

# Assignment 5

Name – Sati, Ankit

Date – 2/13/2022

Section – 001

SID – as14128

Total in points (Maximum 100 points)–

Professors Comments –

## PART 1

Get familiar with Machine Learning using Spark; execute and document the following exercise

### Step 1 –

This is like assignment 3 where we need to create a new cluster. All the required services are the part of the HDI cluster and we need to push the ML doc as mentioned in the tutorial.

Create HDInsight cluster ...

Basics Storage Security + networking Configuration + pricing Tags Review + create

New to HDInsight? Get started with our [training resources](#).  
Create a managed HDInsight cluster. Select from Spark, Kafka, Hadoop, Storm, and more. [Learn more](#)

**Project details**  
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* Azure for Students  
Resource group \* (New) cclab5 Create new

**Cluster details**  
Name your cluster, pick a region, and choose a cluster type and version. [Learn more](#)

Cluster name \* lab5  
Region \* East US  
Availability zone ⓘ  
Cluster type \* Spark Change  
Version \* Spark 3.1 (HDI 4.0)

**Cluster credentials**  
Enter new credentials that will be used to administer or access the cluster.

Cluster login username \* admin  
Cluster login password \* .....  
Confirm cluster login password \* .....  
Secure Shell (SSH) username \* sshuser  
Use cluster login password for SSH

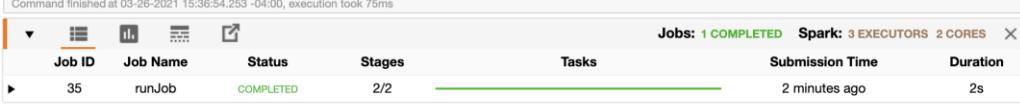
**Step 2 –** This uses logistic regression. Suppose you have a set of feature vectors  $x_i \in R^n$  for  $i \in [0, m]$ . Associated with each feature vector we have a binary result  $y_i$ . We are interested in the probability  $P(y=1|x)$  which we write as the function  $p(x)$ . However because  $p(x)$  is between 0 and 1 it is not expressible as a linear function of  $x$  so we can't use regular linear regression, so we look at the odds expression  $p(x)/(1-p(x))$  and make the guess that its log is linear. In other words

$$\ln(p(x)/(1-p(x))) = b_0 + b \cdot x$$

```
#inspections = spark.sparkContext.textFile('wasb:///HdiSamples/HdiSamples/FoodInspectionData/Food_Inspections1.csv')\
#          .map(csvParse)

inspections = spark.sparkContext.textFile('Food_Inspection.csv')\
          .map(csvParse)
```

It is used frequently in machine learning to map a real number into a probability range [0, 1].

```
In [1]: sc
Command finished at 03-26-2021 15:36:54.253 -04:00, execution took 75ms


Jobs: 1 COMPLETED Spark: 3 EXECUTORS 2 CORES



| ID | YARN Application ID            | Kind    | State | Spark UI             | Driver log           | Current session? |
|----|--------------------------------|---------|-------|----------------------|----------------------|------------------|
| 7  | application_1616777260895_0011 | pyspark | idle  | <a href="#">Link</a> | <a href="#">Link</a> | ✓                |



SparkSession available as 'spark'.  
<SparkContext master=yarn appName=remotesparkmagics>


In [2]: from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
Command finished at 03-26-2021 15:36:54.531 -04:00, execution took 261ms
In [3]: def csvParse(s):
    import csv
    from StringIO import StringIO
    sio = StringIO(s)
    value = csv.reader(sio).next()
    sio.close()
    return value
Command finished at 03-26-2021 15:36:54.601 -04:00, execution took 48ms
```

### Step – 3

If we do this another way, we can ask how accurate are we in finding the failing restaurants? This is a bit harder because there are far fewer of them. In this case we are interested in true-negatives, so

Precision is the probability that a (randomly selected) negative prediction is correct.

```
df.show(5)
+---+-----+-----+-----+
| id |      name |   results | violations |
+---+-----+-----+-----+
|1978294|KENTUCKY FRIED CH...|      Pass|32. FOOD AND NON-...|
|1978279|          SOLO FOODS|Out of Business|           |
|1978275|SHARKS FISH & CHI...|      Pass|34. FLOORS: CONST...|
|1978268|CARNITAS Y SUPERM...|      Pass|33. FOOD AND NON-...|
|1978261|          WINGSTOP|      Pass|           |
+---+-----+-----+-----+
only showing top 5 rows

print("passing = %d"%df[df.results == 'Pass'].count())
print("failing = %d"%df[df.results == 'Fail'].count())
passing = 61204
failing = 6625
```

### Step 4-

Precision and recall are then defined as:

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

Precision is the probability that a (randomly selected) positive prediction is correct.

Recall is the probability that a (randomly selected) restaurant with a passing grade is predicted to be passing.

The problem here is to predict the grade (Passing or Failing) of City of Chicago restaurant inspections based on the notes made by the inspector.

This uses logistic regression. Suppose you have a set of feature vectors  $x_i \in R^n$  for  $i$  in  $[0, m]$ . Associated with each feature vector we have a binary result  $y_i$ . We are interested in the probability  $P(y = 1|x)$  which we write as the function  $p(x)$ . However because  $p(x)$  is between 0 and 1 it is not expressible as a linear function of  $x$  so we can't use regular linear regression, so we look at the odds expression  $p(x)/(1 - p(x))$  and make the guess that its log is linear. In other words

$$\ln\left(\frac{p(x)}{1 - p(x)}\right) = b_0 + b \cdot x$$

where the offset  $b_0$  and the vector  $b = [b_1, b_2, \dots, b_n]$  define a hyperplane for linear regression. solving this for  $p(x)$  we get

$$p(x) = \frac{1}{1 + e^{-(b_0 + b \cdot x)}}$$

And we say  $y = 1$  if  $p(x) > 0$  otherwise it is zero. Unfortunately finding the best  $b_0$  and  $b$  is not as easy as straight linear regression, but simple Newton like iterations will converge to good solutions.

We note that the logistic function  $\sigma(t)$  is defined as follows:

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

It is used frequently in machine learning to map a real number into a probability range  $[0, 1]$ .

```
#sc = pyspark.SparkContext('local[*]')
sc
```

**SparkContext**

[Spark UI](#)

**Version**

v3.2.1

**Master**

local[\*]

**AppName**

PySparkShell

```
[1]: from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
```

```
[2]: def csvParse(s):
    import csv
    from StringIO import StringIO
    sio = StringIO(s)
    value = csv.reader(sio).next()
    sio.close()
    return value
```

