Welcome There!

Let us begin!

The document you are reading is not a static web page, but an interactive environment called a Colab notebook that lets you write and execute code.

Getting Started

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

System Level Commands

You can run any (allowed) system commands usin "!" sign.

```
!ls ..
```

```
bin      dev   lib32   mnt                       root  sys    var
boot     etc   lib64   NGC-DL-CONTAINER-LICENSE  run   tmp
content  home  libx32  opt                       sbin  tools
datalab  lib   media   proc                      srv   usr
```

```
!git
```

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
```

```
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

```
!cat /etc/os-release
```

```
NAME="Ubuntu"
VERSION="20.04.5 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.5 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

Machine Learning

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Colab is used extensively in the machine learning community with applications including:

- Getting started with Torch, TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

Working with Data

We mostly use CSV, json, and sometimes pickles to store our data. You can save them either in your local system or google drive and simply use them in colab.

Mounting Google Drive locally

The example below shows how to mount your Google Drive on your runtime using an authorization code, and how to write and read files there. Once executed, you will be able to see the new file (`foo.txt`) at https://drive.google.com/.

This only supports reading, writing, and moving files; to programmatically modify sharing settings or other metadata, use one of the other options below.

Note: When using the 'Mount Drive' button in the file browser, no authentication codes are necessary for notebooks that have only been edited by the current user.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
with open('/content/drive/My Drive/foo.txt', 'w') as f:
  f.write('Hello Google Drive!')
!cat /content/drive/My\ Drive/foo.txt
```

```
Hello Google Drive!
```

```
drive.flush_and_unmount()
print('All changes made in this colab session should now be visible in Drive.')
```

```
All changes made in this colab session should now be visible in Drive.
```

Read CSV files

```
import csv
with open('dummy.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        pass
```

Load Json files

```
import json
with open('dummy.json', 'r') as f:
    data = json.load(f)
```

Pickle

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

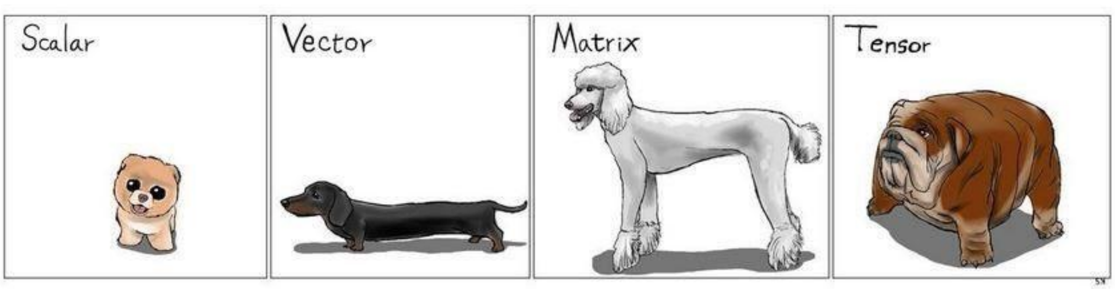
```
dummy_object = {"hey": [1], "hi": [46]}
import pickle
with open("dummy.pickle", 'wb') as f:
    pickle.dump(dummy_object, f)
with open("dummy.pickle", "rb") as f:
    d = pickle.load(f)
d
```

```
{'hey': [1], 'hi': [46]}
```

Numpy

NumPy gives you an enormous range of fast and efficient ways of creating arrays and manipulating numerical data inside them. While a Python list can contain different data types within a single list, all of the elements in a NumPy array should be homogeneous. The mathematical operations that are meant to be performed on arrays would be extremely inefficient if the arrays weren’t homogeneous.

NumPy arrays are faster and more compact than Python lists. An array consumes less memory and is convenient to use. NumPy uses much less memory to store data and it provides a mechanism of specifying the data types. This allows the code to be optimized even further.



```
import numpy as np
x = np.array([
              [1, 2],
              [3, 4]
              ])
y = np.array([[5, 6]])
x.shape, y.shape
```

```
((2, 2), (1, 2))
```

```
z = np.dot(y, x)
z, z.shape
```

```
(array([[23, 34]]), (1, 2))
```

```
z = np.concatenate([x, y], axis=0)
z, z.shape
```

```
(array([[1, 2],
        [3, 4],
        [5, 6]]), (3, 2))
```

```
a = np.array([[[[[[[[1, 2, 3]]]]]]]])
a.shape
```

```
(1, 1, 1, 1, 1, 1, 1, 3)
```

```
np.squeeze(a), np.squeeze(a).shape
```

```
(array([1, 2, 3]), (3,))
```

Alright, Hands on Keyboard!!

