JOB SEARCH ASSISTANT

Submitted by:
H Anarghya (1BM17IS028)
Lalitashree R Hegde
(1BM17IS039)
Pannaga Sharma ML
(1BM17IS053)
Prateeka C Bhat (1BM17IS058)

Work done as part of the Alternative Assessment for the V Semester Undergraduate (UG) Course Artificial Intelligence (16IS5DEAIN) during the Academic Year 2019-20.

Course Coordinator
Prof. Gururaja H.S.
Assistant Professor, Dept. of ISE

Department of Information Science & Engineering

B.M.S. College of Engineering
Bangalore-560019

Abstract

The way people search for jobs has changed drastically in the 21st century. The days when people used to visit companies carrying their documents and resumes in search of jobs are long gone. With the advent of internet, people now tend to apply for jobs online and go through the interview process after their resumes get shortlisted. In spite of enough job opportunities, people find it hard to look for their dream jobs. This is mainly due to the fact that many job roles are often overlapping and hence it is difficult for a fresher or a professional to figure out which job roles would be preferable for him/her. These problems of lack of proper guidance with regard to job search can be overcome by advancements in technology like Artificial intelligence, Machine Learning and Natural Language Processing etc.

In today's internet era, most of the people virtually live online. This involves conversing with virtual assistants popularly known as chatbots. These chatbots are conversational agents which interact with the users and produce responses based on the user input. For example, Google Assistant, Amazon's Alexa and Apple's Siri take user input in the form of text and speech. Chatbots which act as personal virtual assistants can be leveraged to allow users to get personalized recommendations and responses depending on their preferences.

A chatbot alone is not intelligent. The chatbot must have the ability to learn to become an intelligent chatbot. This is done using neural networks and deep learning. AI chatbot is trained to learn from data and hence respond to user requests in the most appropriate way possible. "Jobot" is one such chatbot. It is a job search assistant that recommends suitable IT jobs in different companies located in Bangalore. The recommendation is based on the user input such as

- qualification,
- level of experience
- technical skills etc.

The intelligent agent follows the PAGE model where

P - Percept

A - Actions

G - Goal

E -Environment.

Introduction

With so many different IT related jobs in demand nowadays, it is difficult for a job aspirant, be it a fresh graduate or a professional with years of experience to determine where he/she fits best in the industry. This problem is most apparent in today's data-driven economy where the job descriptions and roles are closely related and often overlapping. Before applying for jobs, the candidates have to do a lot of research as to what the company is looking for. This typically involves visiting the company's website, searching the job roles they offer, understanding their requirements etc. One company's goal and agenda differ from other companies in the corporate world. The candidates must have to perform the tedious work of knowing about hundreds of companies before they apply for the jobs. If there's a way to recommend only some jobs suitable for the candidate based on their choices, the applicants need only apply for those specific job roles available at XYZ companies. This project aims at providing the solution to the above mentioned problems existing in the current system using AI chatbot.

Jobot is a job search assistant which recommends jobs at companies based on the users' preferences. It is an AI chatbot which facilitates the user with queries and assist them with job hunt. It aims to provide a fast and convenient way to allow users to get personalized recommendations related to job roles at different companies. The chatbot will be implemented using the Flask framework of Python and Keras. Flask is a web framework that is used to build a Graphical User Interface (GUI) for the chatbot. Keras is a high-level neural networks API used to train the chatbot using deep learning.

System Requirement

The software and hardware requirements necessary to implement the chatbot are stated below.

