COMPILER DESIGN PROJECT

- BALAJI S(2018103014)
DHANANJEYAN A K(2018103523)
VIJAY J A(2018103622)

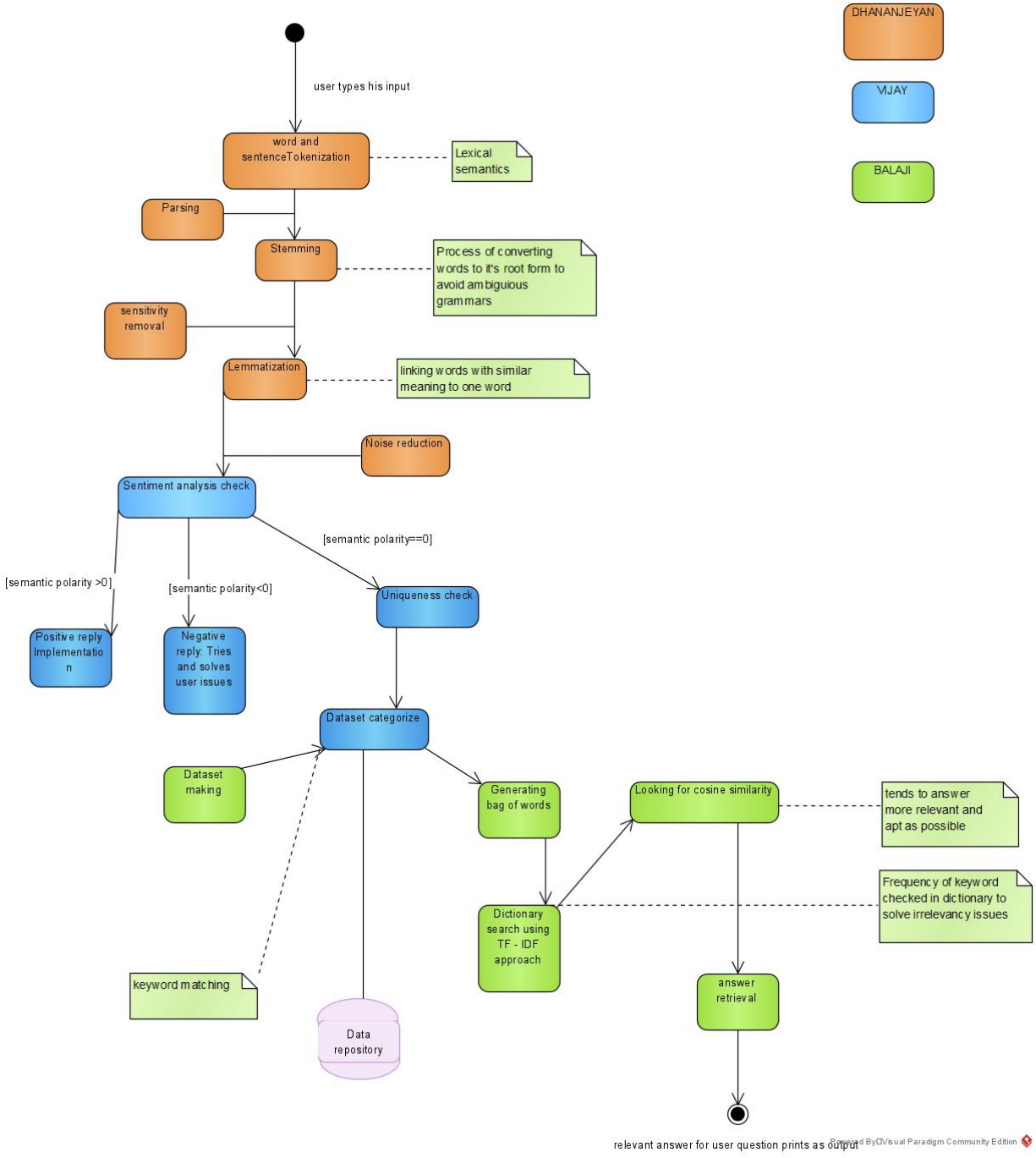
IMPLEMENTATION OF CHATBOT USING SEMANTIC ANALYSIS

DOMAIN:CRICKET(CRICBOT):

INTRODUCTION:

Chatbots are increasingly becoming **common and a powerful tool** to engage online visitors by interacting with them in their natural language. Earlier, websites used to have live chats where agents would do conversations with the online visitor and answer their questions. But, it' s obsolete now when the websites are getting high traffic and it' s expensive to hire agents who have to be live 24/7. Training them and paying their wages would be a huge burden on the businesses. Chatbots would **solve the issue by being active around the clock** and engage the website visitors without any human assistance.

