**Group Names: Ruby Robles ID: 20246862**

**Audrey Sanchez ID: 20526887**

**ELEE 4333 07R Machine Learning Fall 2020**

**Final Project – Fake News Detection 12/09/20**

**Abstract:**

The idea is to detect “REAL” news from the “FAKE” one(s) with the use of language detection patterns. The code will be a mixture between the original coding plus the coding from homework 6 of chapter 6 (provided by the professor). This will then show how we can use machine learning to solve a problem such as this.

**Introduction:**

We will use the code called, “FakeNews” (.ipynb file) mixed with the HW.6 of CH.6 Basics of Neural Networks code, and the dataset called, “news’ (.csv file) that we got from Kaggle, and for the code, since we got the idea from “Top 47 Machine Learning Projects for 2020”, to get us started we will start using that code provided in the website to see how it works and then make it our own by introducing some of the techniques we’ve learned from our Machine Learning class.

**Data/simulation results:**

The dataset we will be using for this python project- is called “news.csv”. This dataset has a shape of 7796×4. The first column identifies the news, the second the title, the third the text, and the fourth the labels that portray whether the news is REAL or FAKE. The screen shot of the table is shown below on figure 1.

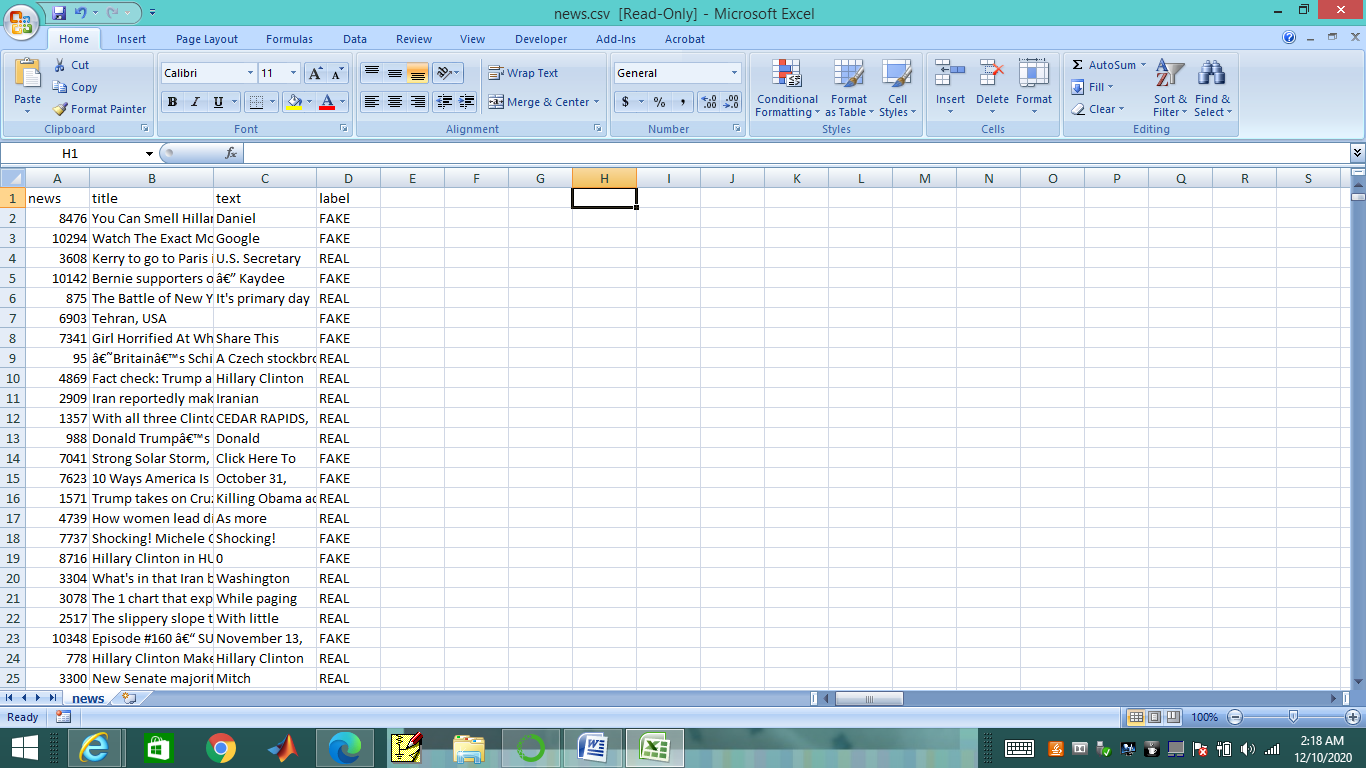
****

Figure 1 – A portion of the Dataset “news.csv” Table

The original code mixed with HW.6:

#Package imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import itertools

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

In [17]:df=pd.read\_csv('../Final Project ML/news.csv')

#Get shape and head

df.shape

df.head()

Out[17]:

|  | Unnamed: 0 | Title | text | label |
| --- | --- | --- | --- | --- |
| 0 | 8476 | You Can Smell Hillary’s Fear | Daniel Greenfield, a Shillman Journalism Fello... | FAKE |