## The version in this notebook uses a slightly different input file from the one in the Azure HDInsight demo.

This notebook will run on spark on your laptop.

```
[3]: #inspections = spark.sparkContext.textFile('wasb:///HdiSamples/HdiSamples/FoodInspectionData/Food_Inspections1.csv')\
#          .map(csvParse)

inspections = spark.sparkContext.textFile('/users/dennisgannon/OneDrive/Docs7/Food_Inspections1.csv')\
          .map(csvParse)
```

```
[4]: inspections.count()
```

```
[4]: 103994
```

```
[5]: inspections.take(10)
```

```
[5]: [[1978294,
      'KFC',
      'KENTUCKY FRIED CHICKEN',
      'Pass',
      '32. FOOD AND NON-FOOD CONTACT SURFACES PROPERLY DESIGNED, CONSTRUCTED AND MAINTAINED - Comments: OBSERVED THE FLOOR BIN BROKEN/CRACKED AT CHICKEN FRIED PREP AREA, INSTRUCTED TO REPLACE. | 33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETERGENTS - Comments: OBSERVED THE EXTERIOR OF THE FRYER, OVEN NOT CLEAN, INSTRUCTED TO CLEAN. ALSO CLEAN AND SANITIZE REAR CHICKEN PREP TABLE. | 35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: OBSERVED THE CEILING IN THE CHICKEN COOLER, NOT CLEAN, DUSTY. INSTRUCTED TO CLEAN. | 36. LIGHTING: REQUIRED MINIMUM FOOT-CANDLES OF LIGHT PROVIDED, FIXTURES SHIELDED - Comments: OBSERVED BROKEN LIGHT SHIELD IN LADIES RESTROOM, INSTRUCTED TO REPLACE. | 21. * CERTIFIED FOOD MANAGER ON SITE WHEN POTENTIALLY HAZARDOUS FOODS ARE PREPARED AND SERVED - Comments: VIOLATION CORRECTION'],
```

```
[6]: df.show(5)
```

id	name	results	violations
1978294	KENTUCKY FRIED CHICKEN	Pass	32. FOOD AND NON-FOOD CONTACT SURFACES PROPERLY DESIGNED, CONSTRUCTED AND MAINTAINED - Comments: OBSERVED THE FLOOR BIN BROKEN/CRACKED AT CHICKEN FRIED PREP AREA, INSTRUCTED TO REPLACE.   33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETERGENTS - Comments: OBSERVED THE EXTERIOR OF THE FRYER, OVEN NOT CLEAN, INSTRUCTED TO CLEAN. ALSO CLEAN AND SANITIZE REAR CHICKEN PREP TABLE.   35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: OBSERVED THE CEILING IN THE CHICKEN COOLER, NOT CLEAN, DUSTY. INSTRUCTED TO CLEAN.   36. LIGHTING: REQUIRED MINIMUM FOOT-CANDLES OF LIGHT PROVIDED, FIXTURES SHIELDED - Comments: OBSERVED BROKEN LIGHT SHIELD IN LADIES RESTROOM, INSTRUCTED TO REPLACE.   21. * CERTIFIED FOOD MANAGER ON SITE WHEN POTENTIALLY HAZARDOUS FOODS ARE PREPARED AND SERVED - Comments: VIOLATION CORRECTION'
1978279	SOLO FOODS	Out of Business	
1978275	SHARKS FISH & CHICKEN	Pass	34. FLOORS: CONSTRUCTION, MAINTENANCE, AND FINISHES - Comments: OBSERVED THE FLOOR IN THE KITCHEN AREA NOT CLEAN, DUSTY. INSTRUCTED TO CLEAN.   35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: OBSERVED THE CEILING IN THE CHICKEN COOLER, NOT CLEAN, DUSTY. INSTRUCTED TO CLEAN.   36. LIGHTING: REQUIRED MINIMUM FOOT-CANDLES OF LIGHT PROVIDED, FIXTURES SHIELDED - Comments: OBSERVED BROKEN LIGHT SHIELD IN LADIES RESTROOM, INSTRUCTED TO REPLACE.   21. * CERTIFIED FOOD MANAGER ON SITE WHEN POTENTIALLY HAZARDOUS FOODS ARE PREPARED AND SERVED - Comments: VIOLATION CORRECTION'
1978268	CARNITAS Y SUPERMEXICAN	Pass	33. FOOD AND NON-FOOD CONTACT EQUIPMENT - Comments: OBSERVED THE EXTERIOR OF THE FRYER, OVEN NOT CLEAN, INSTRUCTED TO CLEAN. ALSO CLEAN AND SANITIZE REAR CHICKEN PREP TABLE.   35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: OBSERVED THE CEILING IN THE CHICKEN COOLER, NOT CLEAN, DUSTY. INSTRUCTED TO CLEAN.   36. LIGHTING: REQUIRED MINIMUM FOOT-CANDLES OF LIGHT PROVIDED, FIXTURES SHIELDED - Comments: OBSERVED BROKEN LIGHT SHIELD IN LADIES RESTROOM, INSTRUCTED TO REPLACE.   21. * CERTIFIED FOOD MANAGER ON SITE WHEN POTENTIALLY HAZARDOUS FOODS ARE PREPARED AND SERVED - Comments: VIOLATION CORRECTION'
1978261	WINGSTOP	Pass	

only showing top 5 rows

```
[7]: print("passing = %d"%df[df.results == 'Pass'].count())
print("failing = %d"%df[df.results == 'Fail'].count())
```

```
[7]: passing = 61204
failing = 20225
```

```

count_results_df = spark.sql("SELECT results, COUNT(results) AS cnt FROM \
    CountResults GROUP BY results").toPandas()

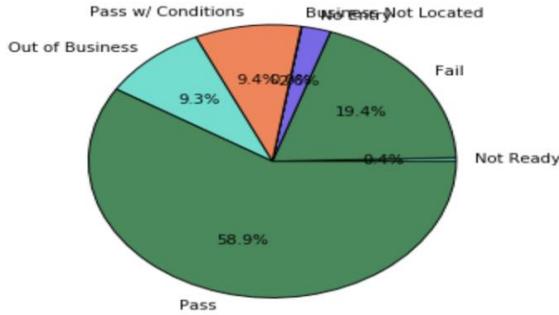
5]: count_results_df
5]:
```

	results	cnt
<b>0</b>	Not Ready	429
<b>1</b>	Fail	20225
<b>2</b>	No Entry	2678
<b>3</b>	Business Not Located	44
<b>4</b>	Pass w/ Conditions	9725
<b>5</b>	Out of Business	9689
<b>6</b>	Pass	61204

```

5]: %matplotlib inline
import matplotlib.pyplot as plt

labels = count_results_df['results']
sizes = count_results_df['cnt']
colors = ['turquoise', 'seagreen', 'mediumslateblue', 'palegreen', 'coral']
plt.pie(sizes, labels=labels, autopct='%.1f%%', colors=colors)
plt.axis('equal')
5]: (-1.0101138270539849, 1.0, -1.0114322486525658, 1.0040925495551682)
```



Precision and recall are then defined as:

$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$

Precision is the probability that a (randomly selected) positive prediction is correct.

Recall is the probability that a (randomly selected) restaurant with a passing grade is predicted to be passing.

```
2]: print('so precision = %f' % \
         (float(true_positive['cnt'])/(float(true_positive['cnt'])+float(false_positive['cnt']))))

so precision = 0.911911

3]: print('and recall = %f' % \
         (float(true_positive['cnt'])/(float(true_positive['cnt'])+float(false_negative['cnt']))))

and recall = 0.971670
```

If we do this another way, we can ask how accurate are we in finding the failing restaurants? This is a bit harder because there are far fewer of them. In this case we are interested in true-negatives, so

Precision is the probability that a (randomly selected) negative prediction is correct.

