Person-Job Fit: Classification and Ranking Model

Amina Mohsin Sheikh (20987635) and **Mohammed Ibrahim Shariff (21041214)**
University of Waterloo

Abstract

Recruiters are always on the hunt to find the most suitable candidates to fill vacant positions. This often requires manually going through resumes to browse for the specific requirements laid out in the job description. This task is further made tedious/impossible when dealing with thousands of resumes for a single job posting. The tremendous increase in Applicant Tracking Systems (ATS) designed to facilitate the process serves as validation of this.

Our model, albeit similar to the ATS system is unique in terms of the architecture it uses. The final goal of it being to rank the top n resumes for a given job description based on the job title, experience, and skills of a potential candidate.

To tackle the problem, we have first extracted the job title as a label and skills as an added feature. The model itself operates two-fold: the first segment focuses on classification i.e., using a feed forward neural network architecture to predict the label i.e., job title based on the resume content. The second segment focuses on the resume and job description matching algorithm wherein we have used different BERT models to first create the sentence embeddings for both the resume content and job description and then on the basis of the label matched the resume to the job description to return n resumes with the highest cosine similarity score.

KEYWORDS: regex, feed-forward neural network, BERT models, cosine similarity

Github Repository:

https://github.com/amsheikh-spec/Resume-Job-Description-Matching/tree/main

1 Background/Related Work

A great deal of work has been done to shortlist and match the most suitable candidates for a particular job posting. With the rapid progress being made in the field of Natural Language Processing, there exists a vast variety of techniques that can be employed to solve the problem. This can be witnessed in the numerous publications which exist on the topic and range from as simple as using a count vectorizer to utilising deep learning based methods. All with the objective of finding the most suitable candidate for a job.

A very simple model is adopted by Randerson, where he first imports and compiles the resume and job description in a single variable and runs a count vectorizer to identify the matching elements. Cosine similarity is then used to calculate the match score (randerson112358, 2020). Shreya, in contrast uses a slightly more advanced technique where she uses AI techniques to build a resume screener; before this can be done, the documents need to be pre-processed via tokenization and stop word removal. Word embeddings are then created using a word2vec model and cosine similarity is used to match the resumes to the job description. This is followed by extraction of skills from the shortlisted resumes to return the matched content (Shreya, 2023). Nivetha employs a similar approach as Shreya, with the difference of using doc2vec to train the model in place of the conventional word2vec (B, 2022).

However, when dealing with a large pile of resumes, it helps to classify the documents so that similar resumes get grouped together. This can be done using the 'job title' as a label. Kajal explores resume screening techniques and uses one-vs-rest K-Nearest Neighbor classifier to first train the model to identify the correct label and make predictions on the test set (Kumari, 2021). Similarly, Lin et al. first extract key features from the resume such as age, gender, degree, work experience, etc. They then used a word2vec model to capture semantic similarity and used k-means clustering to group and classify the different industry resumes. Accuracy of predicting the class is gauged using various 'shallow' and 'deep' estimators (Lin et al., 2016).

In contrast, to these models, while our model

is similar in terms of the approach and end goal, the technique altogether is very different from first building a neural network to predict class labels and then using different pre-trained BERT models to create embeddings which are used to calculate the cosine similarity. Jun has built a similar ranking model but their paper is more focused towards comparing the performance of BERT models to that of SBERT models (Jun, 2021). We thus hope to employ a mix of the aforementioned techniques and assess how well they perform in context to our data.

2 Approach

2.1 Regex

The resumes that we are working with have data stored in a consistent format and can be divided into 3 chunks. The first chunk covers the experience of the employee (not labeled), the second chunk covers 'Education' (labeled) and the last chunk has the 'Skills' (labeled) section.

Of the two labeled sections, we focused on extracting 'Skills' as it is a better indicator of a candidate's compatibility for a job. 'Education' for most candidates contained only the name of the college they attended and lacked information on the level of qualification i.e., Bachelors, Masters, PhD, etc making it unsuitable for comparison purposes.

To store 'Skills' as a separate feature we used regex to extract all the information stored under the respective header. The main purpose for extracting 'Skills' is so that if need arises i.e., resume-job description matching doesn't produce high match scores, then we can match skills to the job description to generate the top n candidates.

This model could serve as an alternative to comparing the entire content of the resumes to that of the job description which can be a time-consuming task especially given a large dataset.