| 1 | 10294 | Watch The Exact Moment Paul Ryan Committed Pol... | Google Pinterest Digg Linkedin Reddit Stumbleu... | FAKE |
| 2 | 3608 | Kerry to go to Paris in gesture of sympathy | U.S. Secretary of State John F. Kerry said Mon... | REAL |
| 3 | 10142 | Bernie supporters on Twitter erupt in anger ag... | — Kaydee King (@KaydeeKing) November 9, 2016 T... | FAKE |
| 4 | 875 | The Battle of New York: Why This Primary Matters | It's primary day in New York and front-runners... | REAL |

In [18]:#DataFlair - Get the labels

labels=df.label

labels.head()

#First column is labels - Split the dataset

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df['text'], labels,

test\_size=0.2, random\_state=7)

In [19]:np.random.seed(1) # set a seed so that the results are consistent

no\_of\_different\_labels = 100

lr = np.arange(no\_of\_different\_labels)

X = np.transpose(x\_train)

Y = np.transpose(y\_train)

X\_test = np.transpose(x\_test)

In [20]:# Neural network from scratch

def softmax(x):

t=np.exp(x)

s = t/np.sum(t, axis=0)

return s

In [21]:def layer\_sizes(X, Y):

### START CODE HERE ### (≈ 3 lines of code)

n\_x = X.shape[0] # size of input layer

n\_h = 25

n\_y = Y.shape[0] # size of output layer

### END CODE HERE ###

return (n\_x, n\_h, n\_y)

#X\_assess, Y\_assess = layer\_sizes\_test\_case()

(n\_x, n\_h, n\_y) = layer\_sizes(X, Y)

print("The size of the input layer is: n\_x = " + str(n\_x))

print("The size of the hidden layer is: n\_h = " + str(n\_h))

print("The size of the output layer is: n\_y = " + str(n\_y))

The size of the input layer is: n\_x = 5068

The size of the hidden layer is: n\_h = 25

The size of the output layer is: n\_y = 5068

In [22]:# GRADED FUNCTION: initialize\_parameters

def initialize\_parameters(n\_x, n\_h, n\_y):

np.random.seed(2) # we set up a seed so that your output matches ours although the initialization is random.