Software Requirements

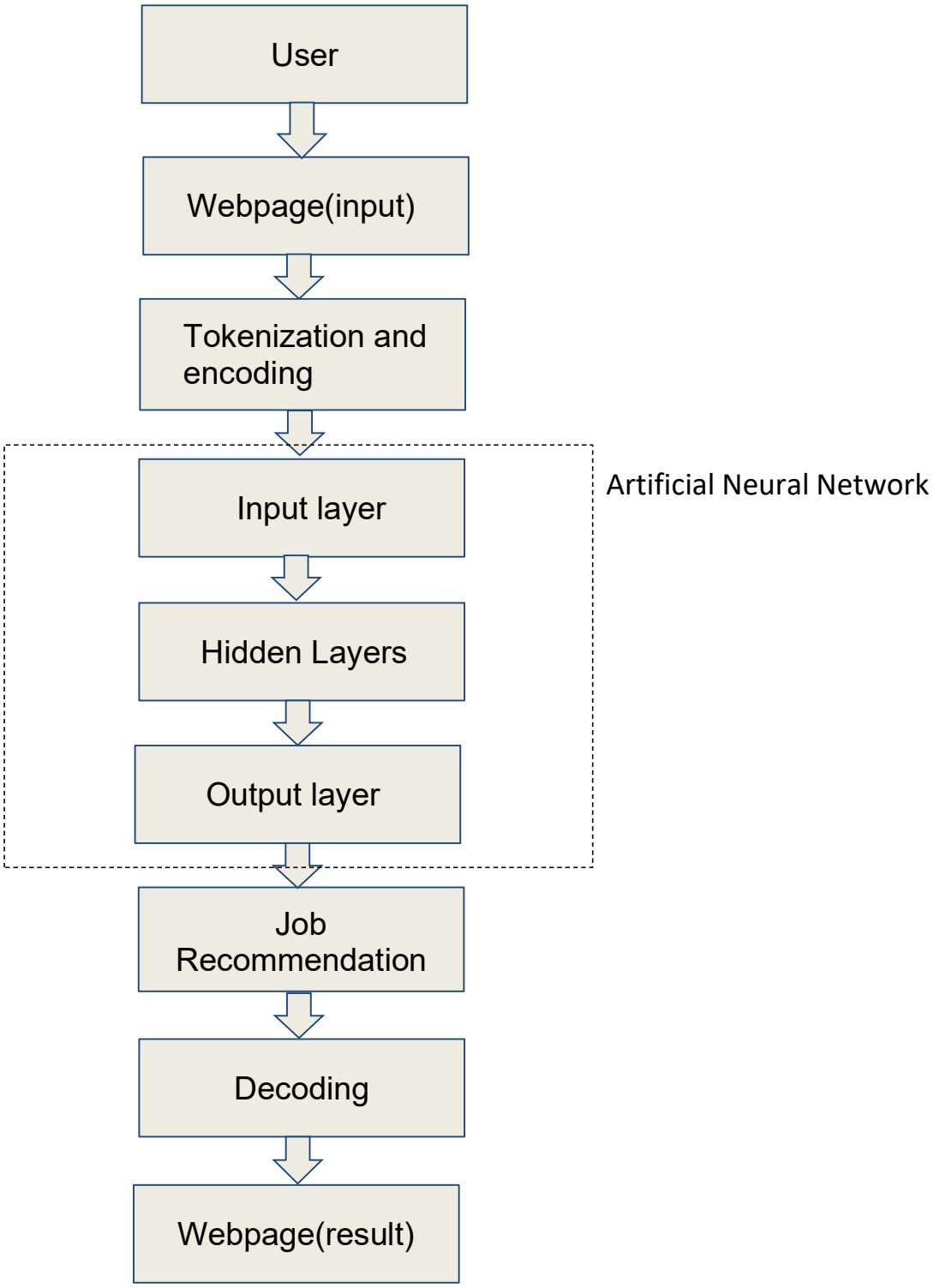
- JetBrains PyCharm Community Edition 2019
- Python 3.x (x=5, 6, 7)
- Flask
- TensorFlow
- Keras
- Sklearn
- HTML
- CSS
- Web browser – Google Chrome/ Mozilla Firefox/Microsoft Edge

Hardware Requirements

- PC running Windows to act as server to host chatbot locally
- Processor – intel core i5/i7

Design

The design of the project is as follows:



This chatbot consists of four main parts: front-end, knowledge-base, back-end and corpus which is the training data. The front end is accountable for enabling communication between the bot and the user. The package provided by Keras called as preprocessing is used to identify the intent and context of the user input. An appropriate response is generated from the users' intent.

The knowledge base defines the chatbots knowledge which is created by traning the model on the training data. The back-end is the Artificial neural network which had been trained to recommend jobs based on the user's input. The Sequential model provided by Keras is used to design the Artificial neural network.

Users will interact with the chatbot from the web pages. Users will carry out their interaction with the chatbot using natural language. The client side of the application is developed using g HTML5, CSS and JavaScript.

After the user input has been encoded into a vector, it is then passed to the neural network. The predict() function then selects a class label from the knowledge base for which the model gave the greatest confidence. The LabelBinarizer then transforms the class label into its corresponding job title.