2.2 Neural Networks

For our classification task i.e., predicting the job title from the resume, we have employed a feed forward neural network. We first used a label encoder to convert our unique labels i.e., job titles to unique numbers ranging from 0 to 6 [0 = database administrator, 1 = systems administrator, 2 = project manager, 3 = software developer, 4 = web developer, 5 = network administrator, 6 = java developer]. The model was then constructed comprising an input layer of word embeddings, 5 hidden layers with ReLU activation and an output layer with softmax activation.

We first divided the data set into a 70:30 training-test split and created embeddings for both data sets using DistilBERT tokenizer. This model has the advantage of being smaller, lighter and faster all the while preserving 95 percent of BERT model efficiency (Face, b). For the tokenizer, a max length of 1050 tokens has been defined (close to seventy-fifth percentile of the resume data set) so as to capture semantic similarity as well as achieve computational efficiency.

In the hidden layers, we have used ReLU as the activation function instead of sigmoid as it is has a lower run time and avoids the problem of vanishing gradient. For best efficiency, the dense parameter is set separately for each of the hidden dimensions. Batch normalization layers are also included to improve the training efficiency and stability. The dropout rate is set to 0.25 and serves as the optimal value to prevent over fitting and improves generalization by randomly dropping neurons.

For training our model, we started with 2 epochs. General, rule of thumb is to use 3 times the number of columns of the data set (Gretel) which in case of our model would be 6. This gives us a high accuracy on our training, validation and test set ensuring that the data does not over fit. In addition, the model uses an early stopping technique on validation data so to continue running the epochs only as long as there is an improvement in the validation data accuracy.

The output layer uses softmax activation to calculate the probability of the outcome with the units set to 7 i.e., unique labels in the data. Moreover, the model uses the cross entropy loss function and adams optimizer. Predictions are made on the test set to report the accuracy as well as the loss. The architecture of the neural network is summarized in Figure 1.

2.3 Fine-Tuning BERT

Fine-tuning BERT (Bidirectional Encoder Representations from Transformers) on our dataset involved using a pre-trained BERT model and adapting it to our dataset. BERT is a powerful language model that has been pre-trained on a large corpus of text, and fine-tuning allows it to learn task-specific features.

First, we installed necessary libraries, including TensorFlow and the Hugging Face Transformers

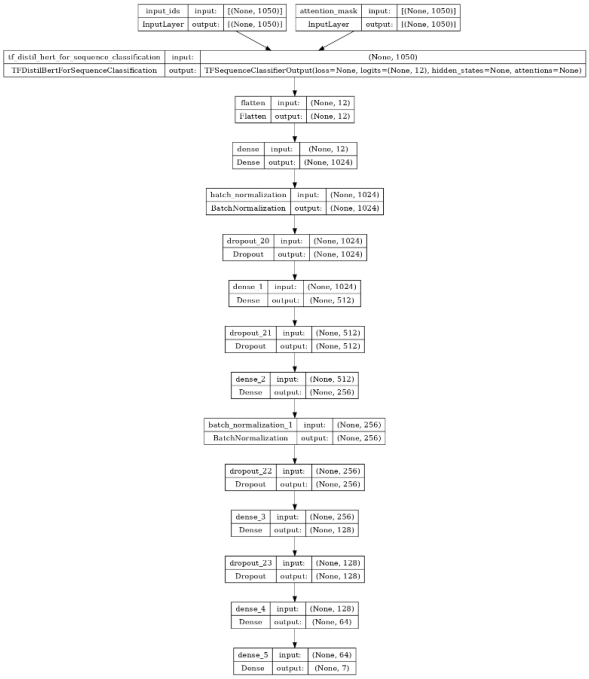

Figure 1: Neural Network Architecture

library, which provides easy access to pre-trained BERT models. Then we prepared our dataset in a format suitable for BERT. We used the BERT tokenizer provided by the Transformers library for this purpose.

We loaded 'manishiitg/distilbert-resume-parts-classify' model from the transformers library for our neural network. We then modified the model architecture accordingly by adding task-specific layers on top of the pre-trained BERT model.