Recall is the probability that a (randomly selected) restaurant with a failing grade is predicted to be failing.

$$\text{Precision} = \frac{tn}{tn + fn}$$
$$\text{Recall} = \frac{tn}{tn + fp}$$

```
4]: print('so the precision of failure prediction = %f' % \
         (float(true_negative['cnt'])/(float(true_negative['cnt'])+float(false_negative['cnt']))))

so the precision of failure prediction = 0.870579

5]: print('and recall is = %f' % \
         (float(true_negative['cnt'])/(float(true_negative['cnt'])+float(false_positive['cnt']))))

and recall is = 0.670001
```

```
In [26]: %%sql -q -o true_positive
SELECT count(*) AS cnt FROM Predictions WHERE prediction = 0 AND results = 'Fail'

Command finished at 03-26-2021 17:53:00.308 -04:00, execution took 3s 289ms


| Job ID | Job Name | Status    | Stages | Tasks | Jobs: 1 COMPLETED | Spark: 3 EXECUTORS 6 CORES | X        |
|--------|----------|-----------|--------|-------|-------------------|----------------------------|----------|
| 34     | runJob   | COMPLETED | 2/2    |       |                   | Submission Time            | Duration |


4 minutes ago 2s

In [27]: true_negative = spark.sql("SELECT count(*) AS cnt FROM Predictions WHERE \
                                 (prediction = 0 AND results = 'Fail')").toPandas()

Command finished at 03-26-2021 17:53:03.605 -04:00, execution took 3s 286ms


| Job ID | Job Name | Status    | Stages | Tasks | Jobs: 1 COMPLETED | Spark: 3 EXECUTORS 6 CORES | X        |
|--------|----------|-----------|--------|-------|-------------------|----------------------------|----------|
| 35     | toPandas | COMPLETED | 2/2    |       |                   | Submission Time            | Duration |


4 minutes ago 2s

In [28]: %%sql -q -o false_positive
SELECT count(*) AS cnt FROM Predictions \
WHERE prediction = 0 AND (results = 'Pass' OR results = 'Pass w/ Conditions')

Command finished at 03-26-2021 17:53:09.801 -04:00, execution took 2s 274ms

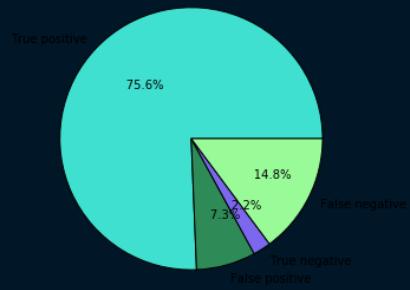

| Job ID | Job Name | Status    | Stages | Tasks | Jobs: 1 COMPLETED | Spark: 3 EXECUTORS 6 CORES | X        |
|--------|----------|-----------|--------|-------|-------------------|----------------------------|----------|
| 36     | runJob   | COMPLETED | 2/2    |       |                   | Submission Time            | Duration |


4 minutes ago 1s
```

```
labels = ['True positive', 'False positive', 'True negative', 'False negative']
sizes = [true_positive['cnt'], false_positive['cnt'], false_negative['cnt'], true_negative['cnt']]
colors = ['turquoise', 'seagreen', 'mediumslateblue', 'palegreen', 'coral']
plt.pie(sizes, labels=labels, autopct='%10.1f%%', colors=colors)
plt.axis('equal')
```

Python

```
(-1.0045326948165894, 1.0, -1.0032166242599487, 1.0048178434371948)
```



## PART 2

Get familiar with Machine Learning using Azure ML; execute and document the following exercise

**Step 1** - This notebook illustrates invoking the webservice for the three way classifier with the Arvix data. The three way example uses three classifiers: a multiclass neural net, a multiclass logistic regression and a random forest classifier. The three way classifier uses a popularity vote system. If any two of the classifiers agree that is the main reply of the three way. the third case is always put forward as a second choice.

### Azure ML service sample.

This notebook illustrates invoking the webservice for the three way classifier with the Arvix data. The three way example uses three classifiers: a multiclass neural net, a multiclass logistic regression and a random forest classifier. The three way classifier uses a popularity vote system. If any two of the classifiers agree that is the main reply of the three way. the third case is always put forward as a second choice.

The python code for the three way python module is here

```
import pandas as pd
labels = { 1: "'bio'", 2: "'compsci'", 3: "'finance'", 4: "'math'"}
#The entry point function can contain up to two input arguments:
#Param: a pandas.DataFrame
#Param: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):

    tclass = dataframe1["Col1"]
    scored1 = dataframe1["Scored Labels"]
    scored2 = dataframe2["Scored Labels"]
    scored3 = dataframe2["Scored Labels (2)"]
    scored = []
    second = []
    lclass = []
    for i in range(0, len(tclass)):
        lclass.extend([tclass[i]])
        if scored2[i] == scored3[i]:
            scored.extend([labels[scored1[i]]])
            second.extend([labels[scored1[i]]])
        elif scored2[i] == scored1[i]:
            scored.extend([labels[scored1[i]]])
            second.extend([labels[scored3[i]]])
        elif scored1[i] == scored3[i]:
            scored.extend([labels[scored1[i]]])
            second.extend([labels[scored2[i]]])
        else:
            scored.extend([labels[scored1[i]]])
            second.extend([labels[scored3[i]]])

    data = {'Col1': lclass, 'Scored Labels': scored, 'second': second}
    df = pd.DataFrame(data, columns=['Col1', 'Scored Labels', 'second'])
    return df
```

```
: import sys
import azure
import socket
import csv
import urllib2
import json
import pandas as pd
import numpy as np
import urllib
import json
import pickle
import unicodedata
```

: The following establishes the urls and keys for each of the three classifiers

```
: url = []
api_key = []
# the first is the neural net version
url.append('https://ussouthcentral.services.azureml.net/workspaces/38d1439e6956413e9fe2950b6530c117/services/281bbba94a
api_key.append('5dtOV9rJgoI8LtnX4N7vNMfhW4vdSPHcpCq1IUNQyTPTSwzIHQ4NADF1KkKvI/Q4mJSVFQh7JQcvyVxk7yuG/Q==')
# the second is the logistic regression version
url.append('https://ussouthcentral.services.azureml.net/workspaces/38d1439e6956413e9fe2950b6530c117/services/08e77a94e7
api_key.append('RKhxsJXpnBFv3CVeL5B927VlKgUbQZBaHp8BbQ2QeMgPkCqUg+W9QY9RanTO3zMNxTywpQBX+db5+n6xxWkugA==')
# this is the url and key for the threeway version
url.append('https://ussouthcentral.services.azureml.net/workspaces/38d1439e6956413e9fe2950b6530c117/services/462fcc1e3
api_key.append('e7iaNA+Pi9KdKZ70+vV7JvZ81E3vKfqQShFuMf+a89GfGrixAHNq4HaHJZhpcY3TBujJmJh3hfKNtoYY3eKvnw==')
```

```

: def sendrequest(datalist, url, api_key):
    #datalist is a list ["class", "document", "title"]
    data = {
        "Inputs": {
            "input1": [
                {
                    "ColumnNames": ["Col1", "Col2", "Col3"],
                    "Values": [ datalist ]
                },
                ],
            "GlobalParameters": { }
        }
    }

    body = str.encode(json.dumps(data))

    headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ api_key)}

    req = urllib2.Request(url, body, headers)

    try:
        response = urllib2.urlopen(req)

        # If you are using Python 3+, replace urllib2 with urllib.request in the above code:
        # req = urllib.request.Request(url, body, headers)
        # response = urllib.request.urlopen(req)

        result = response.read()
        #print(result)
        y = json.loads(result)
        return(y["Results"]["output1"]["value"]["Values"])

    except urllib2.HTTPError, error:
        print("The request failed with status code: " + str(error.code))

    # Print the headers - they include the request ID and the timestamp, which are useful for debugging the failure

