Implementation of Voice Captcha using Python

by

RITIK SINGH – 20BCE1384

A project report submitted to

Dr. Ganala Santoshi

SCOPE

In fulfilment of the requirements for the course of

CSE4015 – Human Computer Interaction

in

B. Tech. COMPUTER SCIENCE ENGINEERING

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

November 2022

Abstract

CAPTCHA challenges are used all over the Internet to prevent automated scripts from spamming web services. However, recent technological developments have rendered the conventional CAPTCHA insecure and inconvenient to use. In this paper, we propose vCAPTCHA, a voice-based CAPTCHA system that would: enable more secure human authentication, more conveniently integrate with modern devices accessing web services, and help collect vast amounts of annotated speech data for different languages, accents, and dialects that are under-represented in the current speech corpora, thus making speech technologies accessible to more people around the world. vCAPTCHA requires users to speak their responses, in order to unlock or use different web services, instead of typing them.

In our project we will be using Python Language to implement the Voice Captcha in which we a user does not have to type the Captcha. At place of typing the Captcha user can simply speak the Captcha.

In this project we have use Google Speech for Voice Recognition , tkinter for developing an interactive UI , PIL and Captcha.image for generating the Captcha and re to use regex for validation of Captcha.

Introduction

There are rising doubts about the effectiveness and security of CAPTCHA, the Completely Automated Public Turing Test to tell Computers and Humans Apart. CAPTCHA and ReCAPTCHA systems are widely used on the Internet, mostly by web services, to differentiate between human users and automated scripts. The most common form of CAPTCHA relies on the limitation of computer vision techniques by presenting users with a short, visually distorted string, and asking them to type out its contents. The emerging concern, however, is based on the fact that modern optical character recognition techniques can solve the most difficult variants of Google's ReCAPTCHA with more than 99% accuracy.

Given the recent advancements in speech processing technologies, we see speech as a very promising modality for CAPTCHA systems, one that could solve the aforementioned problems of usability and security that plague current systems. With more than 70% of users finding speech to be more efficient than typing [9], [10], and 20% of Google mobile searches happening via voice2 , speech seems poised to become the dominant input modality for mobile devices. This partiality to speech input is carried over to CAPTCHA solving as well. In addition, speech synthesis techniques have not matured enough to accurately replicate the human voice. Synthesized speech usually contains certain artifacts that machine learning techniques can identify with more than 90% accuracy.

While we believe speech-based CAPTCHA solutions are well-positioned to replace their text-based counterparts and address the concerns mentioned above, the lack of diverse speech copora poses a major challenge for their wide-scale deployment. Despite its promise, current speech-recognition systems cater only to a limited user-base. Due to insufficient data, popular speech-based personal assistants, such as Google Assistant and Siri, support only 8 and 13 languages respectively, out of roughly 6900 . Even for the popular languages, the available data might not be diverse enough to model a wide range of accents and dialects.

Existing Systems

CAPTCHA stands for the Completely Automated Public Turing test to tell Computers and Humans Apart. CAPTCHAs are tools you can use to differentiate between real users and automated users, such as bots. CAPTCHAs provide challenges that are difficult for computers to perform but relatively easy for humans.

Text-based CAPTCHAs are the original way in which humans were verified. These CAPTCHAs can use known words or phrases, or random combinations of digits and letters. Some text-based CAPTCHAs also include variations in capitalization.

The CAPTCHA presents these characters in a way that is alienated and requires interpretation. Alienation can involve scaling, rotation, distorting characters. It can also involve overlapping characters with graphic elements such as color, background noise, lines, arcs, or dots. This alienation provides protection against bots with insufficient text recognition algorithms but can also be difficult for humans to interpret.

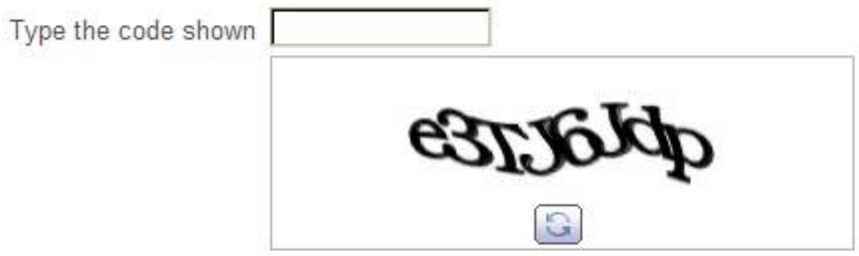


Figure 1

Problem Statement

For nearly a decade, CAPTCHAs worked pretty well, even if people hated them. Although, truth be told, even from the beginning, CAPTCHAs created friction for users. According to a 2010 research paperfc published by Stanford scientists, the average person needed 10 seconds to solve the first-generation CAPTCHA. This is an eternity in the world of online shopping, particularly on mobile devices. What's more, it's likely that recent versions of CAPTCHAs take even longer for humans to solve due to their enhanced complexity.