The modified BERT model was then trained on our dataset. During fine-tuning, we updated the model's parameters. We also adjusted the learning rate, batch size, and the number of training epochs based on our dataset's characteristics which are discussed in section *3.3.1*. After training, we evaluated the fine-tuned BERT model on the test dataset to assess its performance. We use metrics such as accuracy, f1-score, precision, and recall.

2.4 BERT Sentence Transformers

BERT sentence transformers were introduced by google in 2018 to allow computers to understand the semantic meaning of documents by obtaining context from the surrounding text. These models are pretrained using a mix of masked language modeling (mlm) and next sentence prediction on a large dataset (Face, a). A major objective of our model is to assess the effectiveness of BERT transformers with our data for encoding and similarity comparison.

We used the models to create embeddings for the resumes and job descriptions to match the documents and compute the cosine similarity score. For our baseline, we used a single transformer i.e., 'all-MiniLM-L6-v2'. In this document we have focused on comparing different BERT models and refined the process.

To assess the model most suitable for the length of our dataset, we checked the distribution of the word count of each resume to find that the seventy-fifth percentile is approximately 900 tokens with the largest resume having around 10,000 tokens. To create optimal embeddings, we initially considered large BERT transformers such as 'bigbird-roberta-large' and 'xlm-mlm-en-2048' which can process up to 4096 and 2048 tokens respectively (Face, c). However, owing to the large size of these models and the large size of our dataset, we were unable to process the resumes owing to limited CPU/GPU processing power on our local machines.

We thus directed our focus towards smaller BERT transformers capable of processing 256 tokens at a time. Given the large number of tokens in the resumes, we used the sliding window algorithm to process 250 tokens at a time with an overlap of 50 tokens. We created 5 iterations to process up to 1000 tokens all the while preserving semantic context through overlap. For resumes with more than 1000 tokens, embeddings were created for the last 1000 words of the resume as it contains key components such as education, skills and segment of the experience section. It is also the most widely adopted approach for creating embeddings when faced with such limitations (stackoverflow, 2021).

The three BERT transformers that were used for this task are as follows: (i) all-MiniLM-L6-v2 (ii) all-MiniLM-L12-v2 and (iii) paraphrase-albert-small-v2. While all three models share common features such as sentence similarity prediction, compatibility with cosine similarity and training on a large corpus. (i) and (ii) have the advantage of being fine tuned on a 1 billion sentence-pair dataset with (i) also being 5 times faster and producing higher quality (SBERT.net). We have compared the effectiveness of each of these and summarized the effectiveness in the results section.

Statistic	Value
count	23696
mean	719.96
std	627.29
min	5
25%	299
50%	503
75%	910.25
max	9896

Figure 2: Resume Word Count

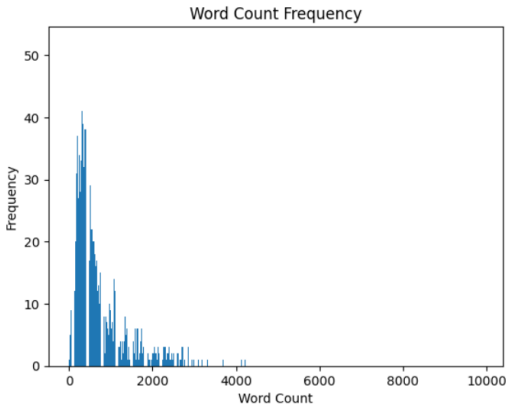

Figure 3: Resume Word Count Distribution

3 Experiments

3.1 Data

For this project, we are using two datasets: (i) Resume Corpus - this is a vast dataset comprising 30k resumes, stored independently as text files alongside their labels i.e., 'job title'. We first matched the resume to its appropriate label based on a unique identifier and stored the information in 'resume-data.csv'. (ii) Job Description - 'jobdescription.csv' is a large file containing key details such as job description, job title, location, etc for over 15k job postings having hundreds of unique job titles. We dropped all columns except for job title and job description as the content of the latter encompassed all necessary details.

3.2 Data Preprocessing

We preprocessed both CSV files to convert labels i.e., 'job title' to lowercase, and also removed special characters/punctuation and duplicates. Similarly, we preprocessed the resume and job description to convert all text to lower case, removed stopwords and punctuation (B, 2022). We then filtered the resume dataset to only keep resumes for job titles that have a count > 20. For job descriptions, we then filtered the job title to only keep those that match the job title in the resume file. This left us with 7 unique job titles in both CSV files with approximately 24k data points for resumes and 500 data points for job descriptions. The corresponding preprocessed and filtered files were then stored in 'preprocessedresume.csv' and 'preprocessedjd.csv' respectively.