```

```

: #this function is used to send multiple requests to the web service with one invocation.
def sendbulkrequest(datalist, url, api_key):
    #datalist is a list ["class", "document", "title"]
    data = {
        "Inputs": {

            "input1": [
                {
                    "ColumnNames": ["Col1", "Col2", "Col3"],
                    "Values": datalist
                },
                ],
            "GlobalParameters": { }
        }
    }

    body = str.encode(json.dumps(data))
    headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ api_key)}

    req = urllib2.Request(url, body, headers)

    try:
        response = urllib2.urlopen(req)

        # If you are using Python 3+, replace urllib2 with urllib.request in the above code:
        # req = urllib.request.Request(url, body, headers)
        # response = urllib.request.urlopen(req)

        result = response.read()
        y = json.loads(result)
        return(y["Results"]["output1"]["value"]["Values"])

    except urllib2.HTTPError, error:
        print("The request failed with status code: " + str(error.code))

    # Print the headers - they include the request ID and the timestamp, which are useful for debugging the failure
    print(error.info())

    return json.loads(error.read())

```

```

        return json.loads(error.read())

def read_azure_blob(subj, base):
    #base is the url for the azure blob store
    docpath = base + "/" + subj + ".csv"
    response = urllib2.urlopen(docpath)
    data = csv.reader(response)
    biglist = []
    for row in data:
        biglist.append(row)
    return biglist

#first read the test data file arxivnophy
base = "https://scimldata.blob.core.windows.net/arxiv"
nophytable = read_azure_blob('arxivnophy', base)

```

next test it with the first entry in the table with the three way.

```

x = sendrequest(nophytable[0], url[2], api_key[2])
print x

[[u"'compsci'", u"'compsci'", u"'compsci'"]]

len(x)

1

multx = sendbulkrequest(nophytable, url[2], api_key[2])

```

Let's see how well the three way does if we ask if either the first or second choice is correct.

```

correct = {"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0}
num = {"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0}

5]: correct = {"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0}
for i in range(len(multx)):
    x = multx[i]
    key = x[0]
    num[key] = num[key]+1.0
    if (key==x[1]) or (key==x[2]):
        correct[key] = correct[key] +1.0
    else:
        for key in correct:
            correct[key] = 100.0*correct[key]/num[key]
print num
print correct

{"'bio': 316.0, "'finance': 232.0, "'compsci': 648.0, "'math': 742.0}
{"'bio': 65.82278481012658, "'finance': 60.3448275862069, "'compsci': 72.37654320987654, "'math': 88.274932614555
25}

7]: def make_confusion_table(multx):
    tble = {"'compsci':{"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0},
            "'finance':{"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0},
            "'bio':{"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0},
            "'math':{"'compsci': 0, "'finance': 0, "'bio': 0, "'math': 0}
    predicted = []
    tclass = []
    for e in multx:
        x = e[0]
        tclass.extend([x])
        sc = e[1]
        predicted.extend([sc])
    for i in range(0, len(tclass)):
        if tclass[i] == 'none':
            print "found none"
        else:
            tble[predicted[i]][tclass[i]] = tble[predicted[i]][tclass[i]] +1
    names = ["'bio'", "'compsci'", "'finance'", "'math'"]
    for x in names:
        total = 0
        for y in names:
            total = total + tble[y][x]
        print "total for " + x + "=" +str(total)
        for y in names:
            tble[y][x] = 100.0*tble[y][x]/(1.0*total)
    df = pd.DataFrame(tble, columns=[ "'bio'", "'compsci'", "'finance'", "'math'"])
    return df

```

```

df = make_confusion_table(multx)
df

total for 'bio'= 316
total for 'compsci'= 648
total for 'finance'= 232
total for 'math'= 742

'bio' 'compsci' 'finance' 'math'
-----  

'bio' 50.316456 20.886076 0.949367 27.848101  

'compsci' 4.938272 62.654321 1.543210 30.864198  

'finance' 5.603448 9.913793 47.844828 36.637931  

'math' 3.908356 13.477089 2.291105 80.323450

#next do the confusion matrix for the neural net classifier
multx2 = sendbulkrequest(nophytable, url[0], api_key[0])

df = make_confusion_table(multx2)
df

total for 'bio'= 316
total for 'compsci'= 648
total for 'finance'= 232
total for 'math'= 742

'bio' 'compsci' 'finance' 'math'
-----  

'bio' 51.265823 19.936709 4.746835 24.050633  

'compsci' 10.493827 57.716049 4.320988 27.469136  

'finance' 6.465517 17.241379 50.431034 25.862069  

'math' 6.469003 16.037736 5.525606 71.967655

```

The code below assumes that the test file is store in Azure blob store as a csv file. It uses the three way classifier and the neural net classifier and the logistic regression classifier to compare the confusion matrix of each.

```

total for 'math'= 742

'bio' 'compsci' 'finance' 'math'
-----  

'bio' 51.265823 19.936709 4.746835 24.050633  

'compsci' 10.493827 57.716049 4.320988 27.469136  

'finance' 6.465517 17.241379 50.431034 25.862069  

'math' 6.469003 16.037736 5.525606 71.967655

# now for the logistic regression service
multx3 = sendbulkrequest(nophytable, url[1], api_key[1])

df = make_confusion_table(multx3)
df

total for 'bio'= 316
total for 'compsci'= 648
total for 'finance'= 232
total for 'math'= 742

'bio' 'compsci' 'finance' 'math'
-----  

'bio' 54.113924 23.417722 0.316456 22.151899  

'compsci' 6.172840 65.740741 1.234568 26.851852  

'finance' 5.172414 9.913793 43.534483 41.379310  

'math' 3.504043 14.555256 2.021563 79.919137

```

as we can see, all three are rather poor. the three way is only slightly better.

