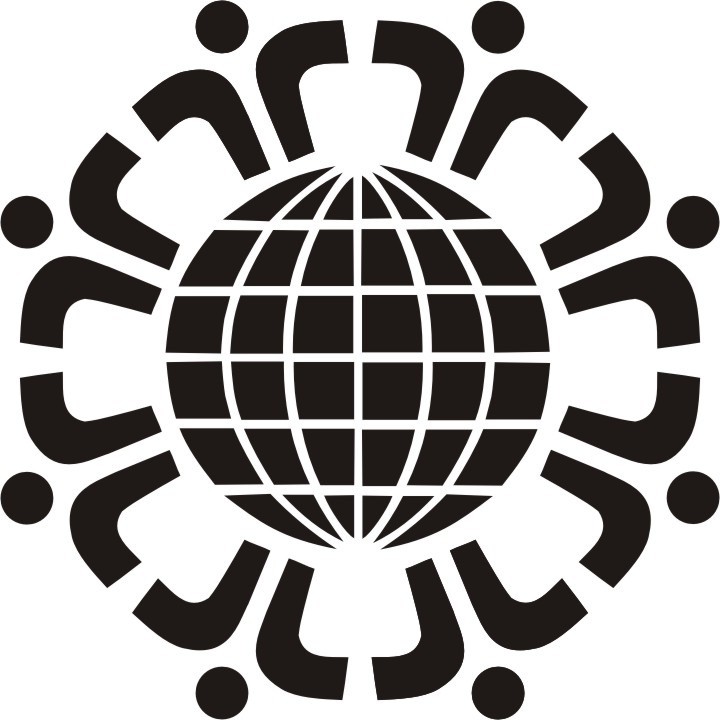
**Project Report**

**On**

**Text Emotion Detection System**

Submitted to: HOD of department

# CERTIFICATE

This is to certify that this report embodies the original work done by**, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**during this project submission as a partial fulfillment of the requirement for the System Design Project of Masters of Computer Application IV Semester, of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_University,\_\_\_\_\_\_\_.

Name\_\_\_\_\_\_\_

Sem\_\_\_\_\_\_\_\_\_

# ACKNOWLEDGEMENT

The satisfaction that accompanies that thesuccessful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

We are grateful to our project guide \_\_\_\_\_\_for the guidance, inspiration and constructive suggestions that helpful us in the preparation of this project.

We also thank our colleagues who have helped in successful completion of the project.

Name\_\_\_\_\_\_\_\_\_\_

**TABLE OF CONTENT**

1. Overview of the Organisation
2. Introduction
3. Python
4. An Introduction to Machine Learning
5. Tensorflow
6. Pandas
7. Karas
8. Code
9. Screenshorts

**CONCLUSION**

**REFERENCES**

**OVERVIEW OF THE ORGANIZATION**

****



**Web –https://vprotechdigital.com/**

**E-Mail** [– Vpotechdigital@gmail.com](mailto:–%20Vpotechdigital@gmail.com)

**No. 1 Software & Development Company in Chandigarh**

**Born in 2016, VproTech digital is a forward looking company focused on software, solution and innovative ideas that are required for 21st century. As an institution being a pioneer in courses that is gave birth to , vprotech digital offers different job practices oriented trainings from different streams including mechanical, civil, electronics and computer science engineering with placement tie ups all over the country.**

**Under the 9001:2015 ISO Certification, we offer you standardize trainings and personality development sessions that helps buildingmindset and vision of the students for their bright future.**

**Vprotech digital**

SCF – 116, SECOND FLOOR, INDUSTRIAL AREA SECTOR 58, PhASE-5, MOHALI, 160059

E-Mail – Vprotechdigital@gmail.com

WEB –**https://vprotechdigital.com/**

Contact 172-4639508, 79019494

**Intoduction**

Social media has become an integral part of the people in 21st century. Due to rapid progress in Information &Technology sector, people have access to any kind of information at the click of a button. Moreover, with the invent of smart-phones and 4G networks, even people from the remote areas are getting connected to Tier 1 and Tier 2 cities. With the growing population in countries like India, it has led to tremendous growth in the number of people using social networks.

Social networks like Facebook, WhatsApp, Twitter, etc.

has eliminated the gap between lives of people. One of the

reasons to use these social networks to know the current

happenings around them and to express their views and

suggestions in the form of likes, share, tweets, polls, email,

etc. This has created a new category of people called

netizens. Communication via social media is done in the

form of text, image, audio and video which contains

information and consumes space, memory and Internet

bandwidth. All these activities done on social media has

resulted into vast amount of information being generated on

a daily basis. Social media analysis has become a interesting

field of research to understand the behavior and thoughts of

people in response to social, economic, cultural, educational

and all others activities happening around the world .

Social media data is in the form of unstructured data since

people are from diverse backgrounds with different race,

culture, language and standard of living project their ideas,

views, opinions, expressions and so on the Internet. So, it has

become a challenge in recent years to extract valuable

information from these ever-growing data in the form of

posts, emails, blogs, micro-blogs, tweets, reviews,

comments, polls and surveys on the Web about an individual,

an organization or government in the process of decision-

making . These opinionated data not only exist on the

Web but also within the large organizations like Google,

Microsoft, Hewlett-Packard, SAP and SAS to know the

opinions of their employees spread across the continents in

the form of customer feedback collected from emails and call

centers or results from surveys conducted by organizations.

This has created strong interest for research in sentiment

analysis.

Sentiment Analysis has emerged as a popular solution to

gain deeper insights such as knowing how happy our people

are, predicting box-office revenues from movie reviews,

predicting election results and stock market from Twitter

data, gaining feedback before releasing a new product and

visualizing of customers feel about product or service run by

multinational companies. From all these sources, Twitter has

become one of the prominent leads in gathering and

expressing users opinions publicly within a short message.

With 140-character limit, people have freedom to express

their feelings, opinions, appraisal, emotions, attitudes,

evaluations and sentiments on present or past events, trends,

lifestyles, government, health care, education, business,

politics and all kinds of activities around the world. Twitter

statistics report states that on every second on an average

around 6,000 tweets are tweeted on Twitter which results to

over 350,000 tweets sent per minute, 500 million tweets per

day and around 200 billion tweets per year.

In this paper, emotion mining on data over Twitter is

analyzed by using Machine Learning techniques on recent

trends, topics, discussions over the Web to determine the

effect of the textual information on the mentality of the

people by identifying their emotions. This paper is organized

in following sections. Related work is discussed in Section

II. The proposed system details are provided in Section III

followed by experiments and results in Section IV. The

conclusion of this paper is provided in Section V with future

work direction

**Python**

Python is an interpreted and open source high-level programming language. Python was createdby **Guido Van Rossum and** released in **1991**. It is support multiple Platforms like **Linux**, **Mac**, **Windows**. It is easy than other programming language, it is

Just like to English language.

### Python = Simplicity

We cannot stress this point enough, but Python is not only easy to learn but also easy to use and implement. With a syntax similar to English, you can master the nitty-gritty of Python coding in a few days. Moreover, Python is dynamically-typed, which makes indentation mandatory, thereby enhancing its readability factor.

### It is an open-source language

You don’t need to pay charges to install and use Python – it is open-source. What this means is that the source code of Python is freely available to the public. You can download it from Python’s official website. Not only that, Python supports the **FLOSS (Free/Libre and Open Source Software)** model, which means you can also change it and distribute it. This allows the Python community to tweak it and improve its features continuously.

### It is a high-level language

Since Python is a high-level language, you need not remember its system architecture, not do you need to perform memory management. This feature contributes to Python’s user-friendliness.

### It is interpreted

Unlike compiled languages like C++ and Jave wherein you must compile the code and then run it, Python is an interpreted language. What this means is that instead of executing the source code all at once, Python executes it line by line. This makes it easier to debug a Python code because you can do it while writing the code.

### It is both object-oriented and functional

An object-oriented programming language is one that can model real-world data, while a functional language focuses on functions (code that can be reused). Python supports both object-oriented and functional programming features. Also, unlike Java, Python supports multiple inheritances.

### It is portable

Python is portable and highly flexible, meaning, a Python code written for a Windows machine or a Linux machine can also run on iOS, and vice versa – you don’t need to make any alterations in the code. So, with Python eliminates the need to write different code for different machines (just make sure there’s no system-dependent feature in your Python code).

### It is extensible and embeddable

Python is an extensible language, as it allows you to write specific parts of your Python code in other programming languages such as C++. Similarly, you can also embed your Python code in the source code of other languages. This allows you to integrate Python’s scripting functionalities into a code written in another language.

