1 score indicates a better balance between precision and recall, which is crucial for recommending the top career paths based on resume input. The ability of ANN to capture non-linear relationships in the data makes it more suitable for this task, as it can better account for the complex patterns present in resume features and career categories. Therefore, ANN proves to be the better choice for predicting career pathways from resumes.

VIII. Future Work

The current implementation covers the first two features of the entire project i.e. it takes user resume from console and outputs the top 5 potential career options to the console. Moreover, the user can input any number between 1 and 5 to view a basic road map for the chosen career. This too, will be displayed on the console. As the project progresses, we plan on creating a proper interface for the user to upload their resumes and view the basic road map for the chosen career. Furthermore, we will also be working on suggesting open job opportunities based on the interests and skills of the user.