## PART 3

Get familiar with Machine Learning using Amazon ML; execute and document the following exercise

## Running the notebook on the amazon sagemaker service

The documentation for this one is a bit sparse and the example is really just of a demo for how easy it is to use their “Simple Network Builder” to define a LSTM network and train it with stochastic gradient decent on data from the Penn Treebank Project. One command starts the learning:

Amazon SageMaker Studio File Edit View Run Kernel Git Tabs Settings Help

mxnet.ipynb x rnn-lstm.ipynb x

2 vCPU + 4 GiB Cluster Python 3 (Data Science) Share

## Load and Run a LSTM model

One of the many good examples in CNTK is language modeling exercise in Examples/Text/PennTreebank. The documentation for this one is a bit sparse and the example is really just of a demo for how easy it is to use their "Simple Network Builder" to define a LSTM network and train it with stochastic gradient decent on data from the Penn Treebank Project. One command starts the learning:

```
cntk configFile=../Config/rnn.cntk
```

Doing so trains the network, tests it and saves the model. However, to see the model data in an easily readable form you need a trivial addition to the configfile: you need to add the following dumpnode command to put a dump file a directory of your choosing.

```
dumpnode=[  
    action = "dumpnode"  
    modelPath = "$ModelDir$/rnn.dnn"  
    outputFile = "$OutputDir$/modeltext/dump"  
]
```

However we have already run it and the model data is in a ziped file here.  
<https://1drv.ms/u/sAkRG92LIOUagsYYVxAGC4HJL8a3W>

This file is about 50MB so place it in a directory called models. This program does not require CNTK to run.

```
[9]:  
import numpy as np  
import numpy.linalg as la  
import math  
import pwd  
  
/root
```

The following is a utility to pull the word from a line in the vocab file.

```
[14]: modelpath= "/root/"
```

```
[15]: def pullword():  
    i = 0  
    while l[i] == ' ' or l[i] == '\t': i+=1  
    while l[i] != ' ' and l[i] != '\t': i+=1  
    while l[i] == ' ' or l[i] == '\t': i+=1  
    while l[i] != ' ' and l[i] != '\t': i+=1  
    while l[i] == ' ' or l[i] == '\t': i+=1  
    strm = ""  
    while l[i] != ' ' and l[i] != '\t':  
        strm = strm+l[i]  
        i += 1  
    return strm
```

OpenTensor is a function that opens the trained model files generated by cntk.

```
[16]: def opentensor(path):  
    with open(path) as file:  
        El = [[float(digit) for digit in line.split()] for line in file]  
    return np.array(El)
```

```
[17]: E = opentensor(modelpath + '/E0.txt')
```

Doing so trains the network, tests it and saves the model. However, to see the model data in an easily readable form you need a trivial addition to the configfile: you need to add the following dumpnode command to put a dump file a directory of your choosing.

the output vector of the rnn is a vector of length 10000. `output[i]` represents the relative likelihood that the next word is the best to follow the string so far. `Getwordsfromoutput` returns the top 5 candidate words.

The screenshot shows a Jupyter Notebook interface. On the left is a file browser titled 'rnn\_modeldata' containing various text files. On the right is a code cell titled 'Load and Run a LSTM model'. The code cell contains Python code to define a `dumpnode` function and a link to a pre-trained model file. Below the code cell, there is explanatory text and a note about the file size. The code cell itself contains three lines of code:

```
[1]: import numpy as np
import numpy.linalg as la
import math

The following is a utility to pull the word from a line in the vocab file.

[2]: modelpath = "path to your models"

[3]: def pullword(l):
    i = 0
```

Mode: Command | Ln 1, Col 4 | rnn-lstm.ipynb

The screenshot shows a Jupyter Notebook interface. On the left is a file browser titled 'rnn\_modeldata' containing various text files. On the right is a code cell containing Python code for reading a trained model file. The code defines a `pullword` function and an `opentensor` function. It then reads the 'E0.txt' file from the 'rnn\_modeldata' directory. The code cell contains several lines of code:

```
[3]: def pullword(l):
    i = 0
    while l[i] == ' ' or l[i] == '\t': i+=1
    while l[i] != ' ' and l[i] != '\t': i+=1
    while l[i] == ' ' or l[i] == '\t': i+=1
    while l[i] != ' ' and l[i] != '\t': i+=1
    while l[i] == ' ' or l[i] == '\t': i+=1
    strm = ""
    while l[i] != ' ' and l[i] != '\t':
        strm = strm+l[i]
        i += 1
    return strm

OpenTensor is a function that opens the trained model files generated by cntk.

[26]: def opentensor(path):
    with open(path) as file:
        El = [[float(digit) for digit in line.split()] for line in file]
    return np.array(El)

[23]: with open('/home/jovyan/work/rnn_modeldata/E0.txt') as file:
    El = [[float(digit) for digit in line.split()] for line in file]

[25]: np.array(El)

[25]: array([-0.1362291, -0.04745213, -0.54384276, ..., -0.32357392,
       0.16390016, 0.144616 ], [ 0.19129534, 0.13582509, 0.45247782, ..., 0.08192065,
       0.01278295, -0.37417376], [ 0.10613937, -0.10509791, -0.00999349, ..., -0.19641586,
       -0.14362586, 0.20618897], ... [-0.09360378, 0.20040689, 0.33530681, ..., -0.06467007,
       -0.09984568, -0.08167856], [ 0.01179735, -0.32895323, -0.06993713, ..., -0.08004401,
```

Mode: Command | Ln 1, Col 4 | rnn-lstm.ipynb

Filter files by name

Name	Last Modified
bc0.txt	a day ago
b0.txt	a day ago
bi0.txt	a day ago
bo0.txt	a day ago
cls2idx.txt	a day ago
E0.txt	a day ago
vocab.txt	a day ago
W2-new.txt	a day ago
W2.txt	a day ago
WCFO.txt	a day ago
WCIO.txt	a day ago
WC00.txt	a day ago
weightforclass.txt	a day ago
WHCO.txt	a day ago
WHFO.txt	a day ago
WHI.txt	a day ago
WHO.txt	a day ago
word2cls.txt	a day ago
WXC.txt	a day ago
WXF.txt	a day ago
WXIO.txt	a day ago
WXO.txt	a day ago

```
[29]: E = opentensor(modelpath + '/home/jovyan/rnn_modeldata/E0.txt')
b0 = opentensor('/home/jovyan/work/rnn_modeldata/b0.txt')
WH0 = opentensor('/home/jovyan/work/rnn_modeldata/WH0.txt')
WC0 = opentensor('/home/jovyan/work/rnn_modeldata/WC00.txt')
WXF = opentensor('/home/jovyan/work/rnn_modeldata/WXF.txt')
bF = opentensor('/home/jovyan/work/rnn_modeldata/bF0.txt')
WHF = opentensor('/home/jovyan/work/rnn_modeldata/WHF0.txt')
WCF = opentensor('/home/jovyan/work/rnn_modeldata/WCF0.txt')
WXI = opentensor('/home/jovyan/work/rnn_modeldata/WXI0.txt')
WHI = opentensor('/home/jovyan/work/rnn_modeldata/WHI.txt')
WC1 = opentensor('/home/jovyan/work/rnn_modeldata/WC10.txt')
WXC = opentensor('/home/jovyan/work/rnn_modeldata/WXC.txt')
WXO = opentensor('/home/jovyan/work/rnn_modeldata/WXO.txt')
bc = opentensor('/home/jovyan/work/rnn_modeldata/bc0.txt')
bi = opentensor('/home/jovyan/work/rnn_modeldata/bi0.txt')
W2 = opentensor('/home/jovyan/work/rnn_modeldata/W2.txt')

next open the vocabulary file and create a list of the words.
```

```
[34]: wordlines = [line.rstrip('\n') for line in open('/home/jovyan/work/rnn_modeldata/vocab.txt', "r")]

[35]: wordlist = []
for l in wordlines:
    wordlist.extend([pullword(l)])

[36]: worddict = {wordlist[i]: i for i in range(len(wordlist))}

The vocabulary is size 10000 and E is a 150x10000 matrix that has learned the compact representation of each word. getvec takes an english word looks in the wordlist to see if is there. If so, it returns the corresponding column vector of lenght 150.
```

```
[38]: def getvec(word, E):
    try:
        ind = worddict[word]
    except:
        print ("word " + word + " not in dictionary")
        return
    V = E[:,ind]
    V.shape = (150,1)
    return V
```