SYSTEM ARCHITECTURE:



NLP:

NLP is a way for computers to **analyze, understand, and derive meaning from human language** in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as **automatic**

summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

MODULE 1 :

READING DATA:

The main issue with text data is that it is all in text format (strings). However, the Machine learning algorithms need some sort of numerical feature vector in order to perform the task. So before we start with any NLP project we need to pre-process it to make it ideal for working. Basic text pre-processing includes:

- Converting the entire text into **uppercase** or **lowercase**, so that the algorithm does not treat the same words in different cases as different

- **Tokenization** : Tokenization is just the term used to describe the **process of converting the normal text strings into a list of tokens (i.e) words** that we actually want. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings.

```
#Reading in the corpus
with open('chatbot.txt','r', encoding='utf8', errors ='ignore') as fin:
 raw = fin.read().lower()
#TOkenisation
sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words

# Preprocessing
lemmer = WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

- Removing **Noise** i.e everything that isn’ t in a standard number or letter.
- Removing the **Stop words**. Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words
- **Stemming**: Stemming is the **process of reducing inflected (or sometimes derived) words** to their stem, base or root form—generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stemtization”, the result would be a single word “stem”.
- **Lemmatization**: A **slight variant of stemming** is lemmatization. The major difference between these is, that, stemming can often create **non-existent words**, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of lemmatization are that “run” is a base form for words like “running” or “ran” or that the word “better” and

"good" are in the same lemma so they are considered the same.

MODULE 2 :

SENTIMENT ANALYSIS:

Once chatbots could communicate effectively, the next step was to improve user experience. After all, it isn't enough to just provide the right answers, you want to create a delightful experience for your customers. With the help of **sentiment analysis**, chatbots could understand whether the conversation was going well and respond to customer emotions accordingly.

```
def greeting(sentence):
    """If user's input is a greeting, return a greeting response"""
    if(sentence=="how's going?" or sentence=="how are you?" or sentence=="how are you" or sentence=="whatsup"):
      return "I'm fine"
    if(sentence=="who are you?" or sentence=="who are you"):
      return random.choice(ABOUT_RESP)
    if sentence in FAVOURITES:
      return random.choice(FAVOURITES_RESPONSES)
    for word in sentence.split():
        analysis=TextBlob(word)
        if analysis.sentiment.polarity > 0:
            return random.choice(POSITIVE)
        elif analysis.sentiment.polarity < 0:
            return random.choice(NEGATIVE)
        else:
            if word.lower() in GREETING_INPUTS:
              return random.choice(GREETING_RESPONSES)
            elif word.lower() in TEAMS:
                for i in range(0,7):
                    if TEAMS[i].lower() == word.lower():
                        return TEAMS_RESPONSES[i]
                    else: continue
            elif word.lower() in ABOUT:
              return random.choice(ABOUT_RESP)
```

MODULE 3:

GENERATING RESPONSES:

After the initial pre-processing phase, we need to transform text into a **meaningful vector (or array) of numbers**. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

It is because any information about the order or structure of words in the document is discarded and the model is only **concerned with whether the known words occur in the document, not where they occur in the document.**

The intuition behind the Bag of Words is that documents are similar if they have similar content. Also, we can learn something about the meaning of the document from its content alone.

For example, if our dictionary contains the words {Learning, is, the, not, great}, and we want to vectorize the text "Learning is great", we would have the following vector: (1, 1, 0, 0, 1).

```
# Generating response
def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):
        robo_response=robo_response+random.choice(APOLOGIES)
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

TF-IDF APPROACH:

A problem with the **Bag of Words approach** is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much "informational content". Also, it will give more weight to longer documents than shorter documents.

One approach is to **re-scale the frequency of words** by how often they appear in all documents so that the scores for frequent words like "the" that are also frequent across all documents are penalized. This approach to scoring is called Term Frequency-Inverse Document Frequency, or TF-IDF for short, where:

Term Frequency: is a scoring of the frequency of the word in the current document.

TF = (Number of times term t appears in a document)/(Number of terms in the document)

Inverse Document Frequency: is a scoring of how rare the word is across documents.