### It comes with a vast collection of libraries

When you download Python, you will automatically download the extensive collection of Python libraries with it. These libraries are built-in, so you don’t have to write individual code for every single thing. Python has libraries and packages for web browsers, threading, databases, regular expressions, image manipulation, documentation-generation, unit-testing, CGI, email, and much more.

Now that we’ve talked at length about how great a tool Python is let’s check out **twelve real-world applications of Python.**

## **12 Real-world Applications of Python**

### Web Development

When it comes to web development, Python should be your go-to tool. Why?

That’s because Python offers numerous options for web development. For instance, you have Django, Pyramid, Flask, and Bottle for developing web frameworks and even advanced content management systems like Plone and Django CMS. These web frameworks are packed with standard libraries and modules which simplify tasks like content management, database interaction, and interfacing with internet protocols like HTTP, SMTP, XML, JSON, FTP, IMAP, and POP.

Python web frameworks are known for their security, scalability, and flexibility. To add to that, Python’s Package Index comes with useful libraries like Requests, BeautifulSoup, Paramiko, Feedparser, and Twisted Python.

### Game Development

As we mentioned earlier, Python comes loaded with many useful extensions (libraries) that come in handy for the development of interactive games. For instance, libraries like PySoy (a 3D game engine that supports Python 3) and PyGame are two Python-based **libraries used widely for game development**. Python is the foundation for popular games like Battlefield 2, Frets on Fire, World of Tanks, Disney’s Toontown Online, Vega Strike, and Civilization-IV.

Apart from game development, game designers can also use Python for developing tools to simplify specific actions such as level design or dialog tree creation, and even use those tools to export those tasks in formats that can be used by the primary game engine. Also, Python is used as a scripting language by many game engines.

### Scientific and Numeric Applications

Thanks to its massive library base, Python has become a crucial tool in scientific and numeric computing. In fact, Python provides the skeleton for applications that deal with computation and scientific data processing. Apps like FreeCAD (3D modeling software) and Abaqus (finite element method software) are coded in Python.

Some of the most useful Python packages for scientific and numeric computation include:

* SciPy (scientific numeric library)
* Pandas (data analytics library)
* IPython (command shell)
* Numeric Python (fundamental numeric package)
* Natural Language Toolkit (Mathematical And text analysis)

### Artificial Intelligence and Machine Learning

AI and ML models and projects are inherently different from traditional software models. When we talk about AI/ML projects, the tools and technologies used and the skillset required is totally different from those used in the development of conventional software projects. AI/ML applications require a language that is stable, secure, flexible, and is equipped with tools that can handle the various unique requirements of such projects. Python has all these qualities, and hence, it has become one of the most favored languages of Data Science professionals.

Python’s simplicity, consistency, platform independence, great collection of resourceful libraries, and an active community make it the perfect tool for developing AI and ML applications. Some of the best Python packages for AI and ML are:

* SciPy for advanced computing
* Pandas for general-purpose data analysis
* Seaborn for data visualization
* Keras, TensorFlow, and Scikit-learn for ML
* NumPy for high-performance scientific computing and data analysis

 Apart from these libraries, there are also other Python-based libraries like NLTK, Caffee, PyTorch, and Accord. NET, that are useful for AI and ML projects.

### Desktop GUI

Python not only boasts of an English-like syntax, but it also features a modular architecture and the ability to work on multiple operating systems. These aspects, combined with its rich text processing tools, make Python an excellent choice for developing desktop-based GUI applications.

Python offers many GUI toolkits and frameworks that make desktop application development a breeze. PyQt, PyGtk, Kivy, Tkinter, WxPython, PyGUI, and PySide are some of the best Python-based GUI frameworks that allow developers to create highly functional Graphical User Interfaces (GUIs).

### Software Development

Python packages and applications aim to simplify the process of software development. From developing complex applications that involve scientific and numeric computing to developing desktop and web applications, Python can do it all. This is the reason why **Software Developers use Python as a support language** for build control, testing, and management.

For instance, SCons is designed explicitly for build control, Buildbot and Apache Gump allow for automated continuous compilation and testing, and Roundup and Trac are great for bug tracking and project management.

Python also supports data analyzation and visualization, thereby further simplifying the process of creating custom solutions minus the extra effort and time investment.

### Enterprise-level/Business Applications

Enterprise-level software or business applications are strikingly different from standard applications, as in the former demands features like readability, extensibility, and scalability. Essentially, business applications are designed to fit the requirements of an organization rather than the needs of individual customers.

Thus, these applications must be capable of integrating with legacy systems like existing databases and non-web apps. Since business applications are developed, keeping in mind the custom requirements to cater to the specific needs of an organization’s operating model, the entire development process becomes very complicated.

This is where Python can make a significant difference. Python high performance, scalability, flexibility, and readability are just the features required for developing fully-functional and efficient business applications. Furthermore, Python has other tools for business application development, like:

* Odoo, an all-in-one management software that forms a complete suite of enterprise management applications.
* Tryton, a three-tier, high-level, general-purpose application platform, is another fantastic tool for building business applications.

### Education programs and training courses

If there’s any beginner-friendly programming language, it is Python. We’ve said it many times before, and we’re repeating it – Python has an extremely straightforward syntax that’s similar to the English language. It has a short learning curve and hence, is an excellent choice for beginners. Python’s easy learning curve and simplicity are the two main reasons why it is one of the most used programming languages in educational programs, both at beginner and advanced levels.

However, Python is not just great as an introductory language – even professional developers and coders all around the world rely heavily on Python.

### Language Development

Over the years, Python’s design and module architecture has been the inspiration behind the development of many new programming languages such as Boo, Swift, CoffeeScript, Cobra, and OCaml. All of these languages share numerous similarities with Python on grounds like object model, syntax, and indentation.

### Operating Systems

Yes, Python is the secret ingredient behind many operating systems as well, most popularly of Linux distributions. Linux-based Ubuntu’s Ubiquity Installer and Fedora and Red Hat Enterprise’s Anaconda Installer are coded in Python. Even Gentoo Linux leverages Python Portage (package management system). Usually, Python is combined with the C programming language to design and develop operating systems.

### Web Scraping Applications

Python is a nifty tool for extracting voluminous amounts of data from websites and web pages. The pulled data is generally used in different real-world processes, including job listings, price comparison, R&D, etc.

BeautifulSoup, MechanicalSoup, Scrapy, LXML, Python Requests, Selenium, and Urllib are some of the best Python-based web scraping tools.

### Image Processing and Graphic Design Applications:

Alongside all the uses mentioned above, Python also finds a unique use case in image processing and graphic design applications. The programming language is used globally to design and build 2D imaging software like Inkscape, GIMP, Paint Shop Pro, and Scribus. Also, Python is used in several 3D animation packages such as Blender, Houdini, 3ds Max, Maya, Cinema 4D, and Lightwave, to name a few.

**Tkinter Python Library**

Tkinter ("Tk Interface")is python's standard cross-platform package for creating graphical user interfaces (GUIs). It provides access to an underlying Tcl interpreter with the Tk toolkit, which itself is a cross-platform, multilanguage graphical user interface library.Tkinter isn't the only GUI library for python, but it is the one that comes standard. Additional GUI libraries that can be used with python include wxPython, PyQt, and kivy.Tkinter's greatest strength is its ubiquity and simplicity. It works out of the box on most platforms (linux, OSX, Windows), and comes complete with a wide range of widgets necessary for most common tasks (buttons, labels, drawing canvas, multiline text, etc).As a learning tool, tkinter has some features that are unique among GUI toolkits, such as named fonts, bind tags, and variable tracing.

Tkinter gives you the ability to create Windows with widgets in them

* Definition: widget is a graphical component on the screen (button, text label, drop-

down menu, scroll bar, picture, etc...)

* GUIs are built by arranging and combining different widgets on the screen.
* Example
* #!/usr/bin/python
* import tkinter
* top = tkinter.Tk()
* # Code to add widgets will go here...
* top.mainloop()
* This would create a following window −
* 

**Datetime Python Library**

This module provides various time-related functions. For related functionality, see also the [datetime](https://docs.python.org/3/library/datetime.html#module-datetime) and [calendar](https://docs.python.org/3/library/calendar.html#module-calendar) modules.

Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name. It may sometimes be helpful to consult the platform documentation, because the semantics of these functions varies among platforms.

An explanation of some terminology and conventions is in order.