The obtained result is then sent to the webpage. For the given input, a suitable job and the company which is providing that job will be shown.

Implementation

1. Loading data and splitting it into training dataset and test dataset:

```
def __init__(self, dataset_path, vocab_size, max_length):
    self.data = pd.read_csv(dataset_path, header=0, encoding='unicode_escape', names=['Query', 'Description'])
```

```
def split_data(self):
    # Split data to train and test (80 - 20)
    train, test = train_test_split(self.data, test_size=0.2)

    self.train_descs = train['Description']
    self.train_labels = train['Query']

    self.test_descs = test['Description']
    self.test_labels = test['Query']
```

The data collected from different sources is read using the functions in the Pandas package. It is the then split into two sets – training dataset and the test dataset. The training dataset is used to arrive at a formula for predicting future behaviour. Test dataset is used to test the hypothesis created via prior learning.

2. Encoding the data:

```
def data_encode(self):
    # define Tokenizer with Vocab Size
    tokenizer = Tokenizer(num_words=self.vocab_size)
    tokenizer.fit_on_texts(self.train_descs)
    x_train = tokenizer.texts_to_matrix(self.train_descs, mode='tfidf')
    x_test = tokenizer.texts_to_matrix(self.test_descs, mode='tfidf')

    encoder = LabelBinarizer()
    encoder.fit(self.train_labels)
    y_train = encoder.transform(self.train_labels)
    y_test = encoder.transform(self.test_labels)
```

The Tokenizer API is used to encode the text input to numbers as required by the deep learning models. First, it creates the vocabulary index for every word based on its frequency. Then it converts the frequency list into vectors where the length of each vector is the total number of words. The LabelBinarizer is used to assign a class label for each job title.

3. Creating a neural network using Keras Sequential Model:

```
def create_model(self):
    self.model = Sequential()
    self.model.add(Dense(conf_keras_first_go.dense, input_shape=(conf_keras_first_go.vocab_size,)))
    self.model.add(Activation(conf_keras_first_go.activation_function))

    self.model.add(Dense(conf_keras_first_go.dense))
    self.model.add(Activation(conf_keras_first_go.activation_function))

    self.model.add(Dense(conf_keras_first_go.labels))
    self.model.add(Activation(conf_keras_first_go.last_activation_function))

    # Compile the model
    self.compile_model()

def compile_model(self):
    self.model.compile(loss=conf_keras_first_go.loss,
                       optimizer=conf_keras_first_go.optimizer,
                       metrics=[metrics.categorical_accuracy, 'accuracy'])
```

Here, we initialize a new model using the Sequential class of Keras to implement a feedforward neural network. Then we can add as many layers to it as we like. For each layer, we pass the number of units and the activation function to be used. We pass the dataset to the first layer. The output of the previous layer is the input to the next layer.

4. Creating the user interface and connect them by routing:

```
@app.route("/")
def index():
    return render_template('index.html')
```

```
@app.route('/result', methods=['POST', 'GET'])
def result():
    if request.method == 'POST':
        result = request.form.getlist('Job')
        train_model()
        processed_text = first_go_model.prediction(result[0])
        result = {'Job': processed_text}
        return render_template("result.html", result=result)
```

