Candidate Profile Compatibility Predictor

Shivani Mundle, Nikhil Narayanarao Kudupudi, Sharanya Sentil

Abstract

This project addresses the challenge of efficient job searching through advanced Natural Language Processing (NLP) models. The project includes techniques such as Skip-gram embeddings, LSTM, Bidirectional LSTM, BERT, and a custom transformer architecture, the project evolves from traditional methods to sophisticated models. The custom transformer with 2 MultiHead Attention layers emerges as the most effective, achieving an 81% average accuracy in classifying resumes. The findings contribute insights into optimizing predictive capabilities for enhanced job compatibility assessment in recruitment processes.

Introduction

In this dynamic and competitive job market, finding the right talent and suitable opportunities poses a significant challenge. For job seekers such as college students or professionals, navigating through the heaps of job listings to find a position that aligns with their profile and promises fulfilling work experience can be a challenging task. From our collective experiences our group decided to develop a tool that predicts the compatibility between a candidate's profile and a given job description.

As traditional recruitment methods often face limitations to both identifying the most suitable candidates or the candidate finding the most suitable job, our project aims to simplify this process. We are focusing on tools that allow inputting both job descriptions and resumes. The tool will then generate a compatibility score, indicating the degree of alignment between a candidate's skills and the specific requirements of a job. By leveraging compatibility scoring, our tool aims to simplify and enhance the hiring process for both candidates and companies, making it more efficient and targeted.

Our approach involves using various models, from the more traditional Skip-gram embeddings to the advanced BERT transformer. First, we aim to better understand each resume by employing techniques like LSTM, Bidirectional LSTM, pretrained transformers like Bert and custom transformer architecture. Once resumes are categorized, we delve into assessing how closely they align with job requirements, using straightforward metrics like cosine similarity.

For the above task, we are leveraging state-of-the-art natural language processing using BERT (Bidirectional Encoder Representations from Transformers) to find similarity between a candidate's profile and a given job description. BERT allows us to capture nuanced semantic relationships within the textual data by providing context-aware embeddings for both job descriptions and candidate resumes. We utilized BERT's tokenization and embeddings to quantify the similarity between the job description and each resume. This approach offered a more sophisticated analysis that considers the contextual meaning of words and phrases, thereby elevating the precision of our predictive model.

In addition to the BERT and word embeddings approach, we explored the utilization of FastText, a powerful word representation model developed by Facebook. FastText captures word embeddings and the subword information, making it particularly effective for handling out-of-vocabulary words and providing more nuanced context.By incorporating FastText, our project benefits from its ability to handle morphological variations and capture subword information.

Through this research, we aim to offer practical insights that can refine the way resumes are evaluated, ultimately streamlining the recruitment process for better, more informed decision-making.

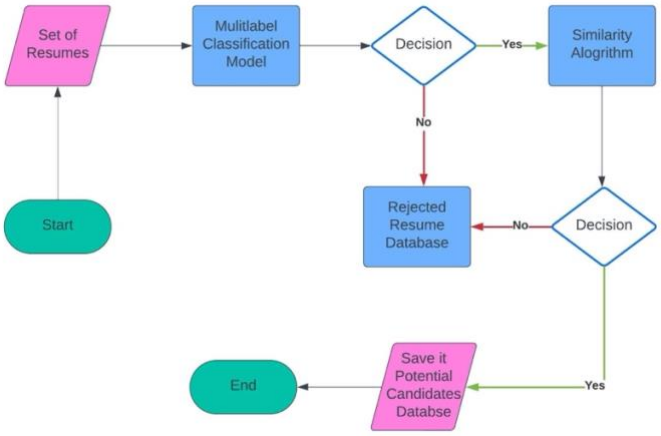

Fig.1 Project Flow Diagram

Keywords: Natural Language Processing, Resume Analysis, BERT, Recruitment, NLP, Skip-gram Embeddings, LSTM.

Background

In this section, we very briefly study different blocks that were used to build our models in this paper

A. Metrics

1. Precision
   Precision is a measure of the accuracy of the positive predictions made by the model. It calculates the ratio of

correctly predicted positive instances to the total instances predicted as positive.

Formula:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Interpretation:
Precision in resume classification measures the accuracy of the model in correctly identifying candidates for specific roles. For instance, if the precision for 'Java Developer' is 0.85, it means that when the model predicts a candidate is a Java Developer, it is accurate 85% of the time. High precision ensures that the selected candidates indeed have the specified skills and qualifications for roles such as 'Java Developer' or 'Database Administrator,' minimizing the risk of false positives in the hiring process.