* The epoch is the point where the time starts, and is platform dependent. For Unix, the epoch is January 1, 1970, 00:00:00 (UTC). To find out what the epoch is on a given platform, look at time.gmtime(0).
* The term seconds since the epoch refers to the total number of elapsed seconds since the epoch, typically excluding [leap seconds](https://en.wikipedia.org/wiki/Leap_second). Leap seconds are excluded from this total on all POSIX-compliant platforms.
* The functions in this module may not handle dates and times before the epoch or far in the future. The cut-off point in the future is determined by the C library; for 32-bit systems, it is typically in 2038.
* Function [strptime()](https://docs.python.org/3/library/time.html#time.strptime) can parse 2-digit years when given %y format code. When 2-digit years are parsed, they are converted according to the POSIX and ISO C standards: values 69–99 are mapped to 1969–1999, and values 0–68 are mapped to 2000–2068.
* UTC is Coordinated Universal Time (formerly known as Greenwich Mean Time, or GMT). The acronym UTC is not a mistake but a compromise between English and French.
* DST is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.
* The precision of the various real-time functions may be less than suggested by the units in which their value or argument is expressed. E.g. on most Unix systems, the clock “ticks” only 50 or 100 times a second.
* On the other hand, the precision of [time()](https://docs.python.org/3/library/time.html#time.time) and [sleep()](https://docs.python.org/3/library/time.html#time.sleep) is better than their Unix equivalents: times are expressed as floating point numbers, [time()](https://docs.python.org/3/library/time.html#time.time) returns the most accurate time available (using Unix gettimeofday() where available), and [sleep()](https://docs.python.org/3/library/time.html#time.sleep) will accept a time with a nonzero fraction (Unix select() is used to implement this, where available).
* The time value as returned by [gmtime()](https://docs.python.org/3/library/time.html#time.gmtime), [localtime()](https://docs.python.org/3/library/time.html#time.localtime), and [strptime()](https://docs.python.org/3/library/time.html#time.strptime), and accepted by [asctime()](https://docs.python.org/3/library/time.html#time.asctime), [mktime()](https://docs.python.org/3/library/time.html#time.mktime) and [strftime()](https://docs.python.org/3/library/time.html#time.strftime), is a sequence of 9 integers. The return values of [gmtime()](https://docs.python.org/3/library/time.html#time.gmtime), [localtime()](https://docs.python.org/3/library/time.html#time.localtime), and [strptime()](https://docs.python.org/3/library/time.html#time.strptime) also offer attribute names for individual fields.
* See [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) for a description of these objects.
* Changed in version 3.3: The [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) type was extended to provide the tm\_gmtoff and tm\_zone attributes when platform supports corresponding structtm members.
* Changed in version 3.6: The [struct\_time](https://docs.python.org/3/library/time.html#time.struct_time) attributes tm\_gmtoff and tm\_zoneare now available on all platforms.

**An Introduction to Machine Learning**

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people. Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs. Any technology user today has benefitted from machine learning. Facial recognition technology allows social media platforms to help users tag and share photos of friends. Optical character recognition (OCR) technology converts images of text into movable type. Recommendation engines, powered by machine learning, suggest what movies or television shows to watch next based on user preferences. Self-driving cars that rely on machine learning to navigate may soon be available to consumers. Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies, or analyze the impact of machine learning processes. In this we’ll look into the common machine learning methods of supervised and unsupervised learning, and common algorithmic approaches in machine learning, including the k-nearest neighbor algorithm, decision tree learning, and deep learning. We’ll explore which programming languages are most used in machine learning, providing y o u with some of the positive and negative attributes of each.

Additionally, we’ll discuss biases that are perpetuated by machine learning algorithms, and consider what can be kept in mind to prevent these biases when building algorithms. Machine Learning Methods In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed. Two of the most widely adopted machine learning methods are supervised learning which trains algorithms based on example input and output data that is labeled by humans, and unsupervised learning which provides the algorithm with no labeled data in order to allow it to find structure within its input data. Let’s explore these methods in more detail. Supervised Learning In supervised learning, the computer is provided with example inputs that are labeled with their desired outputs. The purpose of this method is for the algorithm to be able to “learn” by comparing its actual output with the “taught” outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on additional unlabeled data. For example, with supervised learning, an algorithm may be fed data with images of sharks labeled as fish and images of oceans labeled as water. By being trained on this data, the supervised learning algorithm should be able to later identify unlabeled shark images as fish and unlabeled ocean images as water. A common use case of supervised learning is to use historical data to predict statistically likely future events. It may use historical stock market information to anticipate upcoming fluctuations, or be employed to filter out spam emails. In supervised learning, tagged photos of dogs can be used as input data to classify untagged photos of dogs. Unsupervised Learning In unsupervised learning, data is unlabeled, so the learning algorithm is left to find commonalities among its input data. As unlabeled data are more abundant than labeled data, machine learning methods that facilitate unsupervised learning are particularly valuable. The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data. Unsupervised learning is commonly used for transactional data. You may have a large dataset of customers and their purchases, but as a human you will likely not be able to make sense of what similar attributes can be drawn from customer profiles and their types of purchases.

With this data fed into an unsupervised learning algorithm, it may be determined that women of a certain age range who buy unscented soaps are likely to be pregnant, and therefore a marketing campaign related to pregnancy and baby products can be targeted to this audience in order to increase their number of purchases. Without being told a “correct” answer, unsupervised learning methods can look at complex data that is more expansive and seemingly unrelated in order to organize it in potentially meaningful ways. Unsupervised learning is often used for anomaly detection including for fraudulent credit card purchases, and recommender systems that recommend what products to buy next. In unsupervised learning, untagged photos of dogs can be used as input data for the algorithm to find likenesses and classify dog photos together. Approaches As a field, machine learning is closely related to computational statistics, so having a background knowledge in statistics is useful for understanding and leveraging machine learning algorithms. For those who may not have studied statistics, it can be helpful to first define correlation and regression, as they are commonly used techniques for investigating the relationship among quantitative variables. Correlation is a measure of association between two variables that are not designated as either dependent or independent. Regression at a basic level is used to examine the relationship between one dependent and one independent variable. Because regression statistics can be used to anticipate the dependent variable when the independent variable is known, regression enables prediction capabilities. Approaches to machine learning are continuously being developed. For our purposes, we’ll go through a few of the popular approaches that are being used in machine learning at the time of writing. k-nearest neighbor The k-nearest neighbor algorithm is a pattern recognition model that can be used for classification as well as regression. Often abbreviated as kNN, the k in k-nearest neighbor is a positive integer, which is typically small. In either classification or regression, the input will consist of the k closest training examples within a space. We will focus on k-NN classification. In this method, the output is class membership. This will assign a new object to the class most common among its k nearest neighbors. In the case of k = 1, the object is assigned to the class of the single nearest neighbor. Let’s look at an example of k-nearest neighbor. In the diagram below, there are blue diamond objects and orange star objects. These belong to two separate classes: the diamond class and the star class. k-nearest neighbor initial data set When a new object is added to the space — in this case a green heart — we will want the machine learning algorithm to classify the heart to a certain class. k-nearest neighbor data set with new object to classify When we choose k = 3, the algorithm will find the three nearest neighbors of the green heart in order to classify it to either the diamond class or the star class. In our diagram, the three nearest neighbors of the green heart are one diamond and two stars. Therefore, the algorithm will classify the heart with the star class. k-nearest neighbor data set with classification complete Among the most basic of machine learning algorithms, k-nearest neighbor is considered to be a type of “lazy learning” as generalization beyond the training data does not occur until a query is made to the system. Decision Tree Learning For general use, decision trees are employed to visually represent decisions and show or inform decision making. When working with machine learning and data mining, decision trees are used as a predictive model. These models map observations about data to conclusions about the data’s target value. The goal of decision tree learning is to create a model that will predict the value of a target based on input variables. In the predictive model, the data’s attributes that are determined through observation are represented by the branches, while the conclusions about the data’s target value are represented in the leaves. When “learning” a tree, the source data is divided into subsets based on an attribute value test, which is repeated on each of the derived subsets recursively. Once the subset at a node has the equivalent value as its target value has, the recursion process will be complete. Let’s look at an example of various conditions that can determine whether or not someone should go fishing. This includes weather conditions as well as barometric pressure conditions. fishing decision tree example In the simplified decision tree above, an example is classified by sorting it through the tree to the appropriate leaf node. This then returns the classification associated with the particular leaf, which in this case is either a Yes or a No. The tree classifies a day’s conditions based on whether or not it is suitable for going fishing. A true classification tree data set would have a lot more features than what is outlined above, but relationships should be straightforward to determine. When working with decision tree learning, several determinations need to be made, including what features to choose, what conditions to use for splitting, and understanding when the decision tree has reached a clear ending. Deep Learning Deep learning attempts to imitate how the human brain can process light and sound stimuli into vision and hearing. A deep learning architecture is inspired by biological neural networks and consists of multiple layers in an artificial neural network made up of hardware and GPUs. Deep learning uses a cascade of nonlinear processing unit layers in order to extract or transform features (or representations) of the data. The output of one layer serves as the input of the successive layer. In deep learning, algorithms can be either supervised and serve to classify data, or unsupervised and perform pattern analysis. Among the machine learning algorithms that are currently being used and developed, deep learning absorbs the most data and has been able to beat humans in some cognitive tasks. Because of these attributes, deep learning has become the approach with significant potential in the artificial intelligence space Computer vision and speech recognition have both realized significant advances from deep learning approaches. IBM Watson is a well-known example of a system that leverages deep learning.