### START CODE HERE ### (≈ 4 lines of code)

W1 = np.random.randn(n\_h, n\_x) \* 0.01

b1 = np.zeros((n\_h, 1))

W2 = np.random.randn(n\_y, n\_h) \* 0.01

b2 = np.zeros((n\_y,1))

### END CODE HERE ###

assert (W1.shape == (n\_h, n\_x))

assert (b1.shape == (n\_h, 1))

assert (W2.shape == (n\_y, n\_h))

assert (b2.shape == (n\_y, 1))

parameters = {"W1": W1,

"b1": b1,

"W2": W2,

"b2": b2}

return parameters

parameters = initialize\_parameters(n\_x, n\_h, n\_y)

print("W1 = " + str(parameters["W1"]))

print("b1 = " + str(parameters["b1"]))

print("W2 = " + str(parameters["W2"]))

print("b2 = " + str(parameters["b2"]))

W1 = [[-0.00416758 -0.00056267 -0.02136196 ... 0.00934132 -0.00949018

0.00856809]

[ 0.01064315 -0.00482192 0.00847804 ... 0.00925304 -0.01427225

-0.00176544]

[-0.0005321 -0.00677426 0.00399531 ... -0.00201151 0.00191508

0.00247327]

...

[ 0.00856936 0.01548982 -0.00326624 ... 0.00612518 -0.00237981

-0.0066409 ]

[ 0.02006418 0.02156342 0.00956753 ... -0.00091404 0.00022585

-0.00388974]

[ 0.00197121 -0.01533362 0.02548809 ... -0.01051974 -0.00879136

-0.00747072]]

b1 = [[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]

[0.]]

W2 = [[ 0.01682747 -0.00507544 0.00374676 ... 0.00534955 -0.01592226

-0.0108441 ]

[ 0.01114777 -0.00221656 0.01323403 ... 0.01079433 -0.01306529

-0.00297147]

[-0.00862835 0.01976667 0.01549404 ... 0.0067512 -0.00568955

-0.00375892]

...

[ 0.00458375 0.00887979 0.01969026 ... 0.01510116 0.00299173

-0.00967035]

[ 0.00205594 0.01165605 -0.00609715 ... -0.00143894 -0.00376586

-0.0013448 ]

[ 0.00282355 0.0004492 -0.01039126 ... 0.01168276 -0.00414864

-0.0012387 ]]

b2 = [[0.]

[0.]

[0.]

...

[0.]

[0.]

[0.]]

In [23]:# GRADED FUNCTION: forward\_propagation

def forward\_propagation(X, parameters):

# Retrieve each parameter from the dictionary "parameters"

### START CODE HERE ### (≈ 4 lines of code)

W1 = parameters["W1"]

b1 = parameters["b1"]

W2 = parameters["W2"]

b2 = parameters["b2"]

### END CODE HERE ###

# Implement Forward Propagation to calculate A2 (probabilities)

### START CODE HERE ### (≈ 4 lines of code)

Z1 = np.dot(W1, X) + b1

A1 = np.tanh(Z1)

Z2 = np.dot(W2, A1) + b2

A2 = softmax(Z2)

### END CODE HERE ###

assert(A2.shape == (W2.shape[0], X.shape[1]))

cache = {"Z1": Z1,

"A1": A1,

"Z2": Z2,

"A2": A2}

return A2, cache

In [24]:# GRADED FUNCTION: compute\_cost

def compute\_cost(A2, Y, parameters):

m = Y.shape[1] # number of example

# Compute the cross-entropy cost

### START CODE HERE ### (≈ 2 lines of code)

# for multiple-class task

logprobs = np.multiply(np.log(A2), Y)

cost = -1/m\*np.sum(logprobs)

### END CODE HERE ###

cost = np.squeeze(cost) # makes sure cost is the dimension we expect.