Mode: Command Ln 1, Col 1 rnn-lstm.ipynb

Filter files by name

Name	Last Modified
bc0.txt	a day ago
b0.txt	a day ago
bi0.txt	a day ago
bo0.txt	a day ago
cls2idx.txt	a day ago
E0.txt	a day ago
vocab.txt	a day ago
W2-new.txt	a day ago
W2.txt	a day ago
WCFO.txt	a day ago
WCIO.txt	a day ago
WC00.txt	a day ago
weightforclass.txt	a day ago
WHCO.txt	a day ago
WHFO.txt	a day ago
WHI.txt	a day ago
WHO.txt	a day ago
word2cls.txt	a day ago
WXC.txt	a day ago
WXF.txt	a day ago
WXIO.txt	a day ago
WXO.txt	a day ago

```
[38]: def getvec(word, E):
    try:
        ind = worddict[word]
    except:
        print ("word " + word + " not in dictionary")
        return
    V = E[:,ind]
    V.shape = (150,1)
    return V

[39]: def Sigmoid(x):
    return 1 / (1 + np.exp(-x))

The output vector of the rnn is a vector of length 10000. output[i] represents the relative likelihood that the next word is the best to follow the string so far. Getwordsfromoutput returns the top 5 candidate words.

[40]: def getwordsfromoutput(output):
    lst = []
    for i in range(10000):
        lst.append((output[0,i], i))
    dotsl = sorted(lst, key=lambda tup: -tup[0])
    #print dotsl[0:5]
    st = []
    for i in range(5):
        st.append(wordlist[dotsl[i][1]])
    return st
```

rnns is a direct translation of the lstm equations. the only difference is that we use an english word as input an return a list five possible nextwords as output.

```
[81]: def rnn(word, old_h, old_c):
    #features = SparseInputValue -> [10000 x *]
    Xvec = getvec(word, E)

    i = Sigmoid(np.matmul(WXI, Xvec) + np.matmul(WHI, old_h) + WC1 * old_c + bi)
    f = Sigmoid(np.matmul(WXF, Xvec) + np.matmul(WHF, old_h) + WCF * old_c + bF)

    c = f*old_c + i * (np.tanh(np.matmul(WXC, Xvec) + np.matmul(WXO, old_h) + bc))
    #o = f*old_c + i * (np.tanh(np.matmul(WXO, Xvec) + np.matmul(WHO, old_h) + b0))
```

Mode: Command Ln 1, Col 1 rnn-lstm.ipynb

The screenshot shows a Jupyter Notebook interface. On the left, there's a file browser titled "Filter files by name" showing a list of files in "/rnn\_modeldata/". On the right, a code cell in "Python 3 (ipykernel)" is executing Python code related to an RNN. The code includes imports for np, sigmoid, and random, and defines functions for calculating hidden states and extracting ordered lists of words. A specific cell (75) contains a sentence generation loop that starts with a word and iterates 100 times, printing the resulting sentence.

```

#0 = T * O_U_C + L * (np.tanh(np.matmul(WAU, Xvec) + np.matmul(WHO, old_h)) + bU)

o = Sigmoid(np.matmul(WX0,Xvec)+ np.matmul(WHO, old_h)+( WCO * old_c)+ b0)

h = o * np.tanh(c)

#extract ordered list of five best possible next words
q = h.copy()
q.shape = (1, 200)
output = np.matmul(q, W2)
outlist = getwordsfromoutput(output)
return h, c, outlist

[75]: import random
from random import randint

This takes any word as a starting point and constructs a sentence that is defined by the sequence generated by the RNN.

For the next word we randomly pick one of the top three suggested by the RNN.

[82]: c = np.zeros(shape = (200, 1))
h = np.zeros(shape = (200, 1))
output = np.zeros(shape = (10000, 1))
word = 'big'
sentence = word
for _ in range(100):
    h, c, outlist = rnn(word, h, c)
    word = outlist[randint(0,4)]
    sentence = sentence + " " +word
print (sentence+ ".")

big companies now now how no further hit until mr. mitchell says he hopes him whether might close at night b
ecause although alone americans else even like according david j. j. chief trader among equity brokerage con
cern </s> drewel director john john chief officer richard called an <unk> executive director told yesterday
morning plans again last month according val estimates however despite as issues yields offset losses margin
s lower levels without enough swings as they could look whether can such things too </s> there simply else r
isk enough too enough right unless orders were down despite stock-index trading prices volume mostly.

```

This takes any word as a starting point and constructs a sentence that is defined by the sequence generated by the RNN.

For the next word we randomly pick one of the top three suggested by the RNN.

This screenshot shows the same Jupyter Notebook environment within the Amazon SageMaker Studio interface. The file browser on the left shows the same directory structure. The notebook cell 75 is visible, and the code cell 29 is currently active, displaying the sentence generation logic. The notebook cell 29 has been modified to include a large block of generated text, likely from a previous run or a different cell.

```

[27]: import random
from random import randint

This takes any word as a starting point and constructs a sentence that is defined by the sequence generated by the RNN.

For the next word we randomly pick one of the top three suggested by the RNN.

[29]: c = np.zeros(shape = (200, 1))
h = np.zeros(shape = (200, 1))
output = np.zeros(shape = (10000, 1))
word = 'big'
sentence = word
for _ in range(100):
    h, c, outlist = rnn(word, h, c)
    word = outlist[randint(0,4)]
    sentence = sentence + " " +word
print (sentence+ ".")

big board composite trading volume friday following thursday afternoon according last spring moody savi
ngs trust plc a securities firm which operates several big parts since february when qintex australia a
nnounced an offer according to an analyst john r. john a. james a. james r. university john d. john r.
james d. calif ltd after taking advantage over an important financial position until june afternoon des
pite its stock purchases unless they expect him again within two months ago despite mr. jones seeking f
urther information again says john r. chief economist richard j. j. james & johnson inc according where
rep. john j..
[ ]: ***
[ ]: ***

```

## PART 4

Experiment with Deep Learning using Amazon's preferred Deep Learning toolkit MXNet; execute and document the following exercise

The only change has been the addition of a function to pull an image from a local directory and also configured so that a gpu is not needed.

From the MXNet page: "This is a demo for predicting with a pre-trained model on the full imagenet dataset, which contains over 10 million images and 10 thousands classes. For a more detailed explanation, please refer to predict.ipynb."

This was run using the AWS Machine Learning AMI

The screenshot shows the Amazon SageMaker Studio interface with the 'mxnet.ipynb' notebook open. The left sidebar displays a file tree with several text files and the 'mxnet.ipynb' notebook. The main content area shows the notebook code and output. The code cell [39] contains the following pip installation command:

```
[39]: pip install mxnet  
       pip install opencv-python-headless
```

The output of the command is displayed below the code cell, showing various package dependencies and their versions being installed. The output includes messages about deprecation warnings and root permissions.

This function does the prediction by converting the image to an RGB format of the right size.