2. Recall
Recall measures the ability of the model to capture all the relevant positive instances. It calculates the ratio of correctly predicted positive instances to the total actual positive instances.

Formula:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Interpretation:
Recall in resume classification indicates how well the model captures all instances of candidates with specific skills related to roles. If the recall for 'Network Administrator' is 0.80, it means that the model successfully identifies 80% of candidates with skills suited for network administration roles. High recall ensures that candidates with relevant skills for roles like 'Web Developer' or 'Security Analyst' are not overlooked, contributing to a comprehensive talent identification process.

3. F1 Score:
The F1 score is the harmonic mean of precision and recall. It provides a balanced measure that considers both false positives and false negatives.

Formula:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Interpretation:
The F1 score, being a balanced measure, is crucial for roles where precision and recall are both important. If the F1 score for 'Software Developer' is 0.75, it suggests that the model performs well in terms of both precision and recall for this role. In practical terms, a high F1 score ensures that candidates selected for roles like 'Front-End Developer' or 'Python Developer' not only possess the specified skills but also minimizes the risk of missing out on candidates with those skills, providing a more robust screening process.

Application in Multi-Label Classification

In multi-label classification, where each instance can belong to multiple classes, these metrics are extended to consider the presence of multiple labels. Micro-averaging and macro-averaging are common approaches

Micro-average: Computes the metrics globally by considering all instances and all classes together.
Macro-average: Computes the metrics independently for each class and then averages them.

B. Embeddings

1. Skip-gram Word Embeddings:
Skip-gram is a neural network-based word embedding model that belongs to the word2vec family. It is designed to learn distributed representations of words in continuous vector spaces. The primary objective of Skip-gram is to predict context words (words surrounding a target word), allowing the model to capture semantic relationships and similarities between words.

Key Characteristics:
- Learns by predicting context words given a target word.
- Effective in capturing semantic relationships and word similarities.
- Suitable for tasks requiring fine-grained representations of word meanings.
- Popular in natural language processing (NLP) applications such as sentiment analysis and machine translation.

2. GloVe (Global Vectors for Word Representation):
GloVe, short for Global Vectors for Word Representation, is an unsupervised learning algorithm for obtaining vector representations of words. Unlike Skip-gram, GloVe is based on global statistical information rather than local context windows. It aims to capture the co-occurrence statistics of words in a corpus.

Key Characteristics:

- Learns word embeddings by optimizing a global objective considering word co-occurrence probabilities.
- Particularly efficient in capturing global word relationships and semantic structures.
- Well-suited for tasks where a broader understanding of word co-occurrence is crucial.
- Widely used in applications like document clustering and information retrieval.

C. Deep Learning Models

1. Recurrent Neural Networks (RNNs)

Description: RNNs are a class of neural networks designed for sequence modeling, making them well-suited for tasks involving sequential or time-series data. However, traditional RNNs suffer from the vanishing gradient problem, limiting their ability to capture long-range dependencies in sequences.

Given a sequence of inputs x_t, hidden states h_t, and output y_t, the update equations for a simple RNN can be expressed as follows:

$$h_t = \tanh(W_{ah}h_{t-1} + W_{ax}x_t + b_h)$$
$$Y_t = \text{softmax}\big(W_{yh}h_t + b_y\big)$$

2. Long Short-Term Memory (LSTM) Networks

Description: LSTMs were introduced to address the vanishing gradient problem in RNNs. They include memory cells and gating mechanisms that allow them to capture long-term dependencies more effectively, making them particularly suitable for tasks where context over extended sequences is crucial.

LSTMs have three gates (input, forget, and output gates) and a memory cell. The update equations are as follows:

$$f_t = \sigma\big(W_f[h_{t-1}, x_t] + b_f\big)$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$c_t = f_t * c_{t-1} + i_t * \tanh(W_c[h_{t-1}, x_t] + b_c)$$
$$h_t = o_t * \tanh(c_t)$$

Here, σ denotes the sigmoid activation function, $*$represents element-wise multiplication, and W and b are weight matrices and bias vectors, respectively.

3. Bidirectional LSTM Networks

Description: Bidirectional LSTMs enhance sequence modeling by processing input sequences in both forward and backward directions. This allows the network to capture information from both past and future contexts, offering a more comprehensive understanding of the sequence.