# E.g., turns [[17]] into 17

assert(isinstance(cost, float))

return cost

In [25]:# GRADED FUNCTION: backward\_propagation

def backward\_propagation(parameters, cache, X, Y):

m = X.shape[1]

# First, retrieve W1 and W2 from the dictionary "parameters".

### START CODE HERE ### (≈ 2 lines of code)

W1 = parameters["W1"]

W2 = parameters["W2"]

### END CODE HERE ###

# Retrieve also A1 and A2 from dictionary "cache".

### START CODE HERE ### (≈ 2 lines of code)

A1 = cache["A1"]

A2 = cache["A2"]

### END CODE HERE ###

# Backward propagation: calculate dW1, db1, dW2, db2.

### START CODE HERE ### (≈ 6 lines of code, corresponding to 6 equations on slide above)

dZ2 = A2-Y

dW2 = 1./m\*np.dot(dZ2, A1.T)

db2 = 1./m\*np.sum(dZ2, axis = 1, keepdims=True)

dZ1 = np.dot(W2.T, dZ2) \* (1 - np.power(A1, 2))

dW1 = 1./m\*np.dot(dZ1, X.T)

db1 = 1./m\*np.sum(dZ1, axis = 1, keepdims=True)

### END CODE HERE ###

grads = {"dW1": dW1,

"db1": db1,

"dW2": dW2,

"db2": db2}

return grads

In [26]:# GRADED FUNCTION: update\_parameters

def update\_parameters(parameters, grads, learning\_rate = 1.0):

# Retrieve each parameter from the dictionary "parameters"

### START CODE HERE ### (≈ 4 lines of code)

W1 = parameters["W1"]

W2 = parameters["W2"]

b1 = parameters["b1"]

b2 = parameters["b2"]

### END CODE HERE ###

# Retrieve each gradient from the dictionary "grads"

### START CODE HERE ### (≈ 4 lines of code)

dW1 = grads["dW1"]

db1 = grads["db1"]

dW2 = grads["dW2"]

db2 = grads["db2"]

## END CODE HERE ###

# Update rule for each parameter

### START CODE HERE ### (≈ 4 lines of code)

W1 = W1 - dW1 \* learning\_rate

b1 = b1 - db1 \* learning\_rate

W2 = W2 - dW2 \* learning\_rate

b2 = b2 - db2 \* learning\_rate

### END CODE HERE ###

parameters = {"W1": W1,

"b1": b1,

"W2": W2,

"b2": b2}

return parameters

In [27]:# GRADED FUNCTION: nn\_model

def nn\_model(X, Y, n\_h, num\_iterations = 50, print\_cost=False):

np.random.seed(3)

n\_x = layer\_sizes(X, Y)[0]

n\_y = layer\_sizes(X, Y)[2]

cost\_plot= []

# Initialize parameters, then retrieve W1, b1, W2, b2. Inputs: "n\_x, n\_h, n\_y". Outputs = "W1, b1, W2, b2, parameters".

### START CODE HERE ### (≈ 5 lines of code)

parameters = initialize\_parameters(n\_x, n\_h, n\_y)

W1 = parameters["W1"]

W2 = parameters["W2"]

b1 = parameters["b1"]

b2 = parameters["b2"]

### END CODE HERE ###

# Loop (gradient descent)

for i in range(0, num\_iterations):

### START CODE HERE ### (≈ 4 lines of code)

# Forward propagation. Inputs: "X, parameters". Outputs: "A2, cache".

A2, cache = forward\_propagation(X, parameters)

# Cost function. Inputs: "A2, Y, parameters". Outputs: "cost".

cost = compute\_cost(A2, Y, parameters)

# Backpropagation. Inputs: "parameters, cache, X, Y". Outputs: "grads".

grads = backward\_propagation(parameters, cache, X, Y)

# Gradient descent parameter update. Inputs: "parameters, grads". Outputs: "parameters".

parameters = update\_parameters(parameters, grads)

### END CODE HERE ###

# Print the cost every 1000 iterations

if print\_cost and i % 10 == 0:

print ("Cost after iteration %i: %f" %(i, cost))

cost\_plot.append(cost\_plot)

plt.plot(np.squeeze(cost\_plot))

plt.ylabel('cost')

plt.xlabel('iterations')

plt.title("Cost vs Iteration")

plt.show()

return parameters

…

This is where the problems started, when continuing the code using the same layout as of HW.6 the output kept coming out as an error as shown below on figures 2.1 and 2.2.