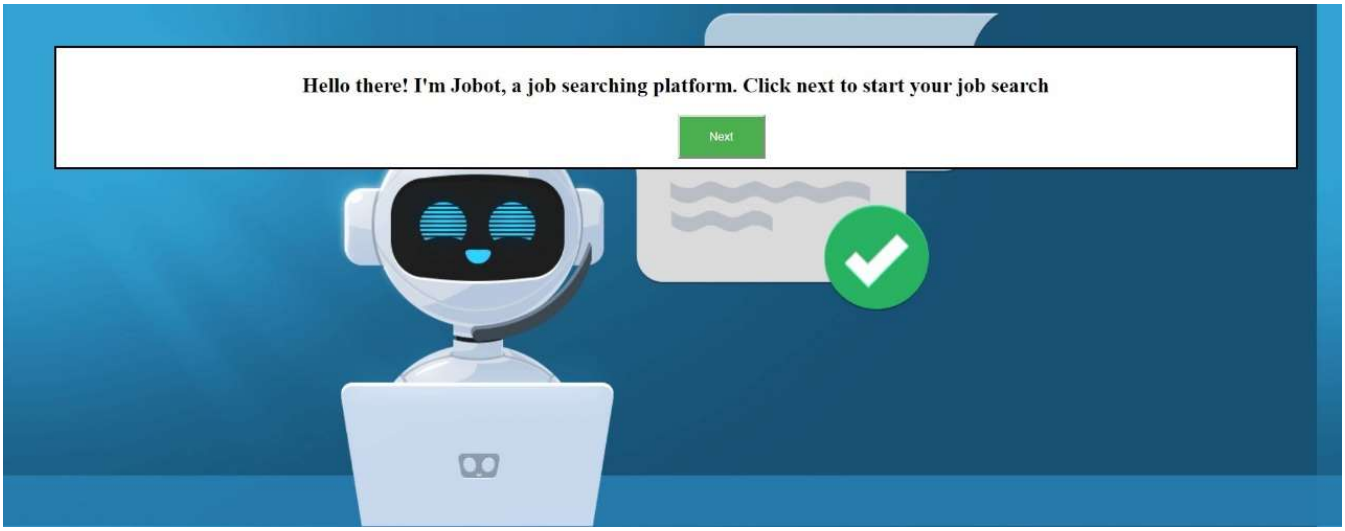
We use HTML and CSS to create the user interface where the user enters all the details such as his/her qualification, experience, skills, etc. All the pages are connected by using the route() function. The route function is used to map the specific URL with the associated function that is intended to perform some task. It is also used to access some particular page. For example, in this project, when the URL contains the string result, it performs the task of processing the input to predict the job title and display it on a new page.

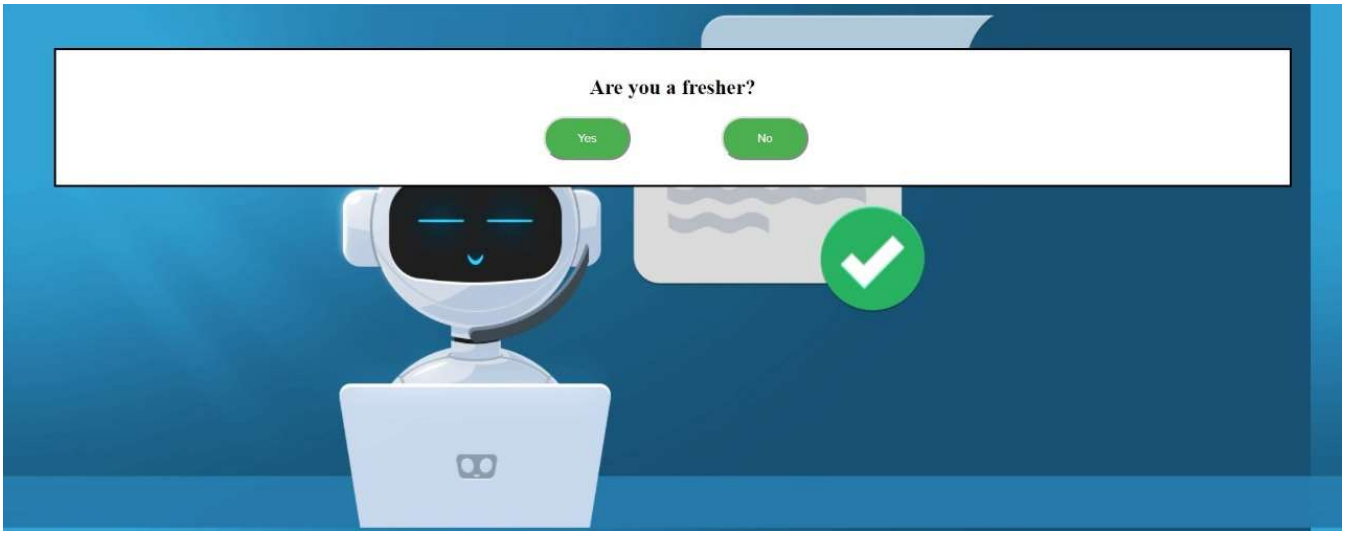
5. Predicting a suitable job:

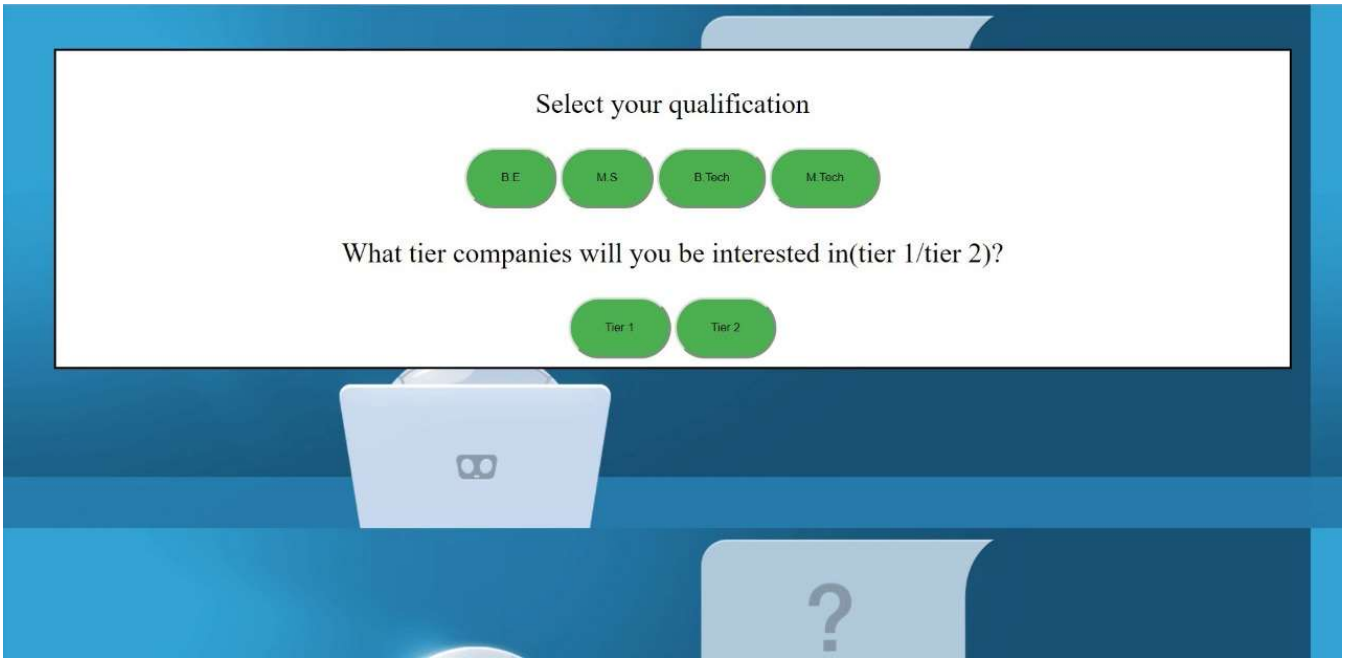
```
def prediction(self, user_text):
    # Encode the text
    encoded_docs = [one_hot(user_text, conf_keras_first_go.vocab_size)]
    # pad documents to a max length
    padded_text = pad_sequences(encoded_docs, maxlen=conf_keras_first_go.max_length, padding='post')
    # Prediction based on model
    prediction = self.model.predict(padded_text)
    # Decode the prediction
    encoder = LabelBinarizer()
    encoder.fit(self.test_labels)
    result = encoder.inverse_transform(prediction)

    return result[0]
```