Bidirectional LSTMs combine the outputs of both forward (h_t^{forward}) and backward (h_t^{backward}) LSTMs at each time step:

a. Forward LSTM

The forward LSTM processes the input sequence from left to right.

$$f_t^{\text{forward}} = \sigma\big(W_f^{\text{forward}}\big[h_{t-1}^{\text{forward}}, x_t\big] + b_f^{\text{forward}}\big)$$
$$i_t^{\text{forward}} = \sigma\big(W_i^{\text{forward}}\big[h_{t-1}^{\text{forward}}, x_t\big] + b_i^{\text{forward}}\big)$$
$$o_t^{\text{forward}} = \sigma\big(W_o^{\text{forward}}\big[h_{t-1}^{\text{forward}}, x_t\big] + b_o^{\text{forward}}\big)$$
$$c_t^{\text{forward}} = f_t^{\text{forward}} * c_{t-1}^{\text{forward}} + i_t^{\text{forward}} * \tanh\big(W_c^{\text{forward}}\big[h_{t-1}^{\text{forward}}, x_t\big] + b_c^{\text{forward}}\big)$$
$$h_t^{\text{forward}} = o_t^{\text{forward}} * \tanh\big(c_t^{\text{forward}}\big)$$

b. Backward LSTM

The backward LSTM processes the input sequence from right to left.

$$f_t^{\text{backward}} = \sigma\big(W_f^{\text{backward}}\big[h_{t+1}^{\text{backward}}, x_t\big] + b_f^{\text{backward}}\big)$$
$$i_t^{\text{backward}} = \sigma\big(W_i^{\text{backward}}\big[h_{t+1}^{\text{backward}}, x_t\big] + b_i^{\text{backward}}\big)$$
$$o_t^{\text{backward}} = \sigma\big(W_o^{\text{backward}}\big[h_{t+1}^{\text{backward}}, x_t\big] + b_o^{\text{backward}}\big)$$
$$c_t^{\text{backward}} = f_t^{\text{backward}} * c_{t+1}^{\text{backward}} + i_t^{\text{backward}} * \tanh\big(W_c^{\text{backward}}\big[h_{t+1}^{\text{backward}}, x_t\big] + b_c^{\text{backward}}\big)$$
$$h_t^{\text{backward}} = o_t^{\text{backward}} * \tanh\big(c_t^{\text{backward}}\big)$$

4. Transformers

Transformers represent a breakthrough in deep learning, introduced by Vaswani et al. in the paper "Attention is All You Need." Unlike traditional sequence models such as RNNs and LSTMs, transformers leverage self-attention mechanisms to capture long-range dependencies efficiently. They excel at parallelization, making them highly scalable and suitable for a wide range of tasks.

Given a sequence of input vectors $x_1, x_2, \ldots, x_n$ the self-attention mechanism computes attention scores (α_{ij}) for each pair of elements. The output (y_t) is then a weighted sum of input elements:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n}\exp(e_{ik})}$$

$$e_{ij} = \text{Attention}(Q_i, K_j)$$

$$y_i = \sum_{j=1}^{n} \alpha_{ij} \cdot V_j$$

Q, K, and V are linear projections of the input vectors, and d_k is the dimensionality of the key vectors.

5. BERT(Bidirectional Encoder Representations from Transformers)

   BERT, introduced by Devlin et al., is a pre-trained transformer-based model designed for natural language understanding tasks. It employs a bidirectional approach, considering context from both left and right directions, which sets it apart from traditional left-to-right language models. BERT has achieved state-of-the-art results on a variety of NLP benchmarks.

   BERT is pre-trained using a masked language model (MLM) objective. Given an input sequence with randomly masked tokens, the model is trained to predict the masked tokens:

$$\mathcal{L}_{MLM}(\theta) = \sum_{i=1}^{N} \log P\left(w_i \,\middle|\, \text{context}(w_i)\right)$$

   Here, N is the number of masked tokens, w_i represents a masked word, and context(w_i) is the surrounding context.

   BERT is also trained using a next sentence prediction objective, where the model predicts whether two sentences follow each other in the original text

$$\mathcal{L}_{NSP}(\theta) = -\sum_{i=1}^{M} \log P\left(\text{IsNext}(s_i, s_{i+1})\right)$$

Dataset

The dataset we used for this project is sourced from the resume corpus dataset[cite]. The resume corpus dataset consists of resume_samples.txt file containing the resumes for different occupations. The entries are in the format **reference_id:::occupations:::resume_text**. The dataset encapsulates a diverse array of professional experiences and backgrounds. In all there are 29780 resumes in the dataset.