**How To Build a Machine Learning Classifier in Python with Scikit-learn**

Implement a simple machine learning algorithm in Python using Scikit-learn, a machine learning tool for Python. Using a database of breast cancer tumor information, you’ll use a Naive Bayes (NB) classifier that predicts whether or not a tumor is malignant or benign. By the end of this tutorial, you’ll know how to build your very own machine learning model in Python. Prerequisites To complete this tutorial, we’ll use Jupyter Notebooks, which are a useful and interactive way to run machine learning experiments. With Jupyter Notebooks, you can run short blocks of code and see the results quickly, making it easy to test and debug your code. To get up and running quickly, you can open up a web browser and navigate to the Try Jupyter website: jupyter.org/try. From there, click on Try Jupyter with Python, and you will be taken to an interactive Jupyter Notebook where you can start to write Python code.

**How To Set Up Jupyter Notebook for Python 3**.

Step 1 — Importing Scikit-learn

Let’s begin by installing the Python module Scikit-learn, one of the best and most documented machine learning libraries for Python. To begin our coding project, let’s activate our Python 3 programming environment.

Make sure you’re in the directory where your environment is located, and run the following command: . my\_env/bin/activate

With our programming environment activated, check to see if the Sckikit-learn module is already installed: (my\_env) $ python -c "import sklearn"

If sklearn is installed, this command will complete with no error. If it is not installed, you will see the following error message: Output Traceback (most recent call last): File "", line 1, in ImportError: No module named 'sklearn'

The error message indicates that sklearn is not installed, so download the library using pip: (my\_env) $ pip install scikit-learn[alldeps] Once the installation completes, launch Jupyter Notebook: (my\_env) $ jupyter notebook

In Jupyter, create a new Python Notebook called ML Tutorial. In the first cell of the Notebook, import the sklearn module:

Jupyter Notebook with one Python cell, which imports sklearn

Now that we have sklearn imported in our notebook, we can begin working with the dataset for our machine learning model.

**Step 2 — Importing Scikit-learn’s Dataset**

The dataset we will be working with in this tutorial is the Breast Cancer Wisconsin Diagnostic Database. The dataset includes various information about breast cancer tumors, as well as classification labels of malignant or benign. The dataset has 5 6 9 instances, or data, on 569 tumors and includes information on 30 attributes, or features, such as the radius of the tumor, texture, smoothness, and area.

Using this dataset, we will build a machine learning model to use tumor information to predict whether or not a tumor is malignant or benign. Scikit-learn comes installed with various datasets which we can load into Python, and the dataset we want is included.

Import and load the dataset: ML Tutorial ... from sklearn.datasets import load\_breast\_cancer # Load dataset data = load\_breast\_cancer() T h e data variable represents a Python object that works like a dictionary.

The important dictionary keys to consider are the classification label names (target\_names), the actual labels (target), the attribute/feature names (feature\_names), and the attributes (data). Attributes are a critical part of any classifier.

Attributes capture important characteristics about the nature of the data. Given the label we are trying to predict (malignant versus benign tumor), possible useful attributes include the size, radius, and texture of the tumor.

Create new variables for each important set of information and assign the data: ML Tutorial ... # Organize our data label\_names = data['target\_names'] labels = data['target'] feature\_names = data['feature\_names'] features = data['data'] We now have lists for each set of information.

To get a better understanding of our dataset, let’s take a look at our data by printing our class labels, the first data instance’s label, our feature names, and the feature values for the first data instance: ML Tutorial ... # Look at our data print(label\_names) print(labels[0]) print(feature\_names[0]) print(features[0])

**How To Build a Neural Network to Recognize Handwritten Digits with TensorFlow**

Neural networks are used as a method of deep learning, one of the many subfields of artificial intelligence. They were first proposed around 70 years ago as an attempt at simulating the way the human brain works, though in a much more simplified form. Individual ‘neurons’ are connected in layers, with weights assigned to determine how the neuron responds when signals are propagated through the network. Previously, neural networks were limited in the number of neurons they were able to simulate, and therefore the complexity of learning they could achieve. But in recent years, due to advancements in hardware development, we have been able to build very deep networks, and train them on enormous datasets to achieve breakthroughs in machine intelligence. These breakthroughs have allowed machines to match and exceed the capabilities of humans at performing certain tasks. One such task is object recognition. Though machines have historically been unable to match human vision, recent advances in deep learning have made it possible to build neural networks which can recognize objects, faces, text, and even emotions. In this tutorial, you will implement a small subsection of object recognition—digit recognition. Using TensorFlow (https://www.tensorflow.org/), an open-source Python library developed by the Google Brain labs for deep learning research, you will take hand-drawn images of the numbers 0-9 and build and train a neural network to recognize and predict the correct label for the digit displayed. While you won’t need prior experience in practical deep learning or TensorFlow to follow along with this tutorial, we’ll assume some familiarity with machine learning terms and concepts such as training and testing, features and labels, optimization, and evaluation. Prerequisites To complete this tutorial, you’ll need a local or remote Python 3 development environment that includes pip for installing Python packages, and venv for creating virtual environments.

Step 1 — Configuring the Project Before you can develop the recognition program, you’ll need to install a few dependencies and create a workspace to hold your files. We’ll use a Python 3 virtual environment to manage our project’s dependencies. Create a new directory for your project and navigate to the new directory: mkdir tensorflow-demo cd tensorflow-demo Execute the following commands to set up the virtual environment for this tutorial: python3 -m venv tensorflow-demo source tensorflow-demo/bin/activate

Next, install the libraries you’ll use in this tutorial. We’ll use specific versions of these libraries by creating a requirements.txt file in the project directory which specifies the requirement and the version we need.

Create the requirements.txt file: (tensorflow-demo) $ touch requirements.txt

Open the file in your text editor and add the following lines to specify the Image, NumPy, and TensorFlow libraries and their versions: requirements.txt image==1.5.20 numpy==1.14.3 tensorflow==1.4.0

Save the file and exit the editor.

Then install these libraries with the following command: (tensorflow-demo) $ pip install -r requirements.txt

With the dependencies installed, we can start working on our project.

Step 2 — Importing the MNIST Dataset The dataset we will be using is called the MNIST dataset, and it is a classic in the machine learning community. This dataset is made up of images of handwritten digits, 28x28 pixels in size.

Here are some examples of the digits included in the dataset: Examples of MNIST images Let’s create a Python program to work with this dataset. We will use one file for all of our work in this tutorial.

Create a new file called main.py: (tensorflow-demo) $ touch main.py

Now open this file in your text editor of choice and add this line of code to the file

To import the TensorFlow library: main.py import tensorflow as tf

Add the following lines of code to your file to import the MNIST dataset and store the image data in the variable mnist: main.py ... from tensorflow.examples.tutorials.mnist

Import input\_datamnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True) # y labels

are oh-encoded When reading in the data, we are using one-hot-encoding to represent the labels (the actual digit drawn, e.g. “3”) of the images. One-hotencoding uses a vector of binary values to represent numeric or categorical values. As our labels are for the digits 0-9, the vector contains ten values, one for each possible digit. One of these values is set to 1, to represent the digit at that index of the vector, and the rest are set to 0.

For example, the digit 3 is represented using the vector [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]. As the value at index 3 is stored as 1, the vector therefore represents the digit 3. To represent the actual images themselves, the 28x28 pixels are flattened into a 1D vector which is 784 pixels in size. Each of the 784 pixels making up the image is stored as a value between 0 and 255.

This determines the grayscale of the pixel, as our images are presented in black and white only. So a black pixel is represented by 255, and a white pixel by 0, with the various shades of gray somewhere in between. We can use the mnist variable to find out the size of the dataset we have just imported. Looking at the num\_examples for each of the three subsets, we can determine that the dataset has been split into 55,000 images for training, 5000 for validation, and 10,000 for testing.