Over time, the original CAPTCHAs became easy to solve for bots equipped with image processing software. To counter the growing capabilities of the bots, the makers of CAPTCHA services began gradually increasing the difficulty of the challenges, ultimately escalating to visual processing challenges that are hard for real humans to solve. In 2018, the Baymard Institute, which performs UX research, estimated that users fail to solve text-based CAPTCHAs roughly 8% of the time. That bumps up to 29% if the CAPTCHA is case-sensitive. Furthermore, there are reports that claim artificial intelligence (AI) systems are better than humans at solving CAPTCHA-like tasks, which are, after all, pattern recognition tasks perfectly suited to AI.

In fact, one researcher demonstrated how with just under 400 lines of JavaScript, by using Google's own machine learning service, his tool successfully solved Google's reCAPTCHA. Google's own CAPTCHA lead told the Verge that in five to 10 years traditional CAPTCHAs will be useless against AIs.

The reality is that bot network operators can already apply a nuclear option -- they can leverage humans to solve CAPTCHAs via Amazon's Mechanical Turk and other "human intelligence task" platforms. Today, CAPTCHA solving services powered by humans are common, out in the open and even offer API access.

Introduction to proposed model

A client web-application instance can interact with vCAPTCHA via HTTP requests. The CAPTCHA Manager acts as the interface between the web application and the other components of vCAPTCHA. The web application requests a challenge from the CAPTCHA Manager, which in turn requests a challenge sentence from the Database Manager. The Database Manager picks a sentence quasirandomly from all the enrolled corpora based on a customizable set of rules. For example, the Database Manager could be instructed to favor sentences for which there are fewer responses in the database, or sentences which are easier, etc. Upon receiving the challenge sentence from the Database Manager, the CAPTCHA Interface constructs and forwards to the web-application, an HTML snippet consisting of the challenge phrase/sentence and buttons to initiate and terminate the recording of the response. We have chosen to send the challenge as a self-contained HTML snippet so that the responsibility of loading the relevant Javascript libraries does not fall on the web-application. When the user terminates the recording, the audio signal is uploaded to the CAPTCHA Manager, which passes it onto the Speech Recognizer and Forgery Detector.

Speech Recognizer and the Forgery Detectors analyze the audio and, respectively, provide scores for how closely the content of the speech matched the challenge sentence and how “human” the speech is. These scores are passed to the Validator which decides whether the user passed the CAPTCHA or not. The CAPTCHA Manager then forwards the verdict of the Validator to the web-application and, if the user passed, it hands over the audio recording to the Database Manager to store in the database along with the challenge sentence. The Categorical Sorter in the database is an optional component that can infer additional information about the audio recording, such as the accent of the speaker, and her environment.

Proposed Solution

we propose vCAPTCHA, a voice-based CAPTCHA system that would: enable more secure human authentication, more conveniently integrate with modern devices accessing web services, and help collect vast amounts of annotated speech data for different languages, accents, and dialects that are under-represented in the current speech corpora, thus making speech technologies accessible to more people around the world. vCAPTCHA requires users to speak their responses, in order to unlock or use different web services, instead of typing them. These user responses are analyzed to determine if they are indeed naturally produced, and then transcribed to ensure that they contain the challenge sentence. We build a prototype for vCAPTCHA in order to assess its performance and practicality.

Components of the model

Speech Recognizer

It is not sufficient to know only that the response was produced by a human, we also need to ensure that the response matches the challenge sentence. This additional check is required because it is entirely possible that, through trial and error or otherwise, attackers can identify certain audio recordings that are falsely verified as being genuine by vCAPTCHA and subsequently use these recordings to bypass it. We use Pocketsphinx [36], via its Python bindings, to transcribe the speech signal that we receive from the user. We realize that several factors, including, accent, background noise, and individual pronunciation styles, can introduce minor errors in legitimate responses. Therefore, we do not expect the transcription to match the challenge sentence exactly. Instead, we use the normalized Levenshtein distance between the transcript and challenge sentence as a metric of similarity. Levenshtein distance is widely used to quantify the similarity between pairs of strings. The normalized Levenshtein distance between the transcript and challenge sentence is the ratio of the number of character-level edits, i.e. insertions, deletions and substitutions, needed to convert the transcript to the challenge sentence, and the length of the challenge string.

