Fatih Karaoglanoglu
Baraa Orabi

# Project's Final Report
# F.A.T.I.M.A.

03/05/2016
CS 461-2

## Overview

F.A.T.I.M.A. (FATIMA's Artifically Tweeting Intelligent Mechanical Autotyper) is software system that is composed of different parts that all together result in managing a Twitter account (twitter.com/fatimasnowreal). Fatima relays on a legacy open source implementation of A.L.I.C.E. (Artificial Linguistic Internet Computer Entity). It also relays on Twitter's API for generating raw content that will be used to generate the tweets. Twitter's API is also used to interact with Fatima's Twitter account. And finally, there is the part of Fatima that ensures that the output of different parts is formatted suitably to needs of the other parts.

## Language Piece

Fatima's backbone is an open source implementation of Alice. The implementation that we relied on is a Python implementation. The library we used is pyAIML library (github.com/creatorrr/pyAIML). The fact that pyAIML is implemented in Python allowed us to easily utilize it and test it since Python is such an easy programming language to design and run on the fly. However, even though pyAIML provided the rule-based system that Alice documentation specifies, it lacked the language content and vocabulary that Alice (and Fatima in turn) needed to generate sound sentences. For that we used another open source content. This was American English AIML dataset 1st and 2nd editions (code.google.com/archive/p/aiml-en-us-foundation-alice/). Having obtained this dataset, we used it to train an Alice instantiation of pyAIML (check file languagePart.py). Now, we have our language backbone setup and ready for operation.

## Content Piece

To provide content for the Alice backbone, we used Twitter's API to download a Twitter user whole tweeting history. Of course to do that we had to register to Twitter API system to have API access to their system. We used an open source implementation (github.com/tweepy/tweepy) that retrieves tweets of a specific user. We choose Karyn MeltzSteinberg (twitter.com/kms_meltzy) as a Twitter user. After downloading all her tweets, we had to run some language reformatting operations on the tweets to remove stuff like Twitter mentions and hashtag symbols. For that we used some regular expressions (check file regextweet.py). Now with the tweets ready and formatted, we send them to our Alice backbone to get her reply which will be the tweet that Fatima will tweet on her Twitter account.

## Twitter Piece

The twitter piece was responsible of taking what Fatima has produced of tweets and to tweet them via @fatimasnowreal Twitter account. We again here used Python to interact with Twitter's API (check file fatima.py).