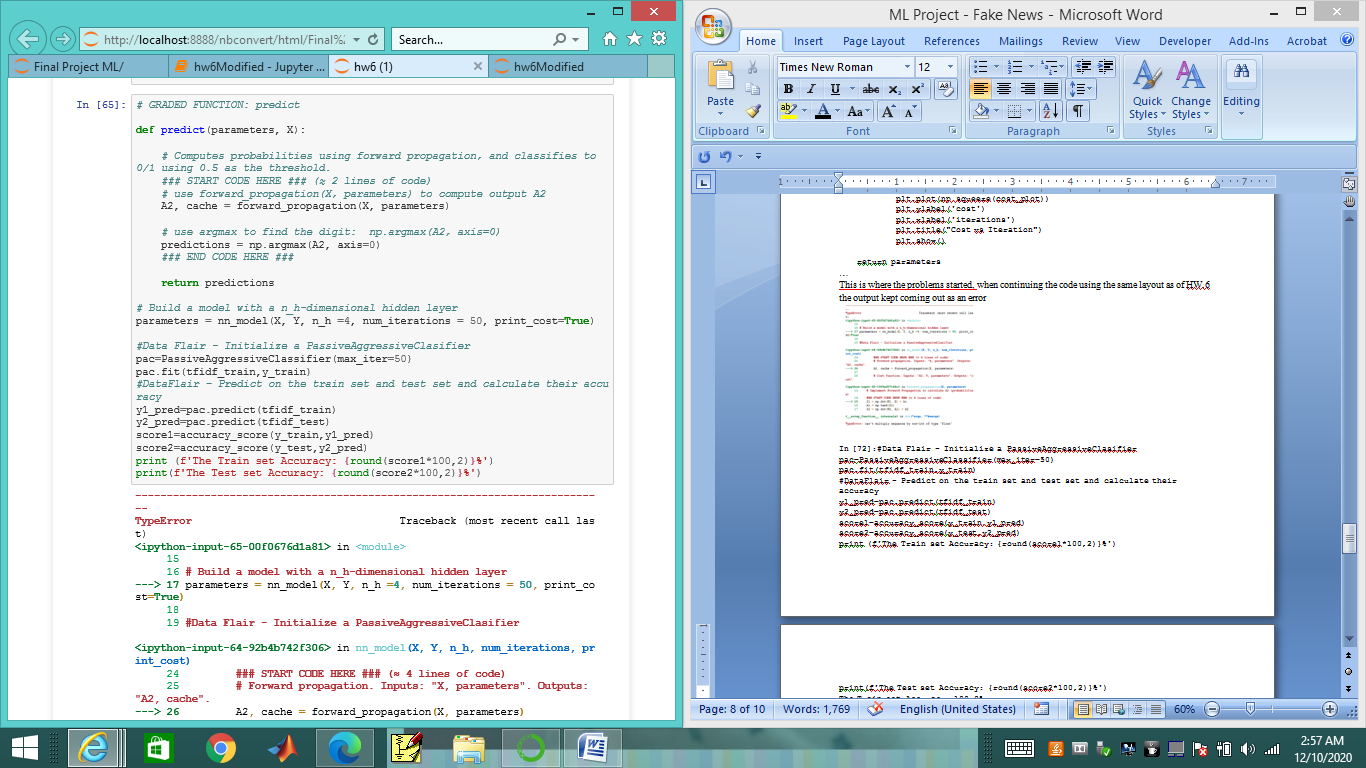