Validator

The Validator receives Forgery and Recognition scores from the the Forgery Detector and Speech Recognizer, respectively, and uses them to determine if the user's response passed the CAPTCHA challenge. We have chosen to use empirically determined threshold values, tR and tF , for SR and SF respectively, to classify successful and unsuccessful responses.

Forgery Detector

The Forgery Detector is responsible for determining if the incoming audio signal was produced by the human or a computer. In designing the Forgery Detector we have two key metrics, security and computational cost. While securing against illegitimate access attempts is the primary function of our CAPTCHA, it would not be practical to deploy it on a global scale if it imposes enormous computational costs. Keeping this in mind, we have decided to trade a few additional percentage points of accuracy offered by complex deep learning based approaches, for the powerful yet significantly simpler Gaussian Mixture Models (GMMs) as the classifier of choice to differentiate between genuine and forged speech.

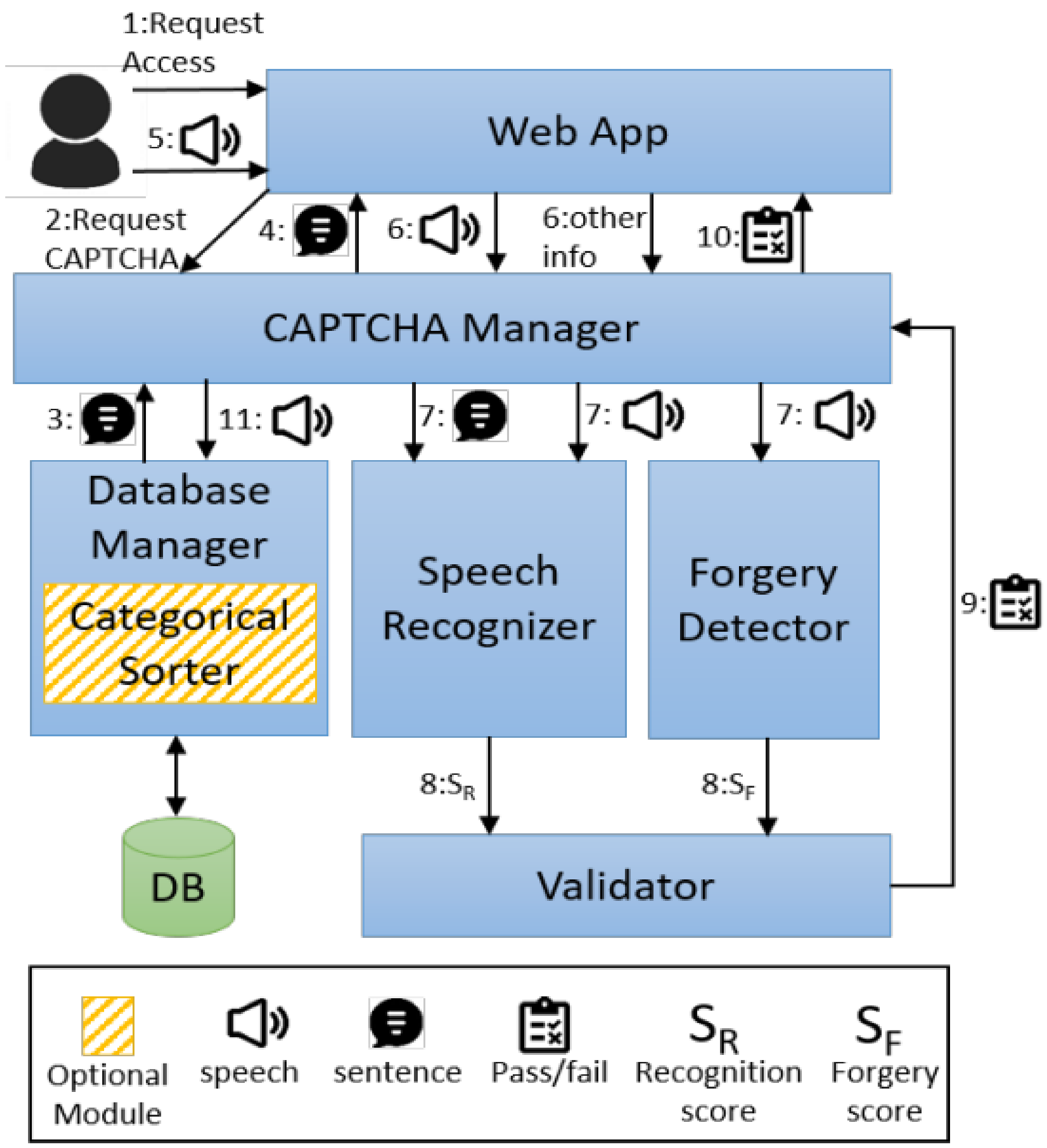


Figure 2

Code

```
import random

import string

import re

from tkinter import Tk, Label, Entry, Button, END

from PIL import ImageTk, Image

from captcha.image import ImageCaptcha

import speech_recognition as sr


def createImage(flag=0):

    global random_string

    global image_label

    global image_display

    global entry

    global verify_label



    if flag == 1:

        verify_label.grid_forget()



    entry.delete(0, END)
    random_string = ''.join(random.choices(string.digits, k=6))

    image_captcha = ImageCaptcha(width=250, height=125)
    image_generated = image_captcha.generate(random_string)
    image_display = ImageTk.PhotoImage(Image.open(image_generated))


    image_label.grid_forget()
    image_label = Label(root, image=image_display)
    image_label.grid(row=1, column=0, columnspan=2,
                    padx=10)

def listen():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Say something...")
