**Designing the data:**

The data set for a domain specific chatbot is subjective to the domain and its application usage. The reason for selecting job seeker profile as a specified domain is that – most of the profiles have similar structure and it is easy to add and update the data into the model. Not only that the conversation between a job seeker and recruiter can get into several topics and multi-levels like education, projects, experiences, certifications, etc. related information. This kind of multi-depth conversation is necessary to test out the contextualization of the framework.

Here, the data set is confined to English language because the python packages has all kind of NLP support for implementing and building the model.

We structured this data into this following model.

**Tag:** a unique name for each topic that the model covers

**Sample Patterns:** Examples of sample query of that topic. These are the questions asked by the recruiter.

**Possible Responses:** Sample responses of the topics. After identifying the topic, one of these responses will be randomly chosen as an answer to the query

**Context:** To identify the content of the topic.

**Specifics:** Specific details of the topic.

**Datatype:** The data type of responses for the topic.

**Sample data set**

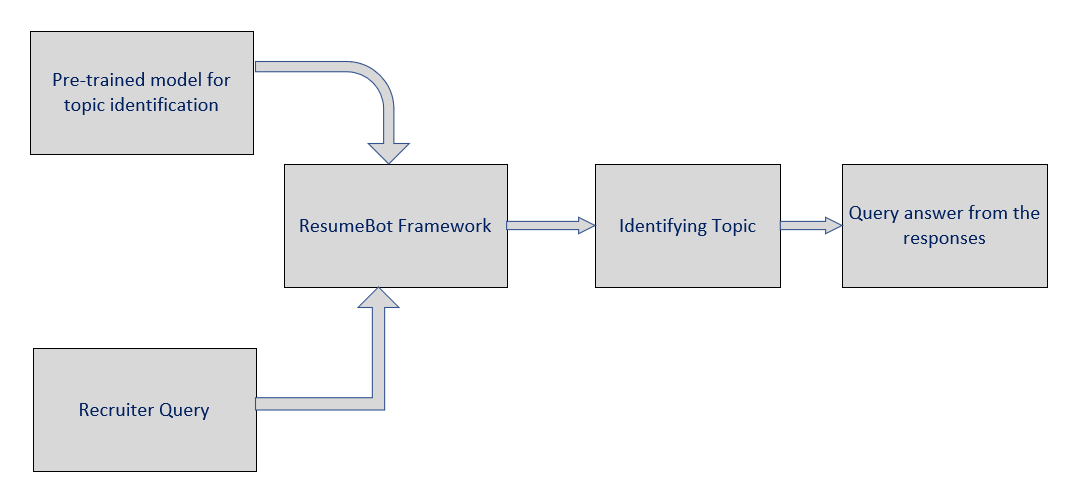


This model to structure chat data has two main advantages. Firstly, if someone has existing chat data, it can mark every query to associate with a tag. In this way, already existing query data and response can be attached to the training dataset. Secondly, if someone has no prior dataset; anyone can create and add data in this format. There is no hassle to specify the flow of the conversation. All the inputs in these 6 input categories can be set by the person who wants to build the chatbot through this framework. We can see that there is no need to pre-set any conditional flow of the conversation. As every class in this structure of the dataset is discrete; if new data occurs, it can be easily entered as one of these tags or in a new tag without worrying about maintaining the relationship between the new data with other classes.

**System design**

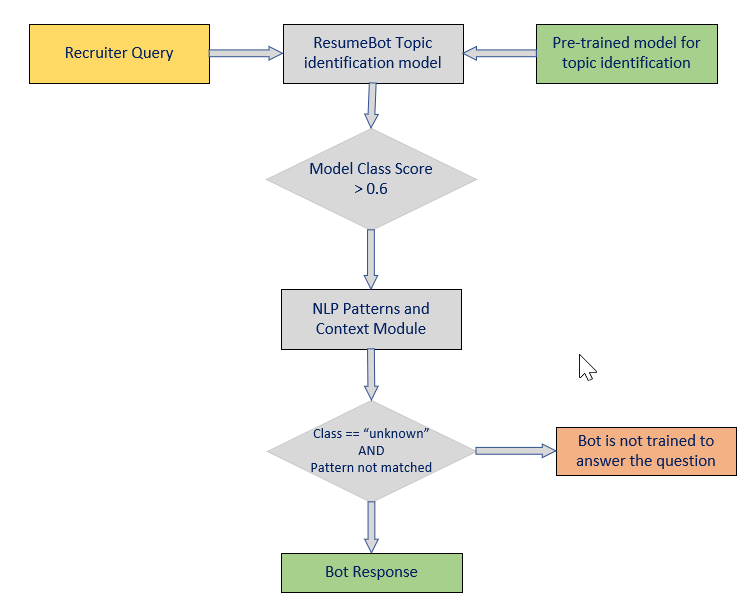
The ResumeBot is a domain-specific chatbot and we have a domain-specific data to train the chatbot. The data is pre-classified based on the topics(classes) and questions(patterns), and every class has some sample examples and some other data to track the flow of the conversation.