Each entry in the dataset is characterized by a unique reference ID, offering distinct identification for every resume. The 'occupations' field contains a semicolon-separated list of roles associated with each resume, offering insights into the professional diversity of the candidates. This field is used for performing the initial classification of resumes. Furthermore, the 'resume_text' field contains unprocessed textual content, detailing candidates' skills, experiences, and qualifications.

Our preprocessing of the text ensures uniformity and accuracy of the dataset for subsequent analyses. Initially, the resume text is converted to lowercase, mitigating inconsistencies due to case variations. Then, the function removes hyperlinks and email addresses, eliminating potential noise and irrelevant information. Special characters, numbers, and punctuation are systematically removed to maintain data consistency and streamline further analyses. Additionally, the function incorporates lemmatization, a process that reduces words to their base or root form, enhancing the ensures a clean resume text which can be used for further analysis.

Original Dataset without preprocessing:

C:\Workspace\java\scrape_indeed\dba_part_1\1.html#1:::Database Administrator;Database Administrator;Database administration;Database;Ms sql server;Ms sql server 2005;Sql server;Sql server 2005;Sql server 2008;Sql server 2008 r2;Sql server 2012;Sql;Sql queries;Stored procedures;Clustering;Backups;T-sql;Virtualization;R2;Maintenance;Problem solving;Shipping:::Database Administrator Database Administrator Database Administrator - Family Private Care LLC Lawrenceville, GA A self-motivated Production SQL Server Database Administrator who possesses† strong analytical and problem solving skills. My experience includes SQL Server† 2005, 2008 and 2012, 2014, SSIS, as well as clustering, mirroring, and high† availability solutions in OLTP environments. I am proficient in database backup,† recovery, performance tuning, maintenance tasks, security, and consolidation.† I am confident that I would make a beneficial addition to any company. Over the† course of my career thus far, I have designed databases to fit a variety of needs,† successfully ensured the security of those databases, problem-solved in order to meet† both back-end and front-end needs, installed and tested new versions database† management systems, customized and installed applications and meticulously† monitored performance for the smoothest front-end experience possible. During my† 5 to 6 years working with databases. Work Experience Database Administrator Family Private Care LLC - Roswell, GA April 2017 to Present Confirm that backups have been made and successfully saved to a secure† location† Planning for backup and recovery of database information.† Maintaining archived data Backing up and restoring databases, Contacting† database vendor for technical support Generating various reports by querying from database as per needed.† Managing and monitoring data replication. Acting as liaison with users High† Availability or Disaster Recovery Logs - Check your high availability and/or disaster recovery process logs. Depending on the solution (Log Shipping,† Clustering, Replication, Database Mirroring, CDP, etc.) that you are using† dictates what needs to be checked. Correcting errors and make necessary† modification† Modify existing databases and

Fig.2 Uncleaned Text

Dataset after preprocessing:

database administrator database administrator family private care llc lawrenceville ga selfmotivated production sql server database administrator possesses strong analytical problem solving skills experience includes sql server ssis well clustering mirroring high availability solutions oltp environments proficient database backup recovery performance tuning maintenance tasks security consolidation confident would make beneficial addition company course career thus far designed databases fit variety needs successfully ensured security databases problemsolved order meet backend frontend needs installed tested new versions database management systems customized installed applications meticulously monitored performance smoothest frontend experience possible years working databases work experience database administrator family private care llc roswell ga april present confirm backups made successfully saved secure location planning backup recovery database information maintaining archived data backing restoring databases contacting database vendor technical support generating various reports querying database per needed managing monitoring data replication acting liaison users high availability disaster recovery logs check high availability andor disaster recovery process logs depending solution log shipping clustering replication database mirroring cdp etc using dictates needs checked correcting errors make necessary modification modify existing databases database management systems direct programmers analysts make changes work part project team coordinate database development see termite project scope limitations train users answers questions approve schedule plan supervise installation testing new products improvements computer systems installation new databases review procedures database management system manuals making changes database select enter codes monitor database performance create production database check backup failure alerts correct errors rerun backups review average duration backup significant changes occurred investigates time happens due networking low bandwidth validate backup files using restore verify create jobs take care task send notification fails verify backup monitoring backup log history cleaning designed find newly added databases define backup plan verify free space drive servers significant variance free space day research cause free space fluctuation resolve necessary often times log files grow monthly jobs automate job job runs every one hour