        r.adjust_for_ambient_noise(source, duration = 1)
        audio = r.listen(source)

    try:
        print("You said: " + r.recognize_google(audio))
        check(r.recognize_google(audio), random_string)
    except sr.UnknownValueError:
        print("Google Speech Recognition could not understand audio")
    except sr.RequestError as e:
        print(
            "Could not request results from Google Speech Recognition service; {0}".format(e))




def check(x, y):

    global verify_label

    x=re.sub("\D","",x)

    verify_label.grid_forget()

    filter(lambda m: m.isdigit(), y)
    if x.lower() == y.lower():
        verify_label = Label(master=root,
                             text="Correct",
                             font="Arial 15",
                     bg='#00ff00',

                     fg="#ffffff"

                     )

        verify_label.grid(row=0, column=0, columnspan=2, pady=10)

    else:

        verify_label = Label(master=root,

                     text="Incorrect!",

                     font="Arial 15",

                     bg='#ff0000',

                     fg="#ffffff"

                     )

        verify_label.grid(row=0, column=0, columnspan=2, pady=10)

        createImage()


if __name__ == "__main__":


    root = Tk()

    root.title('Simple Captcha Generator')

    verify_label = Label(root)

    image_label = Label(root)




    entry = Entry(root, width=10, borderwidth=5,
                font="Arial 15", justify="center")
    entry.grid(row=2, column=0)

    verify_label1 = Label(master=root,
                     text="speak captch :",
                     font="Arial 15",
                     bg='#ffffff',
                     fg="#0f45f6"
                     )

    verify_label1.grid(row=3, column=0)



    createImage()



    reload_button = Button(text="Refresh", command=lambda: createImage(1))
    reload_button.grid(row=2, column=1, pady=10)
    voice_button = Button(text="Start", command=lambda: listen())
    voice_button.grid(row=3, column=1, pady=10)
```