Amazon SageMaker

Amazon SageMaker > Notebook instances > mxnet-cloud

**mxnet-cloud**

**Notebook instance settings**

Name: mxnet-cloud Status: InService Notebook instance type: ml.t2.medium Platform identifier: notebook-al1-v1

ARN: arn:aws:sagemaker:us-east-1:070578317188:notebook-instance/mxnet-cloud Creation time: Apr 14, 2022 18:56 UTC Last updated: Apr 14, 2022 19:01 UTC Lifecycle configuration: - Elastic Inference: - Volume Size: 5GB EBS

**Git repositories**

Name Repository URL Type

There are currently no resources.

**Permissions and encryption**

Feedback © 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

jupyter mxnet Last Checkpoint: 5 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_mxnet\_p27 O Logout

**Predict with pre-trained models**

This is a very slightly modified version of the 'Predict with pre-trained models' example from [http://mxnet.io/tutorials/python/predict\\_imagenet.html](http://mxnet.io/tutorials/python/predict_imagenet.html). The only change has been the addition of a function to pull an image from a local directory and also configured so that a gpu is not needed.

From the MXNet page: "This is a demo for predicting with a pre-trained model on the full imagenet dataset, which contains over 10 million images and 10 thousands classes. For a more detailed explanation, please refer to predict.ipynb."

This was run using the AWS Machine Learning AMI

First we download the model.

if you are using python3 you will need to install urllib.request

and you will need to install mxnet

In [7]: #!pip install mxnet

```
import os, urllib
import urllib
# for python 3 use urllib.request
import mxnet as mx
def download(url,prefix=''):
    filename = prefix+url.split('/')[-1]
    if not os.path.exists(filename):
        # for python3 use urllib.request.urlretrieve(url,filename)
        urllib.urlretrieve(url, filename)

path='http://data.mxnet.io/models/imagenet-11k/'
download(path+'resnet-152/resnet-152-symbol.json', 'full-')
download(path+'resnet-152/resnet-152-0000.params', 'full-')
download(path+'synset.txt', 'full-')
```

In [2]: with open('full-synset.txt', 'r') as f:
 synsets = [l.rstrip() for l in f]

jupyter mxnet Last Checkpoint: 5 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_mxnet\_p27 O nbdiff

```
In [2]: with open('full-synset.txt', 'r') as f:
    synsets = [l.rstrip() for l in f]

    sym, arg_params, aux_params = mx.model.load_checkpoint('full-resnet-152', 0)

In [3]: mod = mx.mod.Module(symbol=sym)
mod.bind(for_training=False, data_shapes=[('data', (1,3,224,224))])
mod.set_params(arg_params, aux_params)

/home/ec2-user/anaconda3/envs/amazonel_mxnet_p27/lib/python2.7/site-packages/mxnet/module/base_module.py:67: UserWarning
  ing: Data provided by label_shapes don't match names specified by label_names ({} vs. ['softmax_label'])
  warnings.warn(msg)

In [4]: %matplotlib inline
import matplotlib
matplotlib.rcParams['savefig', dpi=100]
import matplotlib.pyplot as plt
import cv2
import numpy as np
from collections import namedtuple
Batch = namedtuple('Batch', ['data'])

In [5]: def get_image(url, show=True):
    filename = url.split('/')[-1]
    print(filename)
    urllib.urlretrieve(url, filename)
    # for python 3 urllib.request.urlretrieve(url, filename)
    img = cv2.imread(filename)
    if img is None:
        print('failed to download ' + url)
        return ''
    if show:
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.axis('off')
    return filename
```

jupyter mxnet Last Checkpoint: 5 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_mxnet\_p27 O nbdiff

```
In [6]: def get_local(filename, show=True):
    img = cv2.imread(filename)
    if img is None:
        print('failed to load ' + filename)
        return ''
    if show:
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.axis('off')
    return filename

This function does the prediction by converting the image to an RGB format of the right size.

In [7]: def predict(filename, mod, synsets):
    img = cv2.cvtColor(cv2.imread(filename), cv2.COLOR_BGR2RGB)
    if img is None:
        return None
    img = cv2.resize(img, (224, 224))
    img = np.swapaxes(img, 0, 2)
    img = np.swapaxes(img, 1, 2)
    img = img[np.newaxis, :]

    mod.forward(Batch([mx.nd.array(img)]))
    prob = mod.get_outputs()[0].asnumpy()
    prob = np.squeeze(prob)

    a = np.argsort(prob)[::-1]
    for i in a[0:5]:
        print('p=%2.2f, %s' % (prob[i], synsets[i][synsets[i].find(' '):]))
```

now run the predictor. use a local jpg file or modify to use get\_image with a url or a jpg file  
add a url and run

```
In [8]: url = 'http://www.vetbook.org/wiki/dog/images/thumb/5/5c/Lowchen.jpg/694px-Lowchen.jpg'
url = 'https://cdn.pixabay.com/photo/2016/10/15/12/01/dog-1742295_960_720.jpg'
url = 'https://www.danmurnhvs.com.au/media/DM/Product/750x2000/379034_0_9999_v1_m56577569837963866.jpg'
```

now run the predictor. use a local jpg file or modify to use get\_image with a url or a jpg file  
add a url and run

jupyter mxnet Last Checkpoint: 6 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_mxnet\_p27 O

In [8]:

```
url = 'http://www.vetbook.org/wiki/dog/images/thumb/5/5c/Lowchen.jpg/694px-Lowchen.jpg'
url = 'https://cdn.pixabay.com/photo/2016/10/15/12/01/dog-1742295_960_720.jpg'
url = 'https://www.dammurphys.com.au/media/DM/Product/750x2000/379034_0_9999_v1_m56577569837963866.jpg'
url = 'https://img-new.cgtrader.com/items/20608/heineken_beer_bottle_3d_model_fbx_lwo_lw_lws_obj_max_lxo_79aeal8d-613b'
url = 'https://upload.wikimedia.org/wikipedia/commons/thumb/1/16/Official_Portrait_of_President_Reagan_1981.jpg/1200px-'
url = 'http://3.bp.blogspot.com/-wR12GBT0-Yk/Tiocy2ViGrI/AAAAAAAAB-8/Z9X6Jjc1P34/s1600/great-blue-heron.jpg'
url = 'http://assets.nydailynews.com/polopoly_fs/1.2773122.14726706851/img/httpImage/image.jpg_gen/derivatives/article_'
url = 'http://il.ytimg.com/vi/UxUDJlsnjI/maxresdefault.jpg'
file = get_image(url)
predict(file, mod, synsets)
```

maxresdefault.jpg

p=0.19, garbage truck, dustcart  
p=0.11, blacktop, blacktopping  
p=0.09, garbage man, garbageman, garbage collector, garbage carter, garbage hauler, refuse collector, dustman  
p=0.05, harvester, reaper  
p=0.04, broadcaster, spreader



## PART 5

The Tutorials/ and Examples/ folders contain a variety of example configurations for CNTK networks using the Python API, C# and Brain Script. The examples are structured by topic into Image, Language Understanding, Speech, and so forth. To get started with CNTK we recommend the tutorials

**CNTK 106: Part B - Time series prediction with LSTM (IOT Data)**

In part A of this tutorial we developed a simple LSTM network to predict future values in a time series. In part B we want to use the model on some real world internet-of-things (IOT) data. As an example we want to predict the daily output of a solar panel base on the initial readings of the day.