Fig.3 Cleaned Text

Methodology

GloVe embeddings to convert our textual data into vectors, leveraging the pre-trained model from Stanford Common Crawl. However, we encountered challenges when attempting to find word neighbors using the spaCy package, as the generated vectors did not consistently yield accurate representations. Additionally, the cosine similarity scores between related documents were lower than anticipated, prompting us to explore alternative embedding methods.

Subsequently, we transitioned to skip-gram embeddings, utilizing the FastText package from Facebook. The embeddings generated through skip-gram proved to be more accurate, evidenced by the improved accuracy in identifying similar neighbors during the nearest neighbor retrieval process. Encouraged by these results, we proceeded to implement models, including RNN, LSTM, and Bidirectional models, using these skip-gram embeddings. However, the achieved accuracy across all models was suboptimal, hovering around 45%.

our exploration of skip-gram embeddings from the FastText package led to the development of the LSTM model, denoted as model_50d. This architecture featured a series of five LSTM layers, each with 128 units, strategically enhanced by dropout layers to curb overfitting.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 1499, 128)	91648
dropout_7 (Dropout)	(None, 1499, 128)	0
lstm_6 (LSTM)	(None, 1499, 128)	131584
dropout_8 (Dropout)	(None, 1499, 128)	0
lstm_7 (LSTM)	(None, 1499, 128)	131584
dropout_9 (Dropout)	(None, 1499, 128)	0
lstm_8 (LSTM)	(None, 1499, 128)	131584
dropout_10 (Dropout)	(None, 1499, 128)	0
lstm_9 (LSTM)	(None, 1499, 128)	131584
dropout_11 (Dropout)	(None, 1499, 128)	0
flatten_1 (Flatten)	(None, 191872)	0
dense_3 (Dense)	(None, 128)	24559744
dropout_12 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 32)	4128
dropout_13 (Dropout)	(None, 32)	0
dense_5 (Dense)	(None, 10)	330

Total params: 25182186 (96.06 MB)
Trainable params: 25182186 (96.06 MB)
Non-trainable params: 0 (0.00 Byte)

Fig.4 LSTM Architecture

Following the suboptimal performance of the LSTM model, indicated by a micro average F1 score of 0.40, we transitioned to a Bidirectional LSTM (BiLSTM) architecture. The embeddings for each word were increased to 150 dimensions to capture more intricate semantic information. The BiLSTM model, denoted as model_150d, embraced bidirectional recurrent layers for enhanced information capture. The architecture consisted of multiple Bidirectional LSTM layers, each followed by batch normalization and dropout layers to regulate overfitting. The bidirectional nature of the LSTM layers allowed the model to consider context from both directions, enriching its understanding of sequential dependencies. The micro average F1 score for the BiLSTM model with 150-dimensional embeddings showed improvement, reaching 0.45. While this represented progress, the quest for higher accuracy prompted further exploration.

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 1499, 256)	285696
batch_normalization (BatchNormalization)	(None, 1499, 256)	1024
dropout (Dropout)	(None, 1499, 256)	0
bidirectional_1 (Bidirectional)	(None, 1499, 512)	1050624
batch_normalization_1 (BatchNormalization)	(None, 1499, 512)	2048
dropout_1 (Dropout)	(None, 1499, 512)	0
bidirectional_2 (Bidirectional)	(None, 1499, 512)	1574912
batch_normalization_2 (BatchNormalization)	(None, 1499, 512)	2048
dropout_2 (Dropout)	(None, 1499, 512)	0
lstm_3 (LSTM)	(None, 128)	328192
dropout_3 (Dropout)	(None, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 128)	16512
batch_normalization_3 (BatchNormalization)	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 32)	4128
dropout_5 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 10)	330

Total params: 3266026 (12.46 MB)
Trainable params: 3263210 (12.45 MB)
Non-trainable params: 2816 (11.00 KB)

Fig.4 bidirectional LSTM Architecture

In response to the performance limitations observed with traditional sequence models, we decided to pivot towards transformer architectures. This shift aimed to harness the power of self-attention mechanisms and contextual embeddings, which have demonstrated superior performance in capturing complex relationships within textual data. This strategic move towards transformer models marked a crucial evolution in our approach, with the expectation of achieving enhanced accuracy in the task of classifying resumes and measuring similarity between documents.