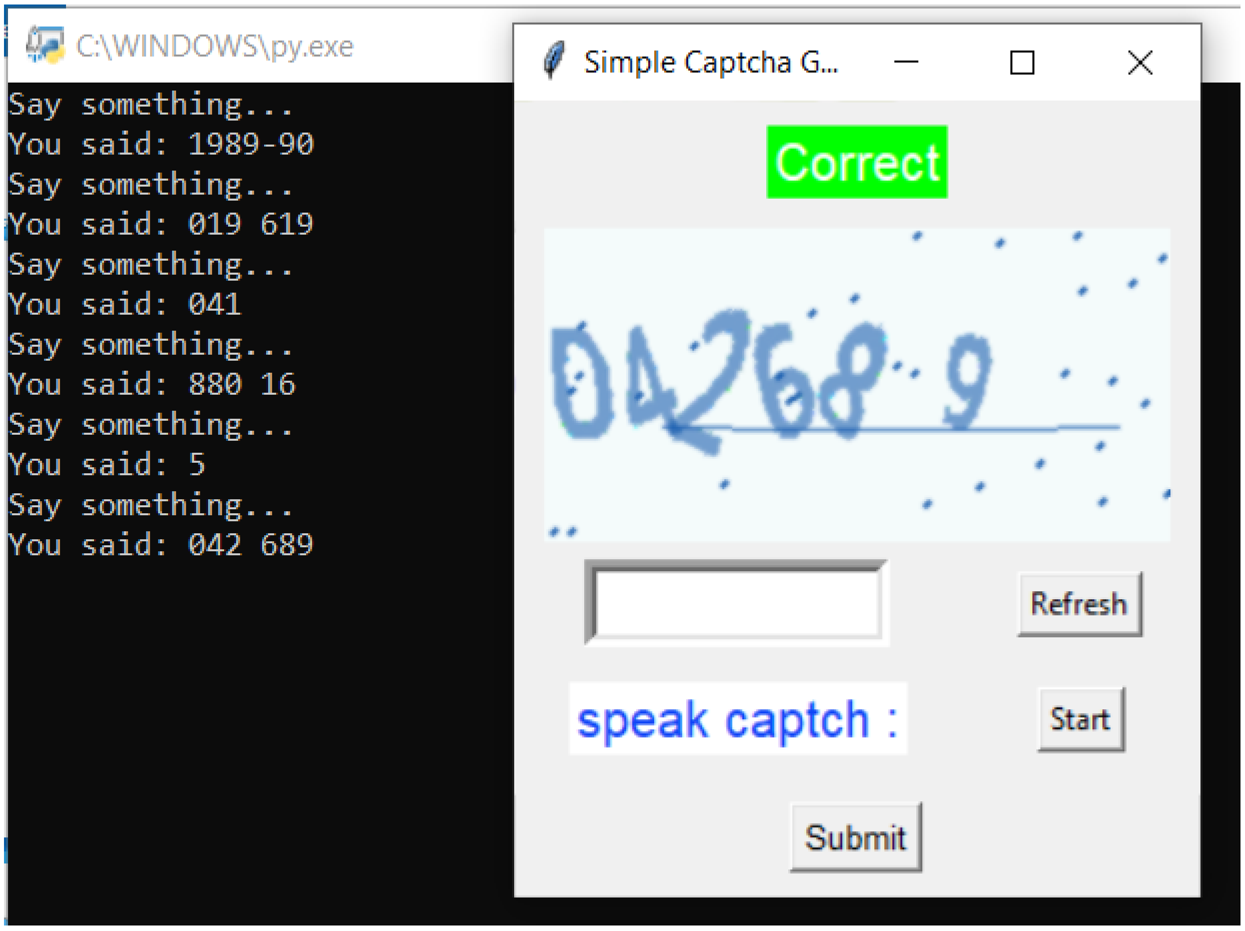
```
    submit_button = Button(root, text="Submit", font="Arial 10", command=lambda: check(entry.get(), random_string))

    submit_button.grid(row=4, column=0, columnspan=2, pady=10)

    root.bind('<Return>', func=lambda Event: check(entry.get(), random_string))


    root.mainloop()
```

Output



```
C:\WINDOWS\py.exe
Say something...
You said: 1989-90
Say something...
You said: 019 619
Say something...
You said: 041
Say something...
You said: 880 16
Say something...
You said: 5
Say something...
You said: 042 689
```

Conclusion and future work

In this paper we have proposed our speech-based vCAPTCHA system that would more effectively secure webresources from bots while minimizing the inconvenience of legitimate, human, users. We have evaluated vCAPTCHA on two recent databases, namely ASVspoof 2015 and ASVspoof 2017. vCAPTCHA achieved a high human success rate of 73.7% and an extremely low attack success rate of 10.2% on the aforementioned datasets. During the course of our evaluation we found that 512-component GMMs, which are common in past works, do not offer a significant performance upgrade over their simpler 256- component counterparts.

We also observed that SCMCs with 40-filters offer the best classification performance while the length of the audio sample has no observable effect. Moving forward, our next step would be to deploy vCAPTCHA with an actual web-service in order to gain insights about user experiences and start actually collecting data. Of course, This step would entail moving from the current prototype implementation to a fully deployable one. After deployment, we would be particularly interested in observing the quality and characteristics of data we gather and how its addition to well-known corpora would effect the performance of speech recognition systems trained on them.

REFERENCES

- https://nsl.qatar.cmu.edu/papers/icws18.pdf
- https://www.forbes.com/sites/forbestechcouncil/2019/08/07/your-captcha-could-be-hurting-your-sales/?sh=41b8999b33c8
- https://baymard.com/blog/captchas-in-checkout
- https://www.simplilearn.com/tutorials/python-tutorial/speech-recognition-in-python
- https://pypi.org/project/captcha
- https://numpy.org/doc/stable/user/whatisnumpy.html