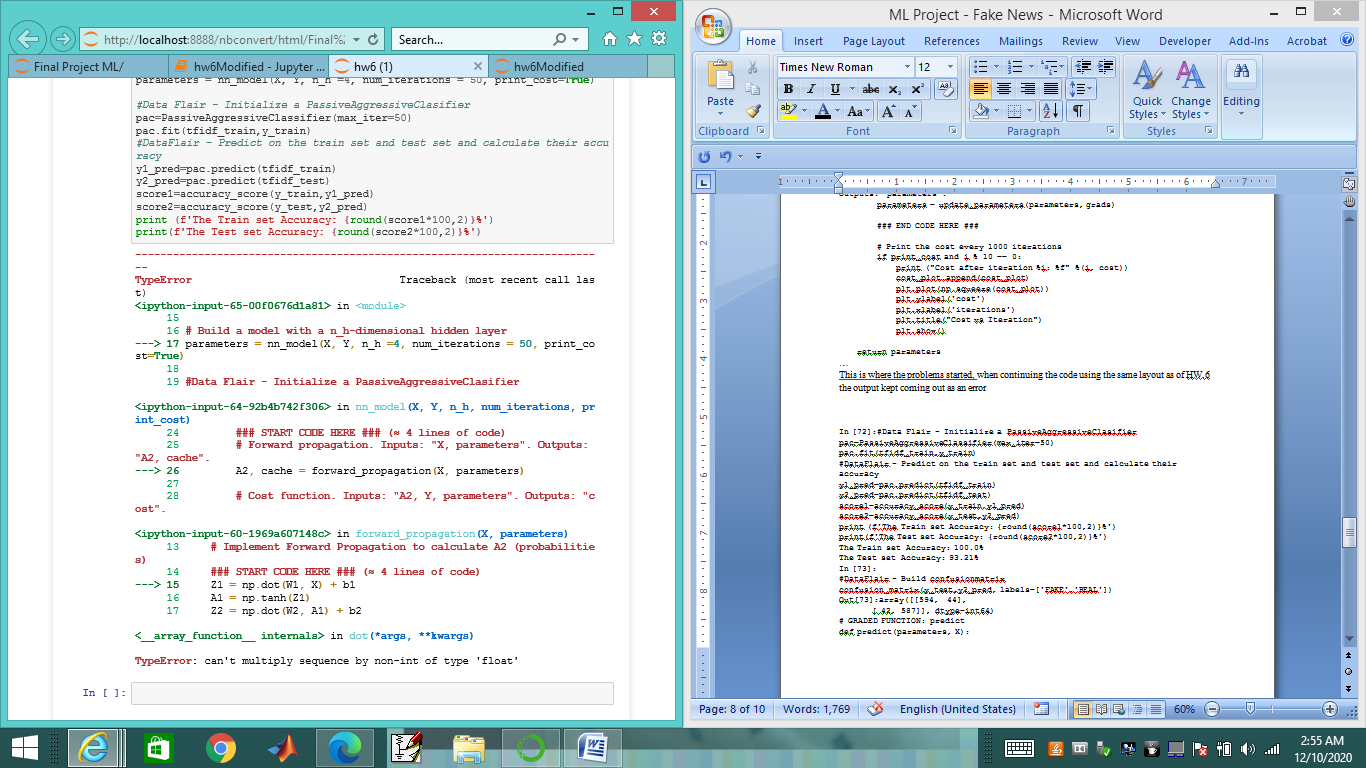