So, when a new query comes from a recruiter, we need to associate the query to our pre-existing classes based on the pattern that already exists in each class. Based on that topic identification; we give the user appropriate answers to their queries.



This looks like a text classification where the corpora are replaced with small sentences. Each class has only 3–10 sample sizes, and each sample contain only one sentence instead of a whole paragraph or document. Even the input sentence is one or two lines of sentences. This sparsity of data makes the model difficult to classify the data even with the well-established text classification techniques. Once the topic is identified, the appropriate response is picked from the dataset. When the query does not have any topic identity and if it matches with the “unknown” class, then responses are prepared based on NLP patterns.

Detailed Flow:



To build this type of system we divided the entire flow into 3 major parts

1. Pre-processing of patterns.
2. Writing NLP patterns
3. Building model
4. Pre-processing of patterns.

The patterns are processed and converted into data that a computer can understand for building the model

* **Tokenization**: We have used the tokenizer because the Reviews there might be many smiley details of some phrases like ‘YAAAY!!’, ‘Yappyyyy!!!’,’: P’ etc. These phrases don’t have any meaning or at least polarity cannot be determined with the use of these phrases. So, using tokenisation these phrases will not be considered by the model.

Example: “Amazon service is good >:-D“ . where :-D is a smiley and not a word a phrase. This symbol is ignored by the tokenizer.

* **Stop word Removal:** A stop word is a commonly used word that a pre-processing engine has been programmed to ignore so that we can save processing time and space. We imported “stopwords” package from NLTK library and customized the stopwords list based on our requirement. We ignored the following set of words from the list. ['not','what', 'which', 'who', 'whom', 'do', 'does', 'did', 'when', 'where', 'why', 'how']. Along with the common words, punctuation marks and special characters are also removed for building topic identification model.
* **Part-Of-Speech (POS) Tagging:** NLTK pos tagging is used to tag each word with the appropriate Part-of-speech. This helps us to understand the contextual meaning of the sentences.

The words having list of POS tags (['CC','DT','EX','MD','PRP','IN','PRP$','RP','TO','UH']) are not considered while building the model for topic identification. But they are used while making NLP pattern matching because these POS tags are important to understand the language flow.

**Example:** The word “Book” can be used as verb and also as a noun.

* **Lemmatization**: Normally lemmatization will return the existing word into the base or dictionary form of a word. We have used this pre-processing technique for the data as the patterns or queries can be given in the informal language and the model cannot predict the sentiment.

Example: Same word with different tenses have been brought down to its root level.

The words “projects”, “took” are converted as “project”,”take”.

* **Count Vectorisation:** To Increase the sparseness in the vector matrix we have converted all the uppercase letters into lower case letters. This made vector considering only unique words.

Example: The letter ‘Good’ can be written in the reviews as ‘Good’, ’good’, ’GOOD’ and many ways with upper and lower cases the vector matrix would consider every single case of good. After converting them into lower case, only ‘good’ is available, vector will consider the same.

* **TF-IDF Vectorisation:** Term frequency and Inverse document frequency is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. We have used TF-IDF to give more weight to the words which are used less.

Consider a query “what are the projects you worked” is converted into “what project worked” after performing the pre-processing steps. As an end result, it will have an array of weights based on its occurrence.

1. Writing NLP Patterns:

The topic identification model is not suitable for all types of questions. For the general questions like who are you? or Who is he? , mapping these into different classes are meaningless and it affects the model performance. In such scenarios, NLP patterns and Regex are helpful.

POS tags are stored along with the words to identify the pattern of the queries. Once the correct pattern is matched, the response can be obtained.

POS tags are helpful while maintaining the context of the conversation. If the conversation is about the job seeker’s bachelor education and when the recruiter asks the bot “When”, the bot responds with an appropriate answer based on its previous context.

1. Building model:

For this experiment, a deep learning neural network model is built with two hidden layers and an activation function of “softmax”. The model is built using TensorFlow package and the training is done with 1000 iterations.