## Results

We ran Fatima on the server of one of our (ex)teammates (Cagdas Oztekin). We ran Fatima for about a day and it has tweeted about 50 tweets. Tweets can be viewed on twitter.com/FatimasnowReal.

```
from __future__ import division
import time
from time import strftime, gmtime
import csv
import tweepy
import random
from random import randint

from tweepy import OAuthHandler

# connect to the database parameters are: [hostname], [username], [password], [databasename]
# planned to run on the server with address 178.62.59.61 (http://cgds.me)
# db = MySQLdb.connect("localhost","fatima","fatimarulez9","fatima" )

# twitter api credentials for fatima
consumer_key = 'pMdGMeuNDVQhmW65NCCp6X5Pz'
consumer_secret = 'fSUajJoqZj31eq5p7tyOuwkerdypRXqJ9ewzHLqrWfM68ElxOQ'
access_key = '705756931923845120-vrYJPDiDUyd6QtIkSls8XB12bZsaxra'
access_secret = 'yAssSsjVE1id9LbJNry0GBVkmGY5oXWfTtUGf6aW52rzf'

# authorize the application to use twitter apai
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tweepy.API(auth)

# BELOVED FUNCTIONS
# @find_followers returns the list of users that follow the user with the given username
# both parameters set to none by default to enable the usage of the function with either of them
# flow_window will be a varying duration in case fatima gets timeouts (CS421 SPOOF)
def find_followers(screen_name = None, twitter_id = None, flow_window=10):
        followers = []

        for page in tweepy.Cursor(api.followers_ids, screen_name=screen_name).pages():
                followers.extend(page)
        time.sleep(flow_window)

        return followers

# @find_follows returns the list of users that are followed by the user with the given username
def find_follows(screen_name = None, twitter_id = None, flow_window=10):
        follows = []
        for page in tweepy.Cursor(api.friends_ids, screen_name=screen_name).pages():
                follows.extend(page)
                time.sleep(flow_window)
        return follows

def is_active(screen_name = None, twitter_id = None, flow_window=10):
    tweet = api.user_timeline(id = twitter_id, count = 1)[0]

    # users are assumed to be active if they have tweeted in 2016
    return str(tweet.created_at)[:4] == '2016'

# id for calkan_cs


i = 0

# close file only to reopen with each iteration to constantly update the file
f = open("fatima.txt", "a")
f.write("\n##### RUN AT ##### " + strftime("%Y-%m-%d %H:%M:%S", gmtime()) + " #####\n")
f.close()

congestion_window = 50
flow_window = 10

fu = open("users.txt", "r")
i = fu.readline()
```

```
if(i == ''):
        i = 0
else:
        print "Use existing users text file"
        i = int(i)
potential_users = []
for p in fu:
        potential_users.append(int(p))
fu.close()

if(len(potential_users) == 0):
        potential_users = [1234176577]


while(i < len(potential_users)):

        # open and close the file with each iteration to see the updates of the file
        # append mode
        f = open("fatima.txt", "a")

        # rewrite the current users to a file in case the program shuts
        fu = open("users.txt", "w")
        fu.write(str(i) + "\n")
        for p in potential_users:
                fu.write(str(p) + "\n")
        fu.close()

        try:
                currentID = potential_users[i]
                current = api.get_user(currentID).screen_name

                followers = find_followers(screen_name=current)
                follows = find_follows(screen_name=current)

                # number of followers and followed by
                num_followers = len(followers)
                num_follows = len(follows)

                #find the people that follow AND are followed by the given user
                mutual_ids = list(set(follows).intersection(followers))
                num_mutual = len(mutual_ids)

                cgds_coefficient = num_mutual/num_follows

                # print the current user's ratio and follow her if ratio is greater than 0.66 and has
been active in 2016
                print "User with name " + current + " has ratio: " + str(cgds_coefficient) + " Time: " +
strftime("%Y-%m-%d %H:%M:%S", gmtime())
                f.write("User with name " + current + " has ratio: " + str(cgds_coefficient) + " Time: "
+ strftime("%Y-%m-%d %H:%M:%S", gmtime()) + "\n")
                if(cgds_coefficient > 0.66 and is_active(twitter_id=currentID)):
                        print "Followed user " + current + " with id " + str(currentID) + " Time: " +
strftime("%Y-%m-%d %H:%M:%S", gmtime())
                        f.write("Followed user " + current + " with id " + str(currentID) + " Time: " +
strftime("%Y-%m-%d %H:%M:%S", gmtime()) + "\n")
                        api.create_friendship(currentID)

                # add the mutual followers to the potential users list
                # stop adding people to the list if list already contains enough people
                for k in mutual_ids:
                        r = randint(0,9)
                        # add some non-determinism, 1/10 chance to add a user to potential users list or
not
                        if k not in potential_users and len(potential_users) < 1000 and r == 4:
                                potential_users.append(k)

                f.write("Potential users length: " + str(len(potential_users)) + " currently iterating
index: " + str(i) + " Time: " + strftime("%Y-%m-%d %H:%M:%S", gmtime()) + "\n")
                print "Potential users length: " + str(len(potential_users)) + " currently iterating
index: " + str(i) + " Time: " + strftime("%Y-%m-%d %H:%M:%S", gmtime())

                i += 1
                # set the increased durations to sleep to their default values
                congestion_window = 50
                flow_window = 10


        except Exception, e:
                print "Exception raised, sleeping."
                f.write("Exception raised. Sleeping.\n")
                time.sleep(congestion_window)
                congestion_window += 1
                flow_window += 1

        f.close()
```

```python
import re


with open('tweets', 'r') as f:
        read_data = f.readlines()
        for l in read_data:
                l = re.sub(r'#\w+ ?', '', l)
                l = re.sub(r'http\S+', '', l)
                l = re.sub(r'@\w+ ?', '',l)
                l = re.sub(r'\"? *RT *:*','',l)
                l = re.sub(r'\"','',l)
                print l
```

```python
import pyAIML/Kernel
import glob

# The Kernel object is the public interface to
# the AIML interpreter.
k = Kernel.Kernel()

# Use the 'learn' method to load the contents
# of an AIML file into the Kernel.
files = glob.glob("pyAIML/aiml-en-us-foundation-alice/*")
for f in files:
        k.learn(f)

# Use the 'respond' method to compute the response
# to a user's input string.  respond() returns
# the interpreter's response, which in this case
# we ignore.


# Loop forever, reading user input from the command
# line and printing responses.
a = True
if a:
        with open("formatted.formatted.tweets") as f:
                read_data = f.readlines()
                for l in read_data:
                        #print l
                        print k.respond(l)
else:
        while True: print k.respond(raw_input("> "))
```