Add the following lines to your file: main.py ... n\_train = mnist.train.num\_examples # 55,000 n\_validation = mnist.validation.num\_examples # 5000 n\_test = mnist.test.num\_examples # 10,000

Now that we have our data imported, it’s time to think about the neural network.

**Step 3 — Defining the Neural Network Architecture**

The architecture of the neural network refers to elements such as the number of layers in the network, the number of units in each layer, and how the units are connected between layers. As neural networks are loosely inspired by the workings of the human brain, here the term unit is used to represent what we would biologically think of as a neuron. Like neurons passing signals around the brain, units take some values from previous units as input, perform a computation, and then pass on the new value as output to other units. These units are layered to form the network, starting at a minimum with one layer for inputting values, and one layer to output values. The term hidden layer is used for all of the layers in between the input and output layers, i.e. those “hidden” from the real world. Different architectures can yield dramatically different results, as the performance can be thought of as a function of the architecture among other things, such as the parameters, the data, and the duration of training. Add the following lines of code to your file to store the number of units per layer in global variables.

This allows us to alter the network architecture in one place, and at the end of the tutorial you can test for yourself how different numbers of layers and units will impact the results of our model: main.py ...

n\_input = 784

# input layer (28x28 pixels) n\_hidden1 = 512

# 1st hidden layer n\_hidden2 = 256

# 2nd hidden layer n\_hidden3 = 128 # 3rd hidden layer n\_output = 10

# output layer (0-9 digits)

With each layer fully connected to the surrounding layers: Diagram of a neural network The term “deep neural network” relates to the number of hidden layers, with “shallow” usually meaning just one hidden layer, and “deep” referring to multiple hidden layers.

Given enough training data, a shallow neural network with a sufficient number of units should theoretically be able to represent any function that a deep neural network can. But it is often more computationally efficient to use a smaller deep neural network to achieve the same task that would require a shallow network with exponentially more hidden units. Shallow neural networks also often encounter overfitting, where the network essentially memorizes the training data that it has seen, and is not able to generalize the knowledge to new data. This is why deep neural networks are more commonly used: the multiple layers between the raw input data and the output label allow the network to learn features at various levels of abstraction, making the network itself better able to generalize. Other elements of the neural network that need to be defined here are the hyperparameters. Unlike the parameters that will get updated during training, these values are set initially and remain constant throughout the process. In your file, set the following variables and values:

main.py ...

learning\_rate = 1e-4

n\_iterations = 1000

batch\_size = 128

dropout = 0.5

The learning rate represents how much the parameters will adjust at each step of the learning process. These adjustments are a key component of training: after each pass through the network we tune the weights slightly to try and reduce the loss. Larger learning rates can converge faster, but also have the potential to overshoot the optimal values as they are updated. The number of iterations refers to how many times we go through the training step, and the batch size refers to how many training examples we are using at each step.

The dropout variable represents a threshold at which we eliminate some units at random. We will be using dropout in our final hidden layer to give each unit a 50% chance of being eliminated at every training step. This helps prevent overfitting. We have now defined the architecture of our neural network, and the hyperparameters that impact the learning process. The next step is to build the network as a TensorFlow graph.

**Step 4 — Building the TensorFlow Graph To build our network, we will set up the network as a computational graph for TensorFlow to execute.**

The core concept of TensorFlow is the tensor, a data structure similar to an array or list. initialized, manipulated as they are passed through the graph, and updated through the learning process. We’ll start by defining three tensors as placeholders, which are tensors that we’ll feed values into later. Add the following to your file: main.py

... X = tf.placeholder("float", [None, n\_input])

Y = tf.placeholder("float", [None, n\_output])

keep\_prob = tf.placeholder(tf.float32)

The only parameter that needs to be specified at its declaration is the size of the data we will be feeding in. For X we use a shape of [None, 784], where None represents any amount, as we will be feeding in an undefined number of 784-pixel images. The shape of Y is [None, 10] as we will be using it for an undefined number of label outputs, with 10 possible classes. The keep\_prob tensor is used to control the dropout rate, and we initialize it as a placeholder rather than an immutable variable because we want to use the same tensor both for training (when dropout is set to 0.5) and testing (when dropout is set to 1.0). The parameters that the network will update in the training process are the weight and bias values, so for these we need to set an initial value rather than an empty placeholder. These values are essentially where the network does its learning, as they are used in the activation functions of the neurons, representing the strength of the connections between units. Since the values are optimized during training, we could set them to zero for now. But the initial value actually has a significant impact on the final accuracy of the model. We’ll use random values from a truncated normal distribution for the weights. We want them to be close to zero, so they can adjust in either a positive or negative direction, and slightly different, so they generate different errors. This will ensure that the model learns something useful. Add these lines: main.py ...

weights = { 'w1': tf.Variable(tf.truncated\_normal([n\_input, n\_hidden1],

stddev=0.1)),

'w2': tf.Variable(tf.truncated\_normal([n\_hidden1, n\_hidden2], stddev=0.1)),

'w3': tf.Variable(tf.truncated\_normal([n\_hidden2, n\_hidden3], stddev=0.1)),

'out': tf.Variable(tf.truncated\_normal([n\_hidden3, n\_output], stddev=0.1)), }

For the bias, we use a small constant value to ensure that the tensors activate in the intial stages and therefore contribute to the propagation.

The weights and bias tensors are stored in dictionary objects for ease of access. Add this code to your file to define the biases: main.py

... biases = { 'b1': tf.Variable(tf.constant(0.1, shape=[n\_hidden1])), 'b2': tf.Variable(tf.constant(0.1, shape=[n\_hidden2])),

'b3': tf.Variable(tf.constant(0.1, shape=[n\_hidden3])), 'out': tf.Variable(tf.constant(0.1, shape=[n\_output])) }

Next, set up the layers of the network by defining the operations that will manipulate the tensors. Add these lines to your file: main.py ...

layer\_1 = tf.add(tf.matmul(X, weights['w1']), biases['b1'])

layer\_2 = tf.add(tf.matmul(layer\_1, weights['w2']), biases['b2']) layer\_3 = tf.add(tf.matmul(layer\_2, weights['w3']), biases['b3']) layer\_drop = tf.nn.dropout(layer\_3, keep\_prob) output\_layer = tf.matmul(layer\_3, weights['out']) + biases['out']

Each hidden layer will execute matrix multiplication on the previous layer’s outputs and the current layer’s weights, and add the bias to these values. At the last hidden layer, we will apply a dropout operation using our keep\_prob value of 0.5.

The final step in building the graph is to define the loss function that we want to optimize. A popular choice of loss function in TensorFlow programs is cross-entropy, also known as log-loss, which quantifies the difference between two probability distributions (the predictions and the labels). A perfect classification would result in a cross-entropy of 0, with the loss completely minimized. We also need to choose the optimization algorithm which will be used to minimize the loss function. A process named gradient descent optimization is a common method for finding the (local) minimum of a function by taking iterative steps along the gradient in a negative (descending) direction. There are several choices of gradient descent optimization algorithms already implemented in TensorFlow, and in this tutorial we will be using the Adam optimizer. This extends upon gradient descent optimization by using momentum to speed up the process through computing an exponentially weighted average of the gradients and using that in the adjustments. Add the following code to your file:

main.py ...

cross\_entropy = tf.reduce\_mean( tf.nn.softmax\_cross\_entropy\_with\_logits(

labels=Y, logits=output\_layer ))

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy) We’ve now defined the network and built it out with TensorFlow. The next step is to feed data through the graph to train it, and then test that it has actually learnt something.

**Step 5 — Training and Testing**

The training process involves feeding the training dataset through the graph and optimizing the loss function. Every time the network iterates through a batch of more training images, it updates the parameters to reduce the loss in order to more accurately predict the digits shown. The testing process involves running our testing dataset through the trained graph, and keeping track of the number of images that are correctly predicted, so that we can calculate the accuracy. Before starting the training process, we will define our method of evaluating the accuracy so we can print it out on mini-batches of data while we train. These printed statements will allow us to check that from the first iteration to the last, loss decreases and accuracy increases; they will also allow us to track whether or not we have ran enough iterations to reach a consistent and optimal result: main.py ...

correct\_pred = tf.equal(tf.argmax(output\_layer, 1),

tf.argmax(Y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

In correct\_pred, we use the arg\_max function to compare which images are being predicted correctly by looking at the output\_layer (predictions) and Y (labels), and we use the equal function to return this as a list of Booleans.