Okay, let's start coding a simple regression to predict whether a given word is verb or not. You will use GloVe word embedding to extract the word meaning, then you will input it to a simple regerssion and output the probability of being a verb! In other words, you will calculate :

$$P(\text{W is verb} \mid \text{W="sample"})$$

Please run these cells to gather required dataset.

Words are ready! You have to download GloVe embedding vectors from this [link](http://nlp.stanford.edu/data/glove.6B.zip) using "wget" command

```
######## YOUR CODE HERE #############
!wget http://nlp.stanford.edu/data/glove.6B.zip
```

```
--2023-03-04 18:30:16--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2023-03-04 18:30:16--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2023-03-04 18:30:16--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.01MB/s    in 2m 39s

2023-03-04 18:32:56 (5.17 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
# Unzip downloaded file! cmd: unzip
###### YOUR CODE HERE ###########
!unzip /content/glove.6B.zip
```

```
Archive:  /content/glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

Next step is loading glove embeddings into a map! You have a mapping from a "Word" to is embedding!

```
import numpy as np
embeddings_index = {}
f = open('glove.6B.300d.txt', encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    coefs = np.expand_dims(coefs, axis=0)
    embeddings_index[word] = coefs
f.close()
```

```
embeddings_index['seek'].shape
```

```
(1, 300)
```

Our embeddings are ready lets gather some english words! Run cells below!!

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
True
```

```
from nltk.corpus import wordnet as wn
import random
SEED = 4678
VERBS = []
for synset in list(wn.all_synsets(wn.VERB)):
    verb = synset.name().split('.')[0]
    if(verb in embeddings_index):
        VERBS.append(verb)

random.Random(SEED).shuffle(VERBS)
NOUNS = []

for synset in list(wn.all_synsets(wn.NOUN)):
    noun = synset.name().split('.')[0]
    if(noun in embeddings_index):
        NOUNS.append(noun)


random.Random(SEED).shuffle(NOUNS)
```

```
len(VERBS), len(NOUNS)
```

```
(10891, 43680)
```

```
NOUNS = NOUNS[:100]
VERBS = VERBS[:100]

x_train = NOUNS[:64] + VERBS[:64]
y_train = [0] * 64 + [1] * 64
x_test = NOUNS[64:] + VERBS[64:]
y_test = [0] * 36 + [1] * 36

c = list(zip(x_train, y_train))
random.Random(SEED).shuffle(c)
x_train, y_train = zip(*c)
y_train = np.array(y_train)
c = list(zip(x_test, y_test))
random.Random(SEED).shuffle(c)
```

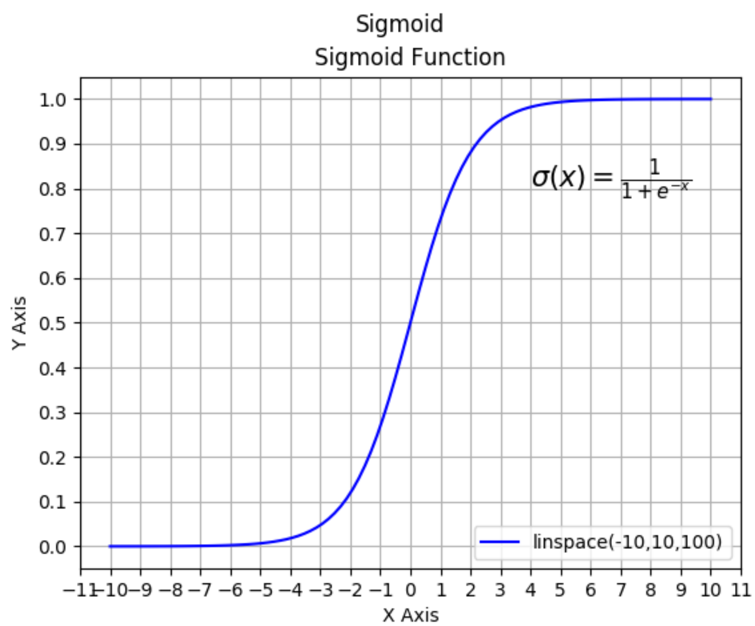
```
x_test, y_test = zip(*c)
y_test = np.array(y_test)
```

```
for word in x_train + x_test :
    assert len(embeddings_index[word][0]) == 300
```

