**Overview:**

The objective of this project is to create a chat bot to converse with users and assist them in their use of Twitter. I will be using the tweepy Python library to connect to the Twitter API and pull any relevant information for the user. Two different language processing techniques (rule based, and similarity based) will be used to try and understand what the user is saying and formulate a response.

**Aims:**

Must:

* Use simple rule based AIML to reply to basic questions and greetings from the user
* Connect and use the Twitter API in some way to assist the user
* Implement cosine similarity to catch more forms of input from the user

Should:

* Allow the user to search for tweets containing a specific word or category
* Allow the user to search for a Twitter account’s most recent tweet
* Simulate full conversation with the user as well as implement memory functionality such as retaining the users name
* Have seemingly intelligent random responses for a more unique experience

Could:

* Upon not finding the account the user is searching for, give them a list of possible accounts to choose from
* Allow the user to tweet or reply to tweets using the chat bot
* Allow the user to retweet or save tweets they like with the chat bot

**System Requirements:**

System:

All computers running any major non mobile OS should be able to run the program with no additional hardware or peripherals required except a standard keyboard. Python is required to be installed as well as several Python libraries:

* Tweepy
* AIML
* NLTK
* Sklearn
* Csv
* Math
* re
* Collections
* Random

User:

* Users can search for a Twitter account’s most recent tweet
* Users can search for tweets by category or word
* Users can greet and ask basic courting questions to the robot with limited amounts of memory
* Users can save and or retweet tweets

**Techniques to use:**

Rule Based AIML:

Explanation: AIML is an artificial mark-up language. It works by defining patterns with corresponding responses. When a message is passed through if it can be matched with a pattern, the pattern’s corresponding response is then given along with any special rules being executed.

Implementation: An XML file will be created containing all the patterns we are looking for; each pattern will have an appropriate response as well as the ability to pull information from the message given. Upon receiving a message, it will be checked (using AIML’s simple rule system) against every phrase for a match. If a match is found the response is given, otherwise we will move onto cosine similarity. Some responses will have special rules such as randomness or memory to make the chat bot appear more intelligent.

Cosine Similarity/Bag of words model:

Explanation: Cosine Similarity measures the similarity between two vectors. It does this by calculating the cosine of the angle between the two vectors to see how closely they point in the same direction. Some words appear more often such as ‘the’ or ‘a’ as such we change the weighting based on the frequency of the word so that they do not have a big impact on our decision.

Implementation: If AIML is unable to find a response the message will be checked using cosine similarity. I will vectorise the message obtaining a vector with each word and its corresponding frequency. A bag of words will be created from a CSV of phrases we would like to check using cosine similarity, this will give us a frequency of all words we are checking and thus will allow us to weight them so that words that appear less often are weighted higher. The vector of the message is then compared with a vector of each phrase in the csv, adjusted by our bag of words, and the phrase that is most like our message is chosen. I will then output its corresponding result. Cosine similarity will also be used when a Twitter account cannot be found and a choice is given to the user, the user’s choice will be compared to the list of possible accounts in the same way as previously described.

**Pseudo code:**

INPUT Message

Get Response from AIML

IF Response = Search for user:  
 Search for user

IF user found:

PRINT(Recent Tweet)

ELSE:

PRINT(List of Potential users)

INPUT Choice

Preform Cosine Similarity between choice and Potential users

Print(Recent Tweet of most likely user)

IF Response = Search for category:

Result = Search for tweet by given category

PRINT(Result)

IF Response is null:

Get Response via cosine similarity

IF Response = Search for user:  
 Search for user

IF user found:

PRINT(Recent Tweet)

ELSE:

PRINT(List of Potential users)

INPUT Choice

Preform Cosine Similarity between choice and Potential users

Print(Recent Tweet of most likely user)

ELSE:

PRINT(Response)

ELSE:

PRINT(Response)