In the pursuit of refining our approach further, we shifted to leveraging pre-trained BERT embeddings. The BERT model was adopted from the 'distilbert-base-uncased' variant, utilizing the Hugging Face Transformers library. The custom model architecture was designed to optimize the classification task based on our specific requirements. The input consisted of two layers for input_ids and attention masks. The BERT embeddings were obtained from the pre-

trained model, with the added flexibility of fine-tuning only specific layers for our task. The pooled output layer from BERT was followed by a densely connected layer with 10 units and a sigmoid activation function, enabling multi-label classification.

- Input Layer (input_ids): Accepts the tokenized input sequence.
- Input Layer (attention_mask): Manages attention masks to highlight relevant tokens.
- BERT Embeddings: Utilizes the pre-trained BERT model with a frozen architecture.
- Dense Layer (10 units, sigmoid activation): Facilitates multi-label classification.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_ids (InputLayer)      [(None, 512)]             0         
                                                                 
 attention_mask (InputLayer  [(None, 512)]             0         
 )                                                               
                                                                 
 bert (TFBertMainLayer)      TFBaseModelOutputWi       109482240 
                             thPoolingAndCrossAt                 
                             tentions(last_hidde                 
                             n_state=(None, 512,                 
                              768),                              
                              pooler_output=(Non                 
                             e, 768),                            
                              past_key_values=No                 
                             ne, hidden_states=N                 
                             one, attentions=Non                 
                             e, cross_attentions                 
                             =None)                              
                                                                 
 output_layer (Dense)        (None, 10)                7690      
                                                                 
=================================================================
Total params: 109,489,930
Trainable params: 7,690
Non-trainable params: 109,482,240
_________________________________________________________________
```

Fig.6 Pretrained Bert Architecture

The initial micro average F1 score of approximately 51% indicated room for improvement, prompting the development of a custom model tailored to the unique intricacies of our task.

In response to the limitations observed with pretrained transformers, we embarked on the design and implementation of a custom transformer architecture. This decision stemmed from the recognition that fine-tuning existing models might not fully capture the nuances of our unique dataset. The objective was to create a transformer architecture specifically tailored to the intricacies of resume data, optimizing its ability to discern relevant patterns and relationships.

Our custom transformer model, named Multilabel Transformer model, exhibits a carefully crafted architecture to maximize its performance in resume classification. The model consists of several key components

- **Embedding Layer:** The initial layer of the model involves an embedding layer responsible for mapping words to vectors. This layer facilitates the conversion of input data into a format suitable for subsequent processing.
- **Multi-Head Self Attention Layer:** A crucial aspect of the architecture is the incorporation of multiple self-attention heads. This design choice allows the model to focus on different aspects of the input simultaneously, enhancing its ability to capture diverse relationships within the resume text.
- **Feedforward Layer:** Following the attention mechanism, a feedforward layer introduces non-linear transformations to the data, promoting the extraction of intricate features and patterns.
- **Global Average Pooling Layer:** To consolidate information and reduce dimensionality, a global average pooling layer is employed. This step facilitates the extraction of essential features from the transformed data.
- **Dense Layers for Classification:** The final layers of the model include dense layers tailored for resume classification. These layers are responsible for making predictions based on the learned features, providing a binary classification output for each job category.

```python
class MultiLabelTransformerModel(tf.keras.Model):
    def __init__(self, vocab_size, num_labels, d_model=256, num_heads=4, ff_dim=4, dropout=0.2, max_len=512):
        super(MultiLabelTransformerModel, self).__init__()

        # Embedding layer
        self.embedding = layers.Embedding(vocab_size, d_model, input_length=max_len)

        # Multi-Head Self Attention layer
        # self.attention = layers.MultiHeadAttention(num_heads=num_heads, key_dim=d_model // num_heads)
        self.attention_heads = [layers.MultiHeadAttention(num_heads=num_heads, key_dim=d_model // num_heads) for _ in range(3)]
        # loop is for adding more MultiHeadAttention layers

        # Feedforward layer
        self.ffn = models.Sequential([
            layers.Dense(ff_dim, activation='relu'),
            layers.Dropout(dropout),
            layers.Dense(d_model)
        ])