Dataset is ready, time to implement our Logistic regression model! Logistic regression takes a vector as input and outputs a probability for each vector!! This model uses Sigmoid function:

The exact formula of sigmoid is:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Hypothesis should be like:

$$H(X, w, b) = Sigmoid(W^T.X + b)$$

Cost function should be like:

$$J = -\Sigma_m Y log(H) + (1 - Y) log(1 - H)$$

```
class LogisticRegression:

    def __init__(self, learning_rate = 0.01, num_iterations = 2000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.w = []
        self.b = 0

    def initialize_weight(self,dim):
        """
        This function creates a vector of zeros of shape (dim, 1)  for w and initializes b to 0.
        returns w and b
        Argument:
        dim -- size of the w vector we want (or number of parameters  in this case)
        """
        #### YOUR CODE HERE #####
        self.w = np.zeros(dim)
        self.b = 0
        return self.w , self.b

    def sigmoid(self,z):
```

```
        """
        Compute the sigmoid of z
        Argument:
        z -- is the decision boundary of the classifier
        """
        #### YOUR CODE HERE #####
        return 1 / (1 + np.exp(-z))

    def hypothesis(self,w,X,b):
        """
        This function calculates the hypothesis for the present model
        Argument:
        w -- weight vector
        X -- The input vector
        b -- The bias vector
        """
        #### YOUR CODE HERE #####
        return self.sigmoid(np.transpose(w) @ X + b)

    def cost(self,H,Y,m):
        """
        This function calculates the cost of hypothesis
        Arguments:
        H -- The hypothesis vector
        Y -- The output
        m -- Number training samples
        """
        #### YOUR CODE HERE #####
        return np.sum(np.dot(Y , np.log(H)) + np.dot((1 - Y) , np.log(1 - H)))

    def cal_gradient(self, w,H,X,Y):
        """
        Calculates gradient of the given model in learning space
        """
        m = X.shape[1]
        dw = np.dot(X,(H-Y).T)/m
        db = np.sum(H-Y)/m
        grads = {"dw": dw,
                 "db": db}
        return grads

    def predict(self,X):
        '''
        Predict whether the label is 0 or 1 using learned logistic regression parameters (w, b)

        Arguments:
        w -- weights, a numpy array of size (n, 1)
        b -- bias, a scalar
        X -- data of size (num_px * num_px * 3, number of examples)

        Returns:
        Y_prediction -- a numpy array (vector) containing all predictions (0/1) for the examples in X
        '''
        X = np.array(X)
        m = X.shape[1]

        Y_prediction = np.zeros((1,m))

        w = self.w.reshape(X.shape[0], 1)
        b = self.b
        # Compute vector "H"
        H = self.hypothesis(w, X, b)

        for i in range(H.shape[1]):
        # Convert probabilities H[0,i] to actual predictions p[0,i]
            if H[0,i] >= 0.5:
                Y_prediction[0,i] = 1
            else:
                Y_prediction[0,i] = 0

        return Y_prediction
    def gradient_position(self, w, b, X, Y):
        """
        It just gets calls various functions to get status of learning model
        Arguments:
        w -- weights, a numpy array of size (dim, 1)
        b -- bias, a scalar
```

```
        X -- data of size (b, dim)
        Y -- true "label" vector (containing 0 or 1 ) of size (b, number of examples)
        """
        m = X.shape[1]
        H = self.hypothesis(w,X,b)         # compute activation
        cost = self.cost(H,Y,m)               # compute cost
        grads = self.cal_gradient(w, H, X, Y) # compute gradient
        return grads, cost

    def gradient_descent(self, w, b, X, Y, print_cost = False):
        """
        This function optimizes w and b by running a gradient descent algorithm

        Arguments:
        w — weights, a numpy array of size (num_px * num_px * 3, 1)
        b — bias, a scalar
        X -- data of size (no. of features, number of examples)
        Y -- true "label" vector (containing 0 or 1 ) of size (1, number of examples)
        print_cost — True to print the loss every 100 steps

        Returns:
        params — dictionary containing the weights w and bias b
        grads — dictionary containing the gradients of the weights and bias with respect to the cost function
        costs — list of all the costs computed during the optimization, this will be used to plot the learning curve.
        """

        costs = []

        for i in range(self.num_iterations):
        # Cost and gradient calculation
            grads, cost = self.gradient_position(w,b,X,Y)


        # Retrieve derivatives from grads
        dw = grads["dw"]
        db = grads["db"]

        # update rule
        w = w - (self.learning_rate * dw)
        b = b - (self.learning_rate * db)

        # Record the costs
        if i % 100 == 0:
            costs.append(cost)

        # Print the cost every 100 training iterations
        if print_cost and i % 100 == 0:
             print ("Cost after iteration %i: %f" %(i, cost))


        params = {"w": w,
                  "b": b}

        grads = {
                 "dw": dw,
                 "db": db
                 }

        return params, grads, costs

    def train_model(self, X_train, Y_train, X_test, Y_test, print_cost = False):
        """
        Builds the logistic regression model by calling the function you've implemented previously

        Arguments:
        X_train — training set represented by a numpy array of shape (features, m_train)
        Y_train — training labels represented by a numpy array (vector) of shape (1, m_train)
        X_test — test set represented by a numpy array of shape (features, m_test)
        Y_test — test labels represented by a numpy array (vector) of shape (1, m_test)
        print_cost — Set to true to print the cost every 100 iterations

        Returns:
        d — dictionary containing information about the model.
        """
        # initialize parameters with zeros
        dim = np.shape(X_train)[0]
        w, b = self.initialize_weight(dim)
```

```
        # Gradient descent

        parameters, grads, costs = self.gradient_descent(w, b, X_train, Y_train, print_cost = False)

        # Retrieve parameters w and b from dictionary “parameters”
        self.w = parameters["w"]
        self.b = parameters["b"]

        # Predict test/train set examples
        Y_prediction_test = self.predict(X_test)
        Y_prediction_train = self.predict(X_train)
        # Print train/test Errors
        train_score = 100 - np.mean(np.abs(Y_prediction_train - Y_train)) * 100
        test_score = 100 - np.mean(np.abs(Y_prediction_test - Y_test)) * 100
        print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_train - Y_train)) * 100))
        print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_test - Y_test)) * 100))
        d = {"costs": costs,
             "w" : self.w,
             "b" : self.b,
             "learning_rate": self.learning_rate,
             "num_iterations": self.num_iterations,
             "train accuracy": train_score,
             "test accuracy" : test_score}

        return d
```