Figure 2.1 – HW.6 code layout

Figure 2.2 – Results for HW.6 code layout

Therefore we ended up going with the original layout as shown below on figure 3.

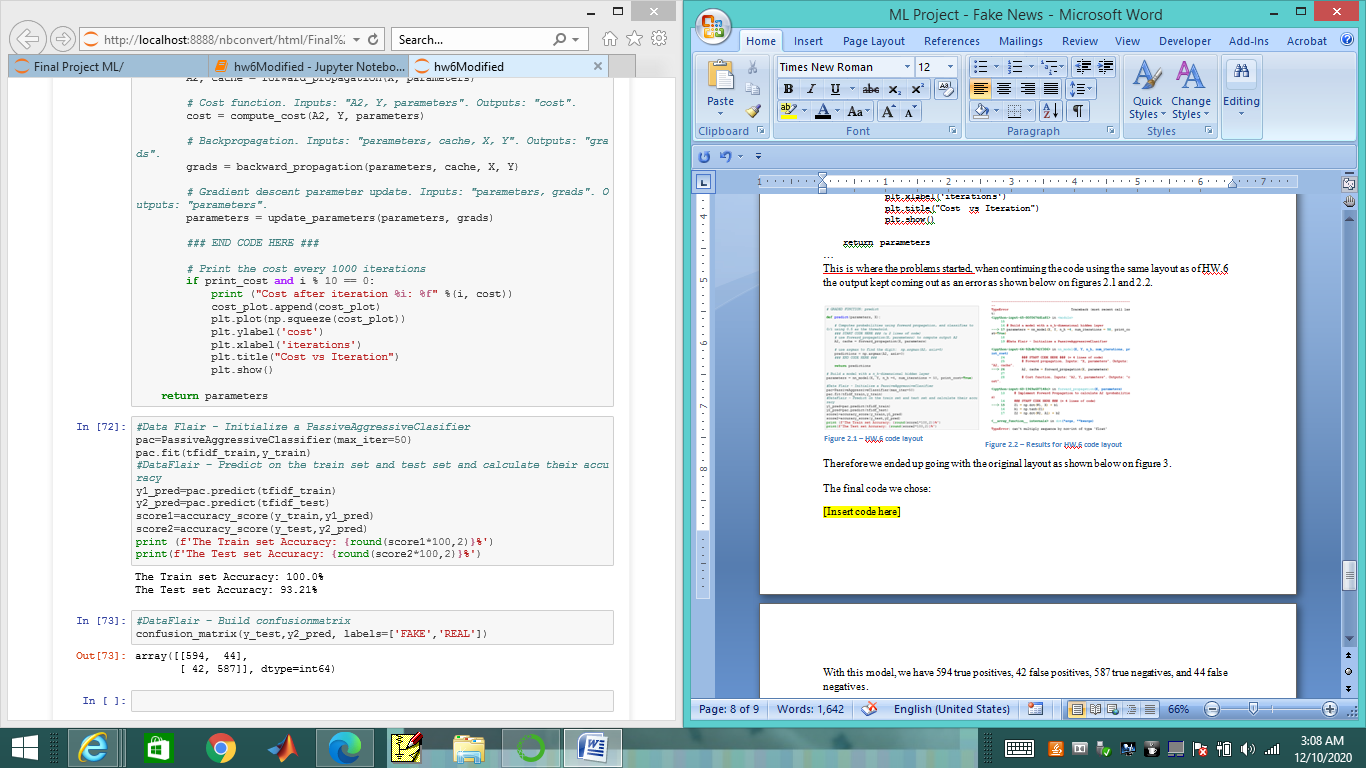


Figure 3

The final code we chose:

In [1]: import numpy as np  
 import pandas as pd  
 import itertools  
 from sklearn.model\_selection import train\_test\_split  
 from sklearn.feature\_extraction.text import TfidfVectorizer  
 from sklearn.linear\_model import PassiveAggressiveClassifier  
 from sklearn.metrics import accuracy\_score, confusion\_matri  
 import matplotlib.pyplot as plt  
 from sklearn.metrics import accuracy\_score

In [2]: #Read the data  
 df=pd.read\_csv('../FinalProject/news.csv')

#Get shape and head  
 df.shape  
 df.head()

Out [2]:

|  | Unnamed: 0 | Title | text | label |
| --- | --- | --- | --- | --- |
| 0 | 8476 | You Can Smell Hillary’s Fear | Daniel Greenfield, a Shillman Journalism Fello... | FAKE |
| 1 | 10294 | Watch The Exact Moment Paul Ryan Committed Pol... | Google Pinterest Digg Linkedin Reddit Stumbleu... | FAKE |
| 2 | 3608 | Kerry to go to Paris in gesture of sympathy | U.S. Secretary of State John F. Kerry said Mon... | REAL |
| 3 | 10142 | Bernie supporters on Twitter erupt in anger ag... | — Kaydee King (@KaydeeKing) November 9, 2016 T... | FAKE |
| 4 | 875 | The Battle of New York: Why This Primary Matters | It's primary day in New York and front-runners... | REAL |