        # Global average pooling layer
        self.global_average_pooling = layers.GlobalAveragePooling1D()

        # Dense layers for classification
        self.dense1 = layers.Dense(512, activation='relu')
        self.dropout = layers.Dropout(dropout)
        self.dense2 = layers.Dense(num_labels, activation='sigmoid')  # Binary classification

    def call(self, inputs):
        x = self.embedding(inputs)

        # Multi-Head Self Attention
        attention_outputs = [attention_head(x, x) for attention_head in self.attention_heads]
        x = layers.Add()(attention_outputs)

        # Feedforward layer
        x = self.ffn(x)

        x = self.global_average_pooling(x)
        x = self.dense1(x)
        x = self.dropout(x)
        output = self.dense2(x)
        return output
```

Fig.7 Custom Transformer Architecture

After classifying the resumes we wanted to develop a method for predicting job compatibility by leveraging BERT embeddings and cosine similarity after classifying resumes into categories. To achieve this, we gathered a dataset of resumes from a designated file (resume_samples.txt). Each entry in the file follows the format ***reference_id:::occupations:::resume_text*,** allowing us to extract crucial information like reference ID, occupations, and the raw resume text.

The data preprocessing phase was vital to ensure the effectiveness of our model. We employed a cleaning

function (clean_txt) that includes essential steps such as lowercasing, removal of hyperlinks, email addresses, and special characters, along with lemmatization to enhance the accuracy of our predictions. The cleaned data was then organized into a structured DataFrame, incorporating columns for reference ID, occupations, and cleaned resume text.

Our methodology involves utilizing the BERT (Bidirectional Encoder Representations from Transformers) model and tokenizer from the Hugging Face Transformers library. A specific custom function (get_bert_embeddings_for_batch) tokenizes and converts batches of texts into BERT embeddings using the pre-trained bert-base-uncased model. Simultaneously, the job description undergoes the same cleaning process, and BERT embeddings are obtained to represent its semantic content.

```
def get_bert_embeddings(text):
    # Tokenize and convert to tensor
    input_ids = tokenizer.encode(text, return_tensors='pt', max_length=512, truncation=True)

    # Get BERT embeddings
    with torch.no_grad():
        outputs = model(input_ids)
        embeddings = outputs.pooler_output.numpy()

    return embeddings

def get_bert_embeddings_for_batch(texts):
    # Tokenize and converted to tensor for a batch of texts
    tokens = tokenizer(texts.tolist(), return_tensors='pt', max_length=512, truncation=True,
    input_ids = tokens['input_ids']

    # Get BERT embeddings for the batch
    with torch.no_grad():
        outputs = model(input_ids)
        embeddings = outputs.pooler_output.numpy()

    return embeddings
```

Fig.8 get_bert_embeddings

To handle memory constraints effectively, we implemented batch processing for the resumes. BERT embeddings for each resume text are calculated and stored in a new column ('embedding_resume') in the DataFrame. Subsequently, cosine similarity, a measure of similarity between vectors, is computed between the BERT embeddings of each resume and the job description. The resulting similarity scores are then stored in another column ('cosine_similarity').

In the final stages, we identify the resume with the highest cosine similarity to the job description using the idxmax function. The results, including the processed job description, details of the most similar resume (occupations, cleaned resume text), and the cosine similarity score, are then displayed. This methodology provides a comprehensive approach to predicting job compatibility and serves as a foundation for further exploration and refinement of the model.

Another alternative we explored was the FastText . It provides an efficient method for generating word embeddings and capturing semantic relationships within texts. Similar to the BERT approach, we implemented a preprocessing step to clean the data and ensure consistency. The FastText generated embeddings were used to calculate similarity scores between the resumes and the job description. The process involves encoding each word in the text into vectors and aggregating them to represent the entire text