We can then cast this list to floats and calculate the mean to get a total accuracy score. We are now ready to initialize a session for running the graph. In this session we will feed the network with our training examples, and once trained, we feed the same graph with new test examples to determine the accuracy of the model. Add the following lines of code to your file:

main.py ...

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

The essence of the training process in deep learning is to optimize the loss function. Here we are aiming to minimize the difference between the predicted labels of the images, and the true labels of the images. The process involves four steps which are repeated for a set number of iterations: Propagate values forward through the network Compute the loss Propagate values backward through the network Update the parameters At each training step, the parameters are adjusted slightly to try and reduce the loss for the next step. As the learning progresses, we should see a reduction in loss, and eventually we can stop training and use the network as a model for testing our new data. Add this code to the file:

main.py ...

# train on mini batches for i in range(n\_iterations):

batch\_x,

batch\_y = mnist.train.next\_batch(batch\_size)

sess.run(train\_step, feed\_dict={ X: batch\_x, Y: batch\_y, keep\_prob: dropout })

# print loss and accuracy (per minibatch) if i % 100 == 0: minibatch\_loss, minibatch\_accuracy = sess.run( [cross\_entropy, accuracy],

feed\_dict={X: batch\_x, Y: batch\_y, keep\_prob: 1.0} ) print( "Iteration",

str(i), "\t| Loss =", str(minibatch\_loss), "\t| Accuracy =", str(minibatch\_accuracy) )

After 100 iterations of each training step in which we feed a mini-batch of images through the network, we print out the loss and accuracy of that batch. Note that we should not be expecting a decreasing loss and increasing accuracy here, as the values are per batch, not for the entire model. We use mini-batches of images rather than feeding them through individually to speed up the training process and allow the network to see a number of different examples before updating the parameters. Once the training is complete, we can run the session on the test images. This time we are using a keep\_prob dropout rate o f 1.0 to ensure all units are active in the testing process. Add this code to the file: main.py ...

test\_accuracy = sess.run(accuracy, feed\_dict={X: mnist.test.images,

Y:mnist.test.labels, keep\_prob: 1.0})

print("\nAccuracy on test set:", test\_accuracy)

It’s now time to run our program and see how accurately our neural network can recognize these handwritten digits. Save the main.py file and execute the following command in the terminal to run the script: (tensorflow-demo) $ python main.py You’ll see an output similar to the following, although individual loss and accuracy results may vary slightly:

Output Iteration 0 | Loss = 3.67079 | Accuracy = 0.140625 Iteration 100 | Loss = 0.492122 | Accuracy = 0.84375 Iteration 200 | Loss = 0.421595 | Accuracy = 0.882812 Iteration 300 | Loss = 0.307726 | Accuracy = 0.921875 Iteration 400 | Loss = 0.392948 | Accuracy = 0.882812 Iteration 500 | Loss = 0.371461 | Accuracy = 0.90625 Iteration 600 | Loss = 0.378425 | Accuracy = 0.882812 Iteration 700 | Loss = 0.338605 | Accuracy = 0.914062 Iteration 800 | Loss = 0.379697 | Accuracy = 0.875 Iteration 900 | Loss = 0.444303 | Accuracy = 0.90625

Accuracy on test set: 0.9206 To try and improve the accuracy of our model, or to learn more about the impact of tuning hyperparameters, we can test the effect of changing the learning rate, the dropout threshold, the batch size, and the number of iterations. We can also change the number of units in our hidden layers, and change the amount of hidden layers themselves, to see how different architectures increase or decrease the model accuracy.

To demonstrate that the network is actually recognizing the handdrawn images, let’s test it on a single image of our own. If you are on a local machine and you would like to use your own hand-drawn number, you can use a graphics editor to create your own 28x28 pixel image of a digit. Otherwise, you can use curl to download the following sample test image to your server or computer: (tensorflow-demo) $ curl -O images/test\_img.png Open the main.py file in your editor and add the following lines of code to the top of the file to import two libraries necessary for image manipulation. main.py import numpy as np from PIL import Image ... Then at the end of the file, add the following line of code to load the test image of the handwritten digit:

main.py ...

img = np.invert(Image.open("test\_img.png").convert('L')).ravel()

The open function of the Image library loads the test image as a 4D array containing the three RGB color channels and the Alpha transparency.

This is not the same representation we used previously when reading in the dataset with TensorFlow, so we’ll need to do some extra work to match the format. First, we use the convert function with the L parameter to reduce the 4D RGBA representation to one grayscale color channel. We store this as a numpy array and invert it using np.invert, because the current matrix represents black as 0 and white as 255, whereas we need the opposite. Finally, we call ravel to flatten the array. Now that the image data is structured correctly, we can run a session in the same way as previously, but this time only feeding in the single image for testing. Add the following code to your file to test the image and print the outputted label.

main.py ... prediction = sess.run(tf.argmax(output\_layer, 1), feed\_dict={X: [img]})

print ("Prediction for test image:", np.squeeze(prediction))

The np.squeeze function is called on the prediction to return the single integer from the array (i.e. to go from [2] to 2).

The resulting output demonstrates that the network has recognized this image as the digit 2. Output Prediction for test image: 2 You can try testing the network with more complex images –– digits that look like other digits, for example, or digits that have been drawn poorly or incorrectly –– to see how well it fare

## What Is Pandas In Python

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named [Numpy](https://www.activestate.com/products/python/python-packages/), which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other [data science](https://www.activestate.com/products/python/python-data-science/) modules inside the Python ecosystem, and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState’s [ActivePython](https://platform.activestate.com/featured-projects).

Pandas make it simple to do many of the time consuming, repetitive tasks associated with working with data, including:

* Data cleansing
* Data fill
* Data normalization
* Merges and joins
* Data visualization
* Statistical analysis
* Data inspection
* Loading and saving data
* And much more

## What is Keras In Python

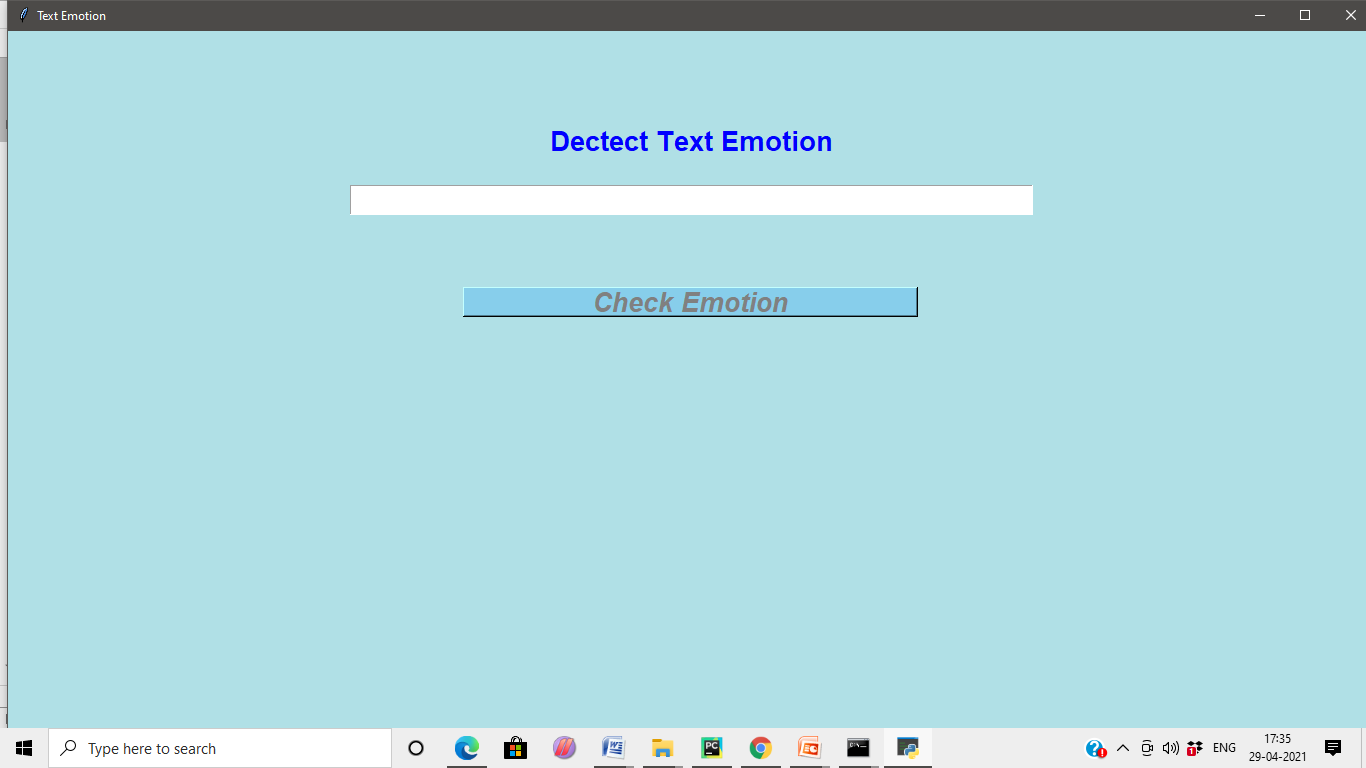
**Keras** is an Open Source Neural Network library written in Python that runs on top of Theano or Tensorflow. It is designed to be modular, fast and easy to use. It was developed by François Chollet, a Google engineer. Keras doesn't handle low-level computation. Instead, it uses another library to do it, called the "Backend.