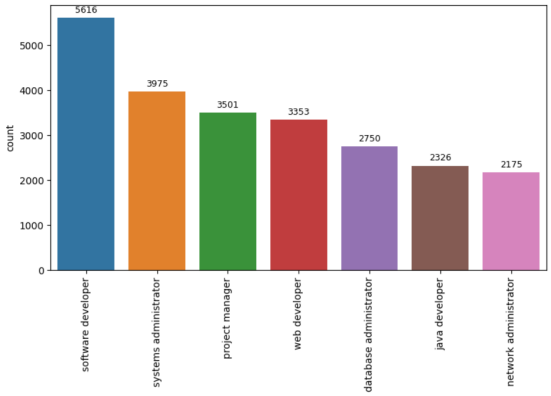

Figure 4: Job Title Count for Resumes

3.3 Experiments

3.3.1 Neural Network

For our Neural Network, we experimented with the hyperparameter's used to train the model. The model has an input layer of word embeddings, hidden layers with ReLU activation and output layer with softmax activation. We varied the number of hidden layers starting with 2 and eventually stuck to 5 as this generated the most optimal results. For each of these layers, we kept a separate dense parameter ranging from 1024 to 64. Batch normalization layers were also included to improve training efficiency.

We initially started with a dropout rate of 0.5 for the hidden layers but eventually had to reduce it as this caused too much of the information to be lost leading to underfitting. The final model thus has a dropout rate of 0.25 for each of the hidden layers. In addition, we started with 2 epochs and increased them up to 6 as previously mentioned that a good rule of thumb is to take three times the number of columns in the dataset. Early stopping techniques were employed to prevent overfitting. The impact of these is explained in the results section.

Moreover, batch size during training was set to 8; since the model performed well with this config-

uration, we did not change it. Lastly, crossentropy loss function was used alongside Adam optimizer with an initial configuration of 0.00005 and uses a decay rate of 0.01 after 10,000 decay steps to allow the optimization process to fine-tune the model more efficiently. The final hyperparameters used in the Neural Network are summarized in Figure 5.

Hyperparameter	Value
Hidden Layer Activation	ReLU
No. of Hidden Layers	5
Dense Parameters	104, 512, 256, 128, 64
Dropout Rate	0.25
No. of Epochs	6
Batch Size	8
Adam Optimizer	0.00005

Figure 5: Neural Network Configurations

3.3.2 BERT Model

As mentioned earlier, given the long length of our resumes, we initially considered large pretrained models i.e., 'bigbird-roberta-195 large' and 'xlm-mlm-en-2048' which can process up to 4096 and 2048 tokens respectively.However, owing to CPU/GPU limitations, we were unable to proceed with these transformers and resorted to smaller models capable of processing upto 256 tokens. The model was then useed to create embeddings for both the resume and job description using the sliding window approach.

3.4 Evaluation Metrics

3.4.1 Classification Model

To assess how well our neural network performed, the key metric we are interested in is **accuracy** of the test set i.e., the number of correct predictions made. In addition, we have also looked at **precision** calculated as the number of true positive predictions divided by the total positive predictions. This serves an indicator of how accurate the predictions are.

Recall is also considered to gauge how effective the model is at correctly identifying positive samples. **f1-score** is included to balance precision + recall and gives higher weight to predictions that perform well on both.

Lastly, **confusion matrix** is incorporated to summarize the errors the model is making. As it is not only important to look at the accuracy of predictions but also the errors the model makes.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

FN = False negative

3.4.2 Ranking Model

The metric of choice here is **cosine similarity**, it is the most commonly used measure when assessing how well two documents match. It calculates the cosine of the angle between two vectors,the closer the result is to 1, the stronger the similarity between the 2 vectors i.e., in our case the resume and job description.

3.5 Results

3.5.1 Classification Model

In terms of accuracy, our model performed well on both train and test set producing an accuracy score of 0.9225 and 0.9248 respectively. The accuracy can be seen to increase with the number of epochs in Figure 6. Similarly, in Figure 7, the loss can be seen to decrease with the number of epochs. The impact of the early stopping technique can be witnessed in both these graphs via prevention of over fitting. Since, we are getting high accuracy with 6 epochs, any epochs above this would probably not be included as it would lead to higher accuracy at the risk of over-fitting.