```
def get_word_embeddings(text, model):
    embeddings = [model.get_word_vector(word) for word in text.split()]
    # If there are no word vectors, returns an empty array
    if not embeddings:
        return np.zeros((1, model.get_dimension()))
    # Return the average of all word vectors
    return np.mean(embeddings, axis=0).reshape(1, -1)
```

Fig.9 get_bert_embeddings with text and model parameters

By incorporating both BERT and FastText methodologies, we aim to evaluate the effectiveness of different embedding techniques in predicting job compatibility. The comparative analysis of these approaches provides a more comprehensive understanding of the strengths and limitations of each method, contributing to the overall robustness of our predictive model.

Result

1. Classification result using LSTM with 150 Dimensional Vector

	precision	recall	f1-score	support
0	0.42	0.55	0.48	608
1	0.84	0.80	0.82	2454
2	0.00	0.00	0.00	1213
3	0.00	0.00	0.00	873
4	0.00	0.00	0.00	560
5	0.23	1.00	0.37	1128
6	0.00	0.00	0.00	884
7	0.00	0.00	0.00	707
8	0.00	0.00	0.00	364
9	0.00	0.00	0.00	387
micro avg	0.43	0.37	0.40	9178
macro avg	0.15	0.24	0.17	9178
weighted avg	0.28	0.37	0.30	9178
samples avg	0.34	0.34	0.32	9178

Fig.10 LSTM Model

2. Classification result using Bidirectional LSTM with 150-Dimensional Vector

	precision	recall	f1-score	support
0	0.24	0.72	0.36	608
1	1.00	0.48	0.64	2454
2	0.43	0.94	0.59	1213
3	0.54	0.29	0.38	873
4	0.94	0.29	0.44	560
5	0.00	0.00	0.00	1128
6	0.40	0.92	0.55	884
7	0.00	0.00	0.00	707
8	0.00	0.00	0.00	364
9	0.00	0.00	0.00	387
micro avg	0.48	0.43	0.45	9178
macro avg	0.35	0.36	0.30	9178
weighted avg	0.49	0.43	0.39	9178
samples avg	0.49	0.49	0.45	9178

Fig.11 Bidirectional LSTM

3. jobFitNet Custom Model - 3 MultiHead Attention

	precision	recall	f1-score	support
0	0.91	0.79	0.85	608
1	0.95	0.94	0.94	2454
2	0.80	0.71	0.75	1213
3	0.73	0.78	0.76	873
4	0.80	0.78	0.79	560
5	0.76	0.68	0.71	1128
6	0.74	0.80	0.77	884
7	0.86	0.83	0.85	707
8	0.77	0.78	0.77	364
9	0.89	0.82	0.85	387
micro avg	0.83	0.81	0.82	9178
macro avg	0.82	0.79	0.80	9178
weighted avg	0.84	0.81	0.82	9178
samples avg	0.86	0.85	0.83	9178

Fig.12 jobFitNet Model

4. Cosine Similarity Model Result

	Data Science Job Description	UI/UX Job Description
Data Science Resume	0.9666395	0.8452022
UI/UX Resume	0.8252022	0.96888655

Conclusion:

In conclusion, the comparative analysis of the four models utilized in this project we found that the custom model featuring 2 MultiHead Attention layers emerged as the most effective in predicting job classification. This conclusion is drawn from the evaluation metrics, which indicate an average accuracy rate of 81% for the classification tasks as we saw above in the results. The inclusion of 2 MultiHead Attention layers in the custom model evidently contributes to its superior performance, showcasing the model's robustness and adaptability in capturing intricate relationships within the dataset. We conclude that tailoring the model to specific characteristics of the dataset, ultimately leads to enhanced predictive capabilities, and it is clear from Fig12 that the custom model jobFitNet with 3 MultiHead Attention has outperformed the Bidirectional LSTM.

Future Scope

Fine-tuning and Hyperparameter Tuning
Further fine-tune the existing jobFitNet models and experiment with different hyperparameter configurations to achieve better performance.

Data Augmentation
Explore data augmentation techniques to artificially increase the size of our training dataset.

Ensemble Learning:
We would explore combining the strengths of LSTM-based models, transformer-based models, and other architectures to improve overall accuracy and robustness.

Active Learning:
Active learning strategies can intelligently select and label the most informative samples for model training. This method is used with limited labelled data.

Domain-Specific Embeddings:
We would like to train embeddings specific to the domain of resumes and job descriptions. Domain-specific embeddings can capture more nuanced relationships and improve model performance.

User Feedback Integration:
We would like to have user feedback where users can tell if the job was a good suggestion with respect to their resume. A recruiter can also tell if the suggested resumes were a good fit. This feedback can be used to continuously improve and refine the model.

References:

1. Data Source : resume-dataset and Resume_corpus
2. Research Paper 1 : Understanding Deep Learning Performance through an Examination of Test Set Difficulty: A Psychometric Case Study
3. Research Paper 2 : BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

4. Appendix :

   Code Repository Link : GitHub link