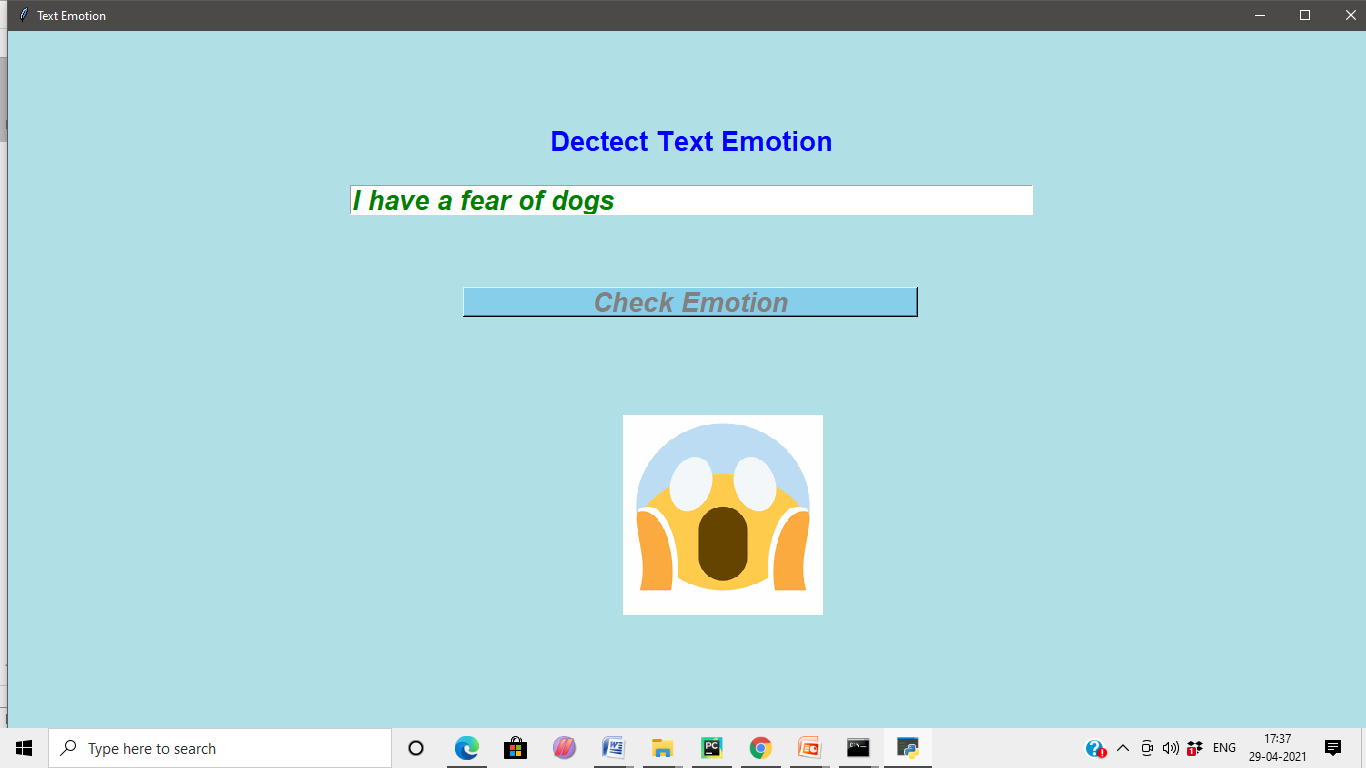
Keras is high-level API wrapper for the low-level API, capable of running on top of TensorFlow, CNTK, or Theano. Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function. Keras in Python doesn't handle Low-Level API such as making the computational graph, making tensors or other variables because it has been handled by the "backend" engine.

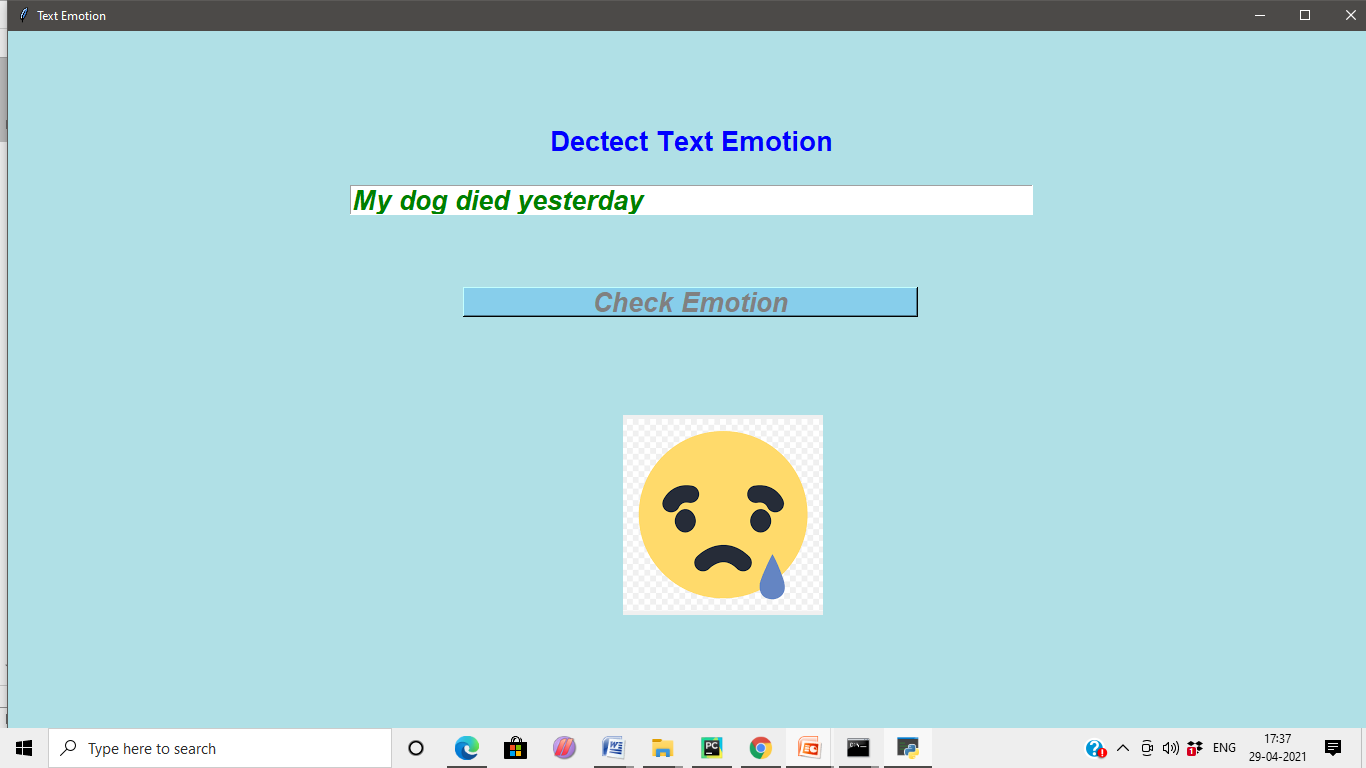
**CODE**

**import** re  
**from** collections **import** Counter  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.svm **import** SVC  
**from** sklearn.svm **import** LinearSVC  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.tree **import** DecisionTreeClassifier  
**import** pickle  
*#import emoji***from** tkinter **import** \*  
**from** PIL **import** ImageTk ,Image  
  
root = Tk()  
w, h = root.winfo\_screenwidth(), root.winfo\_screenheight()  
root.geometry(**"%dx%d+0+0"** % (w, h))  
root.config(bg=**"powder blue"**)  
root.title(**"Text Emotion"**)  
  
ent\_var=StringVar()  
  