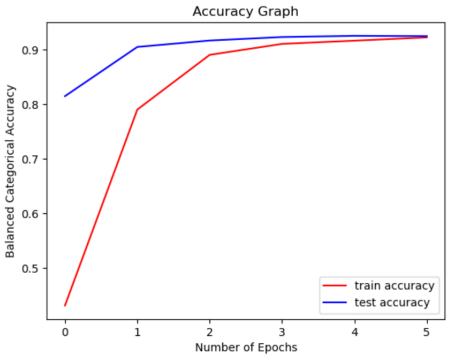

Figure 6: Classification Accuracy

The classification report in Figure 8 summarizes how well the model performed in terms of precision, recall and f1-score. Looking at the table it can be observed that prediction for label 0 i.e., database administrator had the highest precision i.e., 0.98 and a high recall of 0.95. Consequently, it can be observed that it has the highest f1-score. Moreover, since the score for all labels on all metrics is 0.9

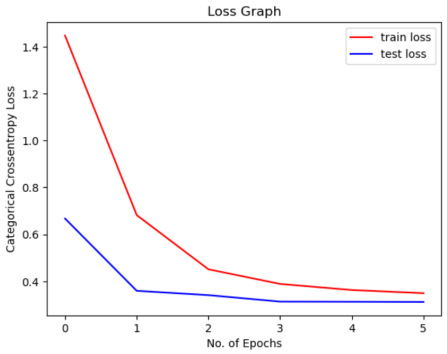

Figure 7: Classification Loss

or above, we have good reason to believe that the model made accurate predictions on the test set.

	precision	recall	f1-score	support
0	0.98	0.95	0.96	868
1	0.92	0.95	0.93	1226
2	0.94	0.90	0.92	1044
3	0.92	0.91	0.92	1714
4	0.88	0.92	0.90	1046
5	0.91	0.91	0.91	673
6	0.94	0.94	0.94	759
accuracy			0.92	7330
macro avg	0.93	0.93	0.93	7330
weighted avg	0.93	0.92	0.92	7330

Figure 8: Classification Report

Figure 9 provides a summary of the1 confusion matrix. With the actual label on the vertical axis and predicted label on the horizontal axis, each row denotes the errors i.e., number of times the actual label was *i* and the model predicted *j*. Similarly, all the entries in the diagonal denote the number of times the model made correct predictions i.e., actual label = predicted label. Looking at the matrix, it can be observed that model primarily made correct predictions.

```
Confusion Matrix:
[[ 824   13   10    9    7    2    3]
 [   3 1165   16    8    5   28    1]
 [   7   37  943   18   11   28    0]
 [   7    6   13 1558   95    1   34]
 [   0    3    4   65  967    1    6]
 [   1   44    9    4    3  612    0]
 [   0    2    3   28   16    0  710]]
```

Figure 9: Confusion Matrix

3.5.2 Ranking Model

To gauge how well the model performs, a job title was chosen at random i.e., Web Developer. Then using a job description corresponding to the selected label, the resumes were first matched based on the job title and then the **top 5** resumes were shortlisted from the total 3,353 candidates for the position.

Resume-Job Description Match: The performance of the different BERT models is summarized below. Each result contains the top 5 resumes along with the cosine similarity score of how well each resume matches the job description.

(i) all-MiniLM-L12-v2

	Resume	Score
0	job seeker work experience turkish airlines ma...	tensor([93.4925])
1	manager manager manager central board shop wau...	tensor([93.4525])
2	web designer developercustomer service web des...	tensor([93.4005])
3	web designer developer web designerdeveloper m...	tensor([93.2931])
4	sales associate senior web developer chief ope...	tensor([93.2503])

(ii) all-MiniLM-L12-v2

	Resume	Score
0	job seeker work experience turkish airlines ma...	tensor([92.5159])
1	developer developer web developer prepress dig...	tensor([91.6480])
2	developer developer web developer prepress dig...	tensor([91.6362])
3	professor ms sql server intro course professor...	tensor([91.5713])
4	pc technician pc technician web developer cost...	tensor([91.2708])

(iii) paraphrase-albert-small-v2

	Resume	Score
0	web designer developer web designerdeveloper o...	tensor([93.0785])
1	web developer web developer web developer dund...	tensor([93.0361])
2	support technician support technician support ...	tensor([92.9733])
3	web developer web developer full stack web dev...	tensor([92.8317])
4	web developer web developer bellevue wa work e...	tensor([92.7874])

Looking at the results, it can be observed that the top 5 resumes shortlisted by each of these 3 models have a cosine similarity score over 90. While model (i) performed best and yielded the highest match-score, model (iii) comes close followed by model (ii).