In [3]: #DataFlair - Get the labels  
 labels=df.label  
 labels.head()

Out [3]: 0 FAKE

1 FAKE

2 REAL

3 FAKE

4 REAL

Name: label, dtype: object

In [4]: #DataFlair - Split the dataset

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df['text'], labels, test\_size=0.2, random\_state=7)

In [5]: #DataFlair - Initialize a TfidfVectorizer

tfidf\_vectorizer=TfidfVectorizer(stop\_words='english', max\_df=0.7)

#DataFlair - Fit and transform train set, transform test set

tfidf\_train=tfidf\_vectorizer.fit\_transform(x\_train)

tfidf\_test=tfidf\_vectorizer.transform(x\_test)

In [6]: #DataFlair - Initialize a PassiveAggressiveClassifier  
 pac=PassiveAggressiveClassifier(max\_iter=50)  
 pac.fit(tfidf\_train,y\_train)

#DataFlair - Predict on the test set and calculate accuracy

y\_pred=pac.predict(tfidf\_test)  
 score=accuracy\_score(y\_test,y\_pred)  
 print(f'Accuracy: {round(score\*100,2)}%')

Accuracy: 92.82%

In [7]: #DataFlair - Build confusion matrix

confusion\_matrix(y\_test,y\_pred, labels=['FAKE','REAL'])

Out [7]: array([[590, 48],

[ 43, 586]])

With this model, we have 590 true positives, 43 false positives, 586 true negatives, and 48 false negatives. Also the accuracy ended up being at 92.82% .

**Discussion:**

We weren’t able to fix the issues, and ended up using the code of the original, it uses the libraries:  
 from sklearn.feature\_extraction.text import TfidVectorizer  
 from sklearn.linear\_model import PassiveAggressiveClassifier  
 from sklearn.metrics import accuracy\_score, confusion\_matrix

TfidVectorizer what it does is it converts text to feature vectors, the “Tfid” are the word frequency scores that try to highlight the words that are more interesting, so then the Vectorizer is used together to be able to use it as input. Then PassiveAggresiveClassifier, what it means is the Passive part works in a way that if the prediction is correct, it will keep the model and it won’t do any changes, the Aggressive, what it means is that if the prediction is wrong, it will make changes to the model, and the Classifier is the one that will classify all of this. The accuracy\_score was used to be able to calculate the accuracy with the values of y\_test and y\_pred. After this confusion\_matrix was used and what it does is simply evaluate the accuracy of a classification and it displays it in an array, the numbers inside mean the number of observations known to be in the group with the predicted values.

**Conclusion:**

We learned to detect fake news with Python. We took a dataset (provided by Kaggle), implemented a TfidfVectorizer, initialized a PassiveAggressiveClassifier, and fit our model. We ended up obtaining an accuracy of 92.98% in magnitude. Even though we weren’t able to fix the issues we had, we can say we learned a lot from this project because by making mistakes is the way to truly learn. We analyzed line by line and searched the web a lot to try to fix the issues. So in the end we were left feeling confident with what we ended up and we were able to understand more of the code and also learned new things from the functions used in the code of the site because it used new ones that we didn’t understand until we started to search what each was and what it did. This project was challenging for us and made us learn a lot more.

**References:**

[Top 47 Machine Learning Projects for 2020 [Source Code Included] - DataFlair (data-flair.training)](https://data-flair.training/blogs/machine-learning-project-ideas/)

https://www.kaggle.com/hassanamin/textdb3

Microsoft Word – Homework 6, chapter 6 (provided from the professor)