**global** X\_train, X\_test  
  
**def** read\_data():  
 data = []  
 **with** open(**'text\_emotion.txt'**, **'r'**)**as** f:  
 **for** line **in** f:  
 line = line.strip()  
 label = **' '**.join(line[1:line.find(**"]"**)].strip().split())  
 text = line[line.find(**"]"**)+1:].strip()  
 data.append([label, text])  
 **return** data  
data = read\_data()  
print(**"Number of instances: {}"**.format(len(data)))  
  
**def** ngram(token, n):  
 output = []  
 **for** i **in** range(n-1, len(token)):  
 ngram = **' '**.join(token[i-n+1:i+1])  
 output.append(ngram)  
 **return** output  
**def** create\_feature(text, nrange=(1, 1)):  
 text\_features = []  
 text = text.lower()  
 text\_alphanum = re.sub(**'[^a-z0-9#]'**, **' '**, text)  
 **for** n **in** range(nrange[0], nrange[1]+1):  
 text\_features += ngram(text\_alphanum.split(), n)  
 text\_punc = re.sub(**'[a-z0-9]'**, **' '**, text)  
 text\_features += ngram(text\_punc.split(), 1)  
 **return** Counter(text\_features)  
  
  
**def** convert\_label(item, name):  
 items = list(map(float, item.split()))  
 label = **""  
 for** idx **in** range(len(items)):  
 **if** items[idx] == 1:  
 label += name[idx] + **" "  
  
 return** label.strip()  
  
emotions = [**"joy"**, **'fear'**, **"anger"**, **"sadness"**, **"disgust"**, **"shame"**, **"guilt"**]  
  
X\_all = []  
y\_all = []  
**for** label, text **in** data:  
 y\_all.append(convert\_label(label, emotions))  
 X\_all.append(create\_feature(text, nrange=(1, 4)))  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y\_all, test\_size = 0.2, random\_state = 15)  
  
**def** train\_test(clf, X\_train, y\_train):  
 clf.fit(X\_train, y\_train)  
 filename=**"trained\_model.sav"** pickle.dump(clf, open(filename, **"wb"**))  
  
**def** text():  
 svc = SVC()  
 lsvc = LinearSVC(random\_state=123)  
 rforest = RandomForestClassifier(random\_state=123)  
 dtree = DecisionTreeClassifier()  
  
 clifs = [svc, lsvc, rforest, dtree]  
 *# train and test them* print(**"| {:25} | {} | {} |"**.format(**"Classifier"**, **"Training Accuracy"**, **"Test Accuracy"**))  
 print(**"| {} | {} | {} |"**.format(**"-"** \* 25, **"-"** \* 17, **"-"** \* 13))  
  
 **for** clf **in** clifs:  
 clf\_name = clf.\_\_class\_\_.\_\_name\_\_  
 full\_train=train\_test(clf, X\_train, y\_train)  
 print(full\_train)  
  
l = [**"joy"**, **'fear'**, **"anger"**, **"sadness"**, **"disgust"**, **"shame"**, **"guilt"**]  
l.sort()  
print(l)  
label\_freq = {}  
  
**for** label, \_ **in** data:  
 label\_freq[label] = label\_freq.get(label, 0) + 1  
*#print(label\_freq)  
  
# print the labels and their counts in sorted order***for** l **in** sorted(label\_freq, key=label\_freq.get, reverse=**True**):  
 print(**"{:10}({}) {}"**.format(convert\_label(l, emotions), l, label\_freq[l]))  
  
joy=Image.open(**"joy.png"**)  
joy=joy.resize((200,200),Image.ANTIALIAS)  
joy=ImageTk.PhotoImage(joy)  
  
fear=Image.open(**"fear.png"**)  
fear=fear.resize((200,200),Image.ANTIALIAS)  
fear=ImageTk.PhotoImage(fear)  
  
anger=Image.open(**"anger.png"**)  
anger=anger.resize((200,200),Image.ANTIALIAS)  
anger=ImageTk.PhotoImage(anger)  
  
sadness=Image.open(**"sadeness.png"**)  
sadness=sadness.resize((200,200),Image.ANTIALIAS)  
sadness=ImageTk.PhotoImage(sadness)  
  
disgust=Image.open(**"disgust.png"**)  
disgust=disgust.resize((200,200),Image.ANTIALIAS)  
disgust=ImageTk.PhotoImage(disgust)  
  
shame=Image.open(**"shame.png"**)  
shame=shame.resize((200,200),Image.ANTIALIAS)  
shame=ImageTk.PhotoImage(shame)  
  
guilt=Image.open(**"guilt.png"**)  
guilt=guilt.resize((200,200),Image.ANTIALIAS)  
guilt=ImageTk.PhotoImage(guilt)  
  
emoji\_dict = {**"joy"**:joy, **"fear"**:fear, **"anger"**:PhotoImage(file = **r"anger.png"**), **"sadness"**:PhotoImage(file = **r"sadeness.png"**), **"disgust"**:PhotoImage(file = **r"disgust.png"**), **"shame"**:PhotoImage(file = **r"shame.png"**), **"guilt"**:PhotoImage(file = **r"guilt.png"**)}  
  
  
t1 = **"This looks so impressive"**t2 = **"I have a fear of dogs"**t3 = **"My dog died yesterday"**t4 = **"I don't love you anymore..!"  
  
from** sklearn.feature\_extraction **import** DictVectorizer  
vectorizer = DictVectorizer(sparse = **True**)  
X\_train = vectorizer.fit\_transform(X\_train)  
X\_test = vectorizer.transform(X\_test)  
  
**def** abcs():  
  
 features = create\_feature(ent\_var.get(), nrange=(1, 4))  
 features = vectorizer.transform(features)  
 model = pickle.load(open(**"trained\_model.sav"**, **"rb"**))  
 prediction = model.predict(features)[0]  
 print(prediction)  
 lbl2 = Label(root, image=emoji\_dict[prediction])  
 lbl2.place(x=(w/5+w/4), y=(h/4+h/4), height=200, width=200)  
  
  
**for** i **in** ent\_var.get():  
 features = create\_feature(i, nrange=(1, 4))  
 features = vectorizer.transform(features)  
 model=pickle.load(open(**"trained\_model.sav"**, **"rb"**))  
 prediction = model.predict(features)[0]  
 print(prediction)  
 lbl2 = Label(root, image=emoji\_dict[prediction])  
 lbl2.place(x=0, y=0)  
  
  
  
  
  
  
lbl=Label(root, text=**"Dectect Text Emotion"**, bg=**"powder blue"**, fg=**"blue"**, font=(**"arial"**, 20, **"bold"**)).place(x=w/4, y=h/8, height=30, width=w/2)  
  
ent=Entry(root, fg=**"green"**, textvariable=ent\_var, font=(**"arial"**, 20, **"bold italic"**))  
ent.place(x=w/4, y=h/5, height=30, width=w/2)  
  
btn=Button(root, text=**"Check Emotion"**, bg=**"sky blue"**, fg=**"gray"**, font=(**"arial"**, 20, **"bold italic"**), command=abcs)  
btn.place(x=w/3, y=h/3, height=30, width=w/3)  
  
  
  
root.mainloop()

**Screenshots**







**CONCLUSION**

The most important part is getting good quality of data. Often

tweets retrieved consists of only URLs and hence pre-processing

data still remains one of the most crucial steps which needs to be

improved. Secondly, only explicit emotions are explored in the

tweet so detecting implicit emotions in objective sentences can in

included in future work. At present, each sentence is treated as

individual unit. Further the association between the sentences can be

included. The proposed system works at the syntactic level only,

semantic level parsing can be implemented further. Finally, the Bag

of words needs to be more exhaustive and hence more words

describing emotions can be added to these bags.

**REFERENCES**

[1] Mayur Jain,” Smart Home System Using Android Mobile Devices”, Journal of Computing Technologies Vol

2, Issue 3 ISSN 2278

 –

 3814. [2] IEEE std. 802.15.4 -

2003: “Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications

for Low Rate

Wireless Personal Area Networks (LRWPANs)”

[3] KiumiAkingbehin,AkinsolaAkingbehin.,“Alternativ es for Short Range Low PowerWireless Communications,” IEEE.2005

: 94 - 95.

[4] QIN Tinghao, DOU Xiaoqian, “Application of ZigBee Technology in Wireless Sensor Network,”

Instrumentation Technology, 2007, pp.57-59