There also appears to be some overlap in terms of the resumes shortlisted by each of these models: models (i) and (ii) have ranked the same resume for the first position, albeit both have different match scores. In contrast, the number one resume selected model (iii) has the fourth rank under model (i).

The difference in results can be attributed to the architecture of the BERT models which varies in terms of pre-training data, embedding quality as well as the degree of fine tuning required.

4 Conclusion

In this study, we developed a classification and ranking model to address the challenge of matching resumes to job descriptions effectively. Our approach combines the power of feed-forward neural networks for classification and BERT-based embeddings for ranking. The main contributions of our work lie in the unique combination of techniques

and the evaluation of different BERT models for creating embeddings.

From our experiments, we observed that the feed-forward neural network achieved promising results in predicting job titles from resume content. The accuracy, precision, recall, and f1-score were all consistently high across different job titles, indicating the model's effectiveness in classifying resumes.

For the ranking aspect, we experimented with various BERT models to create embeddings for resumes and job descriptions. The "all-MiniLM-L6-v2" model emerged as the top performer in terms of cosine similarity-based matching, closely followed by the "paraphrase-albert-small-v2" model.

Key takeaways are that (i) **model architecture** matters i.e., the selection of a suitable neural network and BERT model significantly impacts the overall performance of the system. (ii) **hyperparamter tuning** is important; we were only able to obtain the desired results by experimenting with various hyperparameters, such as the number of hidden layers, dropout rates, batch sizes, and learning rates. Lastly, (iii) **BERT comparison**: the choice of BERT model greatly influences the quality of embeddings and subsequently the ranking results.

Future work that can be done is (i) Assess how the model works with a **different dataset** of resumes - currently, the model at hand is tailored to the format of the resumes we have. (ii) Can employ preliminary techniques to first manually extract the **job title** of the resume and assess how well the predicted title matches the actual job title. (iii) Use **text summarising** methods to first summarise the resume before proceeding with the classification and ranking model. (iv) Build multiple ranking models for **feature-job description matching**; can use this to assess whether to use the whole resume or just the features.Lastly, (v) build **ensemble models** that aggregate predictions from multiple classifiers and could help mitigate biases,

In conclusion, our study sheds light on the effectiveness of combining classification and ranking models to address the challenge of matching resumes to job descriptions. As the field of NLP and machine learning continues to evolve, there are numerous opportunities to enhance and extend this work further.

5 Acknowledgment

We express our sincere appreciation to Professor Olga Vechtomova for her invaluable guidance and unwavering support during this course. Additionally, we are thankful to our Teaching Assistant, Gaurav Sahu, whose continuous help and prompt responses to our inquiries have been of immense benefit.

References

Nivetha B. 2022. Build accurate job resume matching algorithm using doc2vec. Accessed: 2023-08-05.

Hugging Face. a. Bert - hugging face. Accessed: 2023-08-05.

Hugging Face. b. Distilbert. Accessed: 2023-08-05.

Hugging Face. c. Pretrained models. Accessed: 2023-08-05.

Gretel. How many epochs should i train my model with? Accessed: 2023-08-05.

Lum Yao Jun. 2021. Document matching for job descriptions. page 6.

Kajal Kumari. 2021. Resume screening with natural language processing in python. Accessed: 2023-08-05.

Yiou Lin, Hang Lei, Prince Addo, and Xiaoyu Li. 2016. Machine learned resume-job matching solution.

randerson112358. 2020. Match your resume to a job description using python. Accessed: 2023-08-05.

SBERT.net. Sentence transformers - pretrained models. Accessed: 2023-08-05.

Ganeshi Shreya. 2023. Nlp case study: Build your own skill matching algorithm. Accessed: 2023-08-05.

stackoverflow. 2021. How to use bert for long text classification? Accessed: 2023-08-05.