IDF = 1+log(N/n), where, N is the number of documents and n is the number of documents a term t

has appeared in.

```
# Keyword Matching
POSITIVE = ("Cool","Awesome",":)",":))","Hope you like this interaction",
            "I'm very much interested in talking to you","Hi-fi")
NEGATIVE =(";(",":(","Do you feel bored?",
           "Are you not interested??","Cheer up!!!")
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up",
                   "hey","Good morning","Good evening","namaste","vanakkam")
GREETING_RESPONSES = ["hi", "hey", "nods", "hi there", "hello", "I am glad! You are talking to me"]

FAVOURITES = ("which team is your favourite?","favourite team","what team do you like","favourite team?","what team do you like?",
              "your favourite?","your favourite")
FAVOURITES_RESPONSES = ("CSK","RCB","MI","RR","SRH","DC","KKR","kXIP")

TEAMS = ["csk","rcb","kkr","kxip","kkr","rr","mi","srh","i support","i support csk","i support rcb",
         "i support mi","i support RR","i support SRH","i support DC","i support KKR","i support KXIP"]
TEAMS_RESPONSES =["Start the whistles!!","Play bold!!","Chalo paltans!!","Halla bol!!","Orange army","Naya Dilli!!!","Purple Army","Dhoom punjabi"]

APOLOGIES =["Pardon!!??","Sorry! I'm not able to get it","Tell me more precise","What do you mean?"]
```

COSINE SIMILARITY:

TF-IDF weight is a weight often used in **information retrieval** and **text mining.** This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus

Cosine Similarity (d1, d2) = Dot product(d1, d2) / ||d1|| * ||d2||

where d1,d2 are two non zero vectors.

To generate a response from our bot for input questions, the concept of document similarity will be used. We define a function response which searches the user's utterance for one or more known keywords and returns one of several possible responses. If it doesn't find the input matching any of the keywords, it returns a response: " I am sorry! I don't understand you"

```
flag=True
print("ROBO: Hola! I'm Crickbot. I will answer your queries about IPL. If you want to exit, type Bye!")
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        elif(user_response in TEAMS):
          for i in range(0,len(TEAMS)):
            if user_response==TEAMS[i]:
              print("ROBO: "+TEAMS_RESPONSES[i])
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! Enjoy the matches..")
```

RESULTS:

```
        for i in range(0,len(TEAMS)):
            if user_response==TEAMS[i]:
                print("ROBO: "+TEAMS_RESPONSES[i])
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! Enjoy the matches..")
```

```
ROBO: Hola! I'm Crickbot. I will answer your queries about IPL. If you want to exit, type Bye!
tell me about tournament format
ROBO: Tournament format:
Currently, with eight teams, each team plays each other twice in a home-and-away round-robin format in the league phase;
At the conclusion of the league stage, the top four teams will qualify for the playoffs;
The top two teams from the league phase will play against each other in the first Qualifying match, with the winner going straight to the IPL final and the lose
Meanwhile, the third and fourth place teams from league phase play against each other in an eliminator match and the winner from that match will play the loser
```

CONCLUSION AND FINAL SUMMARY:

The necessity of NLP is highly dependant on how your chatbot is built, and what you want it to accomplish.There are a few ways in which this can be determined.In the context of chatbots, **integrating NLP means adding a more human touch**. If you' ve built a chatbot and deployed it for public use, it' s likely that you' ve seen users attempting to ask it questions. It seems very much in line with human nature that users will try to stump the chatbot and throw it off. You can attempt to remedy this by adding default responses, however this tends to fall short quite often as it is nearly impossible to predict which questions will be asked, as well as the manner in which they will be asked.

Overall, **NLP is likely the next step forward** in bridging some of the concerns that users, businesses and developers experience with chatbots. It fills gaps wherever they fall, and help ensure that your chatbot is one that anyone can enjoy interacting with.

REFERENCES:

[1] C. J. N. J. S. S. Divya Madhu, "A novel approach for medical assistance using trained chatbot," in International Conference on Inventive Communication and Computational Technologies (ICICCT), 2017.

[2] S. A. N. H. Hameedullah Kazi, "Effect of Chatbot Systems on Student' s Learning Outcomes," SYLWAN, 2019.

[3] BORKAN, "www.blog.casper.com," 21 september 2016. [Online]. Available: https://blog.casper.com/meet-insomnobot-3000/.

[4] D. B. Mesko, "The Medical Futurist," The Medical Futurist Institute, 2020. [Online]. Available: https://medicalfuturist.com/magazine/.

[5] Saurav Kumar Mishra, DhirendraBharti, Nidhi Mishra," Dr.Vdoc: A Medical Chatbot that Acts as a virtual Doctor", Journal of Medical Science and Technology, Volume: 6, Issue 3,2017.

[6] Abdul-Kader Sameera, John Woods, "Survey on Chatbot Design Techniques in Speech Conversation Systems", (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 6, no. 7, 2015.

[7] S. Divya, V. Indumathi, S. Ishwarya, M. Priyasankari, S. Kalpana Devi, "A Self-Diagnosis Medical Chatbot Using Artificial Intelligence", J. Web Dev. Web Des., vol. 3, no. 1, pp. 1-7, 2018.