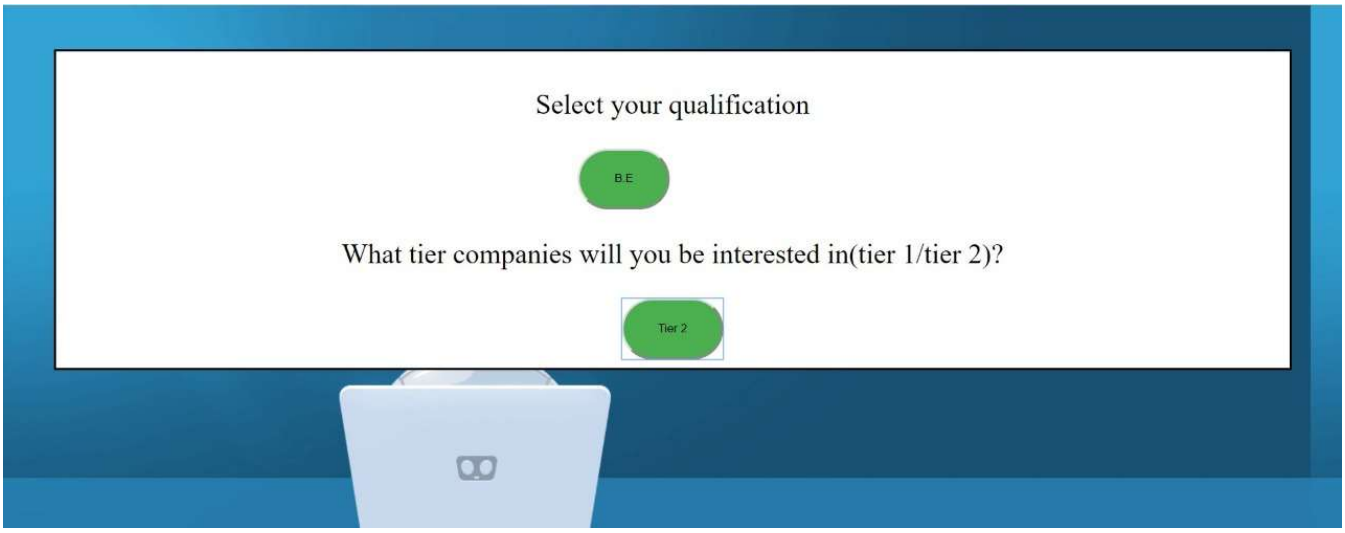
Here, the user data is again encoded using the one hot encoder which is then passed to the network. The output is then decoded to display the job recommendation.

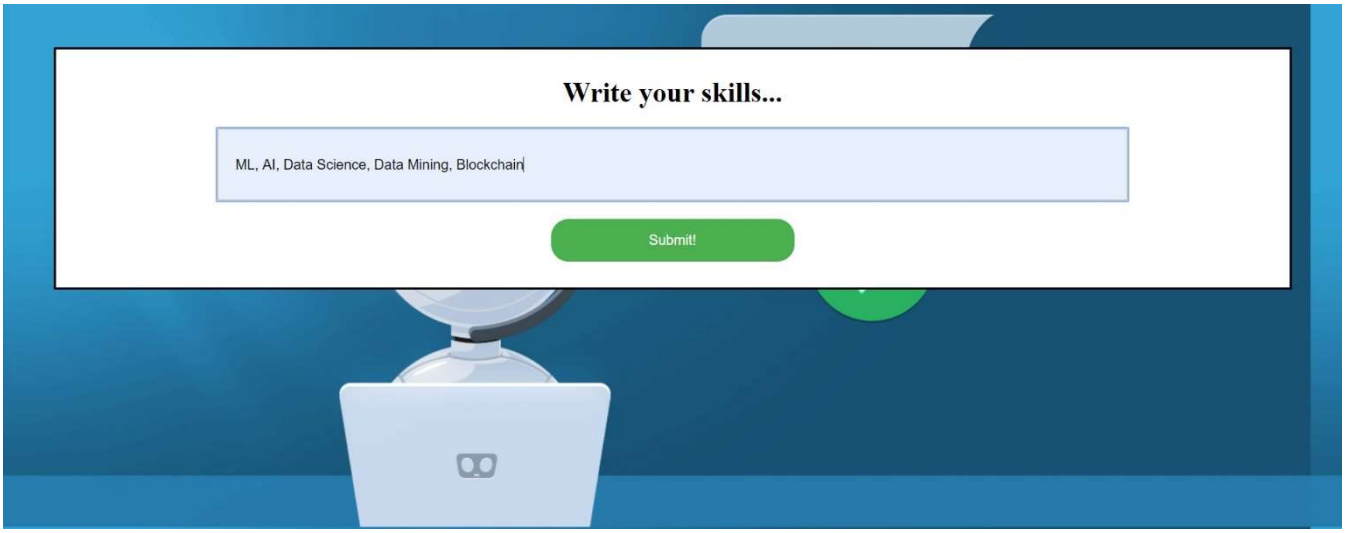
Results


Hello there! I'm Jobot, a job searching platform. Click next to start your job search
Next


Are you a fresher?
Yes
No


Select your qualification
B.E
M.S
B.Tech
M.Tech
What tier companies will you be interested in(tier 1/tier 2)?
Tier 1
Tier 2


Select your qualification
B.E
What tier companies will you be interested in(tier 1/tier 2)?
Tier 2


Write your skills...
ML, AI, Data Science, Data Mining, Blockchain
Submit!


The best job is:
Data and Analytics Manager @ Synup
OK