Congrats!! We've got a simple regression model, time to concatenate all vector embeddings and train the model on it!!

```
X_train = []
X_test = []
for word in x_train:
    ###### YOUR CODE HERE #####
    # you have to add each word embedding to train array
    X_train.append(embeddings_index[word])

for word in x_test:
    ###### YOUR CODE HERE #####
    # you have to add each word embedding to test array
    X_test.append(embeddings_index[word])

X_train = np.concatenate(X_train) # Conacat all embedding to a simple matrix
X_test = np.concatenate(X_test)  # Concat all embeddings to a simple matrix

X_train = X_train.transpose(1, 0)
X_test = X_test.transpose(1, 0)


X_train.shape, X_test.shape

#exptected: ((300, 128), (300, 72))
```

```
((300, 128), (300, 72))
```

Results!

```
regression = LogisticRegression()
results = regression.train_model(X_train, y_train, X_test, y_test, print_cost = False)
```

```
train accuracy: 80.46875 %
test accuracy: 72.22222222222223 %
```

Save them!

Save your results in a pickle and move it to your google drive!

```
##### YOUR CODE HERE #####
import pickle
with open("result.pickle", 'wb') as f:
    pickle.dump(results, f)
```

```
!mv result.pickle /content/drive/MyDrive
```

HAVE FUN

✓ 0s completed at 10:06 PM