**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**

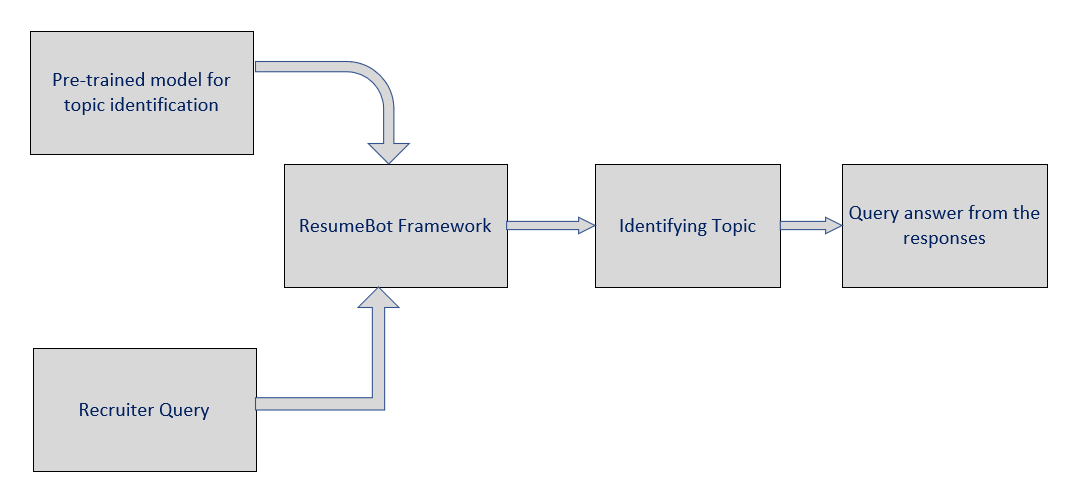
This model to structure chat data has two main advantages. Firstly, if someone has existing chat data, it can mark every query to associate with a tag. In this way, already existing query data and response can be attached to the training dataset. Secondly, if someone has no prior dataset; anyone can 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.

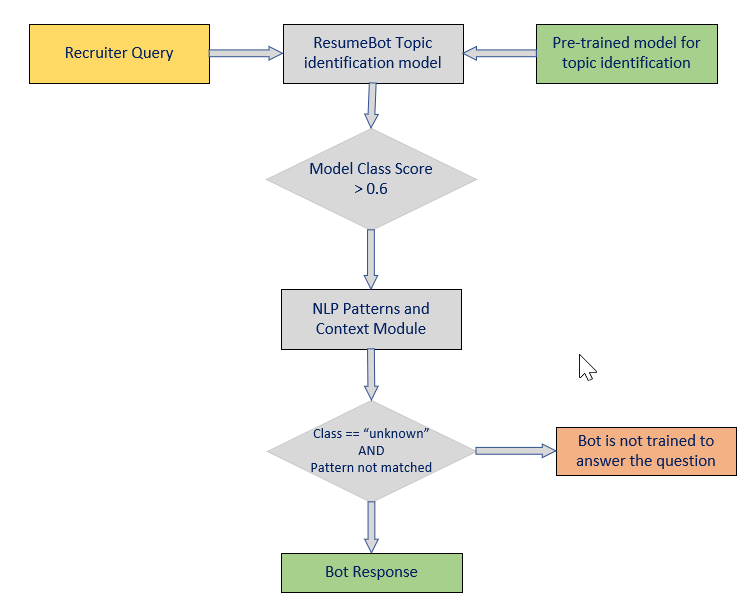
So, when a new query comes from a recruiter, we need to associate the query to our pre-existing classes based on the pattern that already exists in each class. Based on that topic identification; we give the user appropriate answers to their queries.





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.

We tried to implement the same in other models like SVM and Recurrent Neural Networks. SVM is good for building text classification model but it is not a probabilistic model. As per our requirement, we need the probability scores for classes.

Recurrent Neural Networks are feedback networks which is not necessary for this application. Because, the conversation is not going to be discrete.

This model is not built to get trained online because of a few limitations. To provide an online training, we need to feed in the response along with the queries. Also, we need to recalculate the weights of neural networks which is a time-consuming process for 1000 iterations.

**Use Cases:**

Building a complete chatbot is quite complex because we need to prepare the vocabularies and NLP libraries for that. So, in this ResumeBot application, we have implemented few cases and later we will try to complete the application and integrate with our portfolio along with an interview appointment module.

**Handling typos and basic grammar:**

To match with NLP patterns, the sentence should be grammatically correct and void of spelling mistakes. To handle that, we are using a python package called [Gingerit](https://gingerit.readthedocs.io/en/latest/). This helps to correct the words based on its syllabic sound and correct basic level of grammatical errors.

Sample Text: 'tel abt bachler education '

Output: ' Tel about bachelor education'

**Bot Introduction and basic salutation:**

When the recruiter opens the ResumeBot chat window, it should be a blank window. The bot initiates the chat with an introduction and should tell its capabilities. On introduction, ResumeBot asks the recruiter’s name and address him by his/her name. To implement this, we used NLTK’s named-entity recognition technique which extracts the entity with the help of POS tagging based on its context in the sentence.

Along with this, we handled a few basic questions like who are you? How are you? using NLP patterns.

**Maintaining context between conversations:**

The context between sequential chats are necessary to maintain for a chatbot. In this ResumeBot, context of every chat is stored in database and also passed between client and server side through cookies. When a recruiter first queries the bot about the job seeker’s master’s education and if he asks next where did he complete, then the bot tries to understand from the previous query that the recruiter is trying to ask where the job seeker complete his master’s degree. We achieved this, with the help of NLP techniques.

**Challenges:**

**Building vocabularies:**

We need to build word sets to handle different set of queries. For example, “master education” can also be queried as “graduate”. Similarly, “Bachelor degree” can be queried as “undergraduate degree”. Like these, words have synonyms and can be used differently based on the context. Resolving these issues require a lot of time and language specialists to build a possible set of words.

We tried using NLTK’s Wordnet package which is a lexical database of English language. But, we require domain based knowledge to build those word sets and it is subjective to this application.

**Multiple inputs:**

When a recruiter queries two or more questions in a single chat, then it is difficult for the bot to split, understand and answer the questions.

For Example, “When is he completing his master’s degree and where is he doing?”. Here, the recruiter likes to know the person’s graduation date and the location. Even though, the question points to a single class “Master Education” but the user expects two different answers.

**Multiple outputs:**

This bot responds quickly to the query asked by the recruiter and stores the context of the query. If the recruiter asked something wrong and if the person tries to correct himself in the next query, this causes a confusion. This bot is not designed to handle these situations.

**Evaluation:**

ResumeBot is not only a text classification model, it also uses NLP techniques for pattern matching. In this domain specific chatbot, the samples for each class is 3-10 sentences. This makes it hard to split and test the model. The performance and quality of this bot is based on the user’s satisfaction and it cannot be measured using machine learning evaluation methods.

We evaluated this application by taking reviews from different users for each question and also saved the user’s query and bot’s response for our analysis.

**Future Enhancements:**

This ResumeBot is not fully operational because we need to address the challenges mentioned before. Also building vocabularies and adding more patterns for each class improves the quality of the bot. Once we addressed those challenges, we are planning to add the appointment schedule module. This helps the recruiter to book an appointment with the profile owner for the next step.