Solar power forecasting is a challenging and important problem. The solar energy generation forecasting problem is closely linked to the problem of weather variables forecasting. Indeed, this problem is usually split into two parts, on one hand focusing on the forecasting of solar PV or any other meteorological variable and on the other hand estimating the amount of energy that a concrete power plant will produce with the estimated meteorological resource. In general, the way to deal with this difficult problem is usually related to the spatial and temporal scales we are interested in. This tutorial focusses on a simplified forecasting model using previously generated data from solar panel to predict the future.

**Goal**

Using historic daily production of a solar panel, we want to predict the total power production of the solar panel array for a day. We will be using the LSTM based time series prediction model developed in part A to predict the daily output of a solar panel based on the initial readings of the day.

We train the model with historical data of the solar panel. In our example we want to predict the total power production of the solar panel array for the day starting with the initial readings of the day. We start predicting after the first 2 readings and adjust the prediction with each new reading.

In this tutorial, we will use the LSTM model introduced in the CNTK 106A. This tutorial has the following sub-sections:

- Setup
- Data generation
- LSTM network modeling
- Model training and evaluation

For more details on how LSTMs work, see [this excellent post](#).

We need a few imports and constants throughout the tutorial that we define here.

```
In [1]: from matplotlib import pyplot as plt
import math
import numpy as np
import os
```

We need a few imports and constants throughout the tutorial that we define here.

```
In [1]: from matplotlib import pyplot as plt
import math
import numpy as np
import os
import pandas as pd
import random
import time

import cntk as C

try:
    from urllib.request import urlretrieve
except ImportError:
    from urllib import urlretrieve

%matplotlib inline

import cntk.tests.test_utils
cntk.tests.test_utils.set_device_from_pytest_env() # (only needed for our build system)
```

```
In [2]: # to make things reproducible, seed random
np.random.seed(0)
```

There are two run modes:

- *Fast mode*: `isFast` is set to `True`. This is the default mode for the notebooks, which means we train for fewer iterations or train / test on limited data. This ensures functional correctness of the notebook though the models produced are far from what a completed training would produce.
- *Slow mode*: We recommend the user to set this flag to `False` once the user has gained familiarity with the notebook content and wants to gain insight from running the notebooks for a longer period with different parameters for training.

For *Fast mode* we train the model for 100 epochs and results have low accuracy but is good enough for development. The model yields good accuracy after 1000-2000 epochs.

```
In [3]: isFast = True

# we need around 2000 epochs to see good accuracy. For testing 100 epochs will do.
epochs = 100 if isFast else 2000
```

from running the notebooks for a longer period with different parameters for training.

For *Fast mode* we train the model for 100 epochs and results have low accuracy but is good enough for development. The model yields good accuracy after 1000-2000 epochs.

```
In [3]: isFast = True  
# we need around 2000 epochs to see good accuracy. For testing 100 epochs will do.  
EPOCHS = 100 if isFast else 2000
```

## Data generation

Our solar panel, emits two measures at every 30 min interval:

- `solar.current` is the current production in Watt
- `solar.total` is the total produced for the day so far in Watt/hour

Our prediction approach involves starting with the first 2 initial readings of the day. Based on these readings we start predicting and adjust the prediction with each new reading. The training data we are going to use comes as a csv file and has the following format:

```
time,solar.current,solar.total  
7am,6.3,1.7  
7:30am,44.3,11.4  
...  
...
```

The training dataset we use contains data captured for 3 years and can be found [here](#). The dataset is not pre-processed: it is raw data and contains smaller gaps and errors (like a panel failed to report).

## Pre-processing

Most of the code in this example is related to data preparation. Thankfully the pandas library make this easy.

`generate_solar_data()` function performs the following tasks:

- read raw data into a pandas dataframe

The training dataset we use contains data captured for 3 years and can be found [here](#). The dataset is not pre-processed: it is raw data and contains smaller gaps and errors (like a panel failed to report).

## Pre-processing

Most of the code in this example is related to data preparation. Thankfully the pandas library make this easy.

`generate_solar_data()` function performs the following tasks:

- read raw data into a pandas dataframe,
- normalize the data,
- groups by day and
- append the columns "solar.current.max" and "solar.total.max", and
- generates the sequences for each day.

*Sequence generation:* All sequences are concatenated into a single list of sequences. There is no more notion of timestamp in our train input and only the sequences matter.

**Note** if we have less than 8 datapoints for a day we skip over the day assuming something is missing in the raw data. If we get more than 14 data points in a day we truncate the readings.

## Training / Testing / Validation data preparation

We start by reading the csv file for use with CNTK. The raw data is sorted by time and we should randomize it before splitting into training, validation and test datasets but this would make it impractical to visualize results in the tutorial. Hence, we split the dataset in the following manner: pick in sequence, 8 values for training, 1 for validation and 1 for test until there is no more data. This will spread training, validation and test datasets across the full timeline while preserving time order.

```
In [4]: def generate_solar_data(input_url, time_steps, normalize=1, val_size=0.1, test_size=0.1):  
    """  
        generate sequences to feed to rnn based on data frame with solar panel data  
        the csv has the format: time ,solar.current, solar.total  
        (solar.current is the current output in Watt, solar.total is the total production  
        for the day so far in Watt hours)  
    """  
    # try to find the data file local. If it doesn't exists download it.  
    cache_path = os.path.join("data", "iot")  
    data_dir = os.path.dirname(cache_path)
```

```

In [4]: def generate_solar_data(input_url, time_steps, normalize=1, val_size=0.1, test_size=0.1):
    """
    generate sequences to feed to rnn based on data frame with solar panel data
    the csv has the format: time,solar.current, solar.total
    (solar.current is the current output in Watt, solar.total is the total production
     for the day so far in Watt hours)
    """
    # try to find the data file local. If it doesn't exists download it.
    cache_path = os.path.join("data", "iot")
    cache_file = os.path.join(cache_path, "solar.csv")
    if not os.path.exists(cache_path):
        os.makedirs(cache_path)
    if not os.path.exists(cache_file):
        urlretrieve(input_url, cache_file)
        print("downloaded data successfully from ", input_url)
    else:
        print("using cache for ", input_url)

    df = pd.read_csv(cache_file, index_col="time", parse_dates=['time'], dtype=np.float32)
    df["date"] = df.index.date

    # normalize data
    df["solar.current"] /= normalize
    df["solar.total"] /= normalize

    # group by day, find the max for a day and add a new column .max
    grouped = df.groupby(df.index.date).max()
    grouped.columns = ["solar.current.max", "solar.total.max", "date"]

    # merge continuous readings and daily max values into a single frame
    df_merged = pd.merge(df, grouped, right_index=True, on="date")
    df_merged = df_merged[["solar.current", "solar.total",
                           "solar.current.max", "solar.total.max"]]
    # we group by day so we can process a day at a time.
    grouped = df_merged.groupby(df_merged.index.date)
    per_day = []
    for _, group in grouped:
        per_day.append(group)

    # split the dataset into train, validation and test sets on day boundaries
    val_size = int(len(per_day) * val_size)
    test_size = int(len(per_day) * test_size)
    next_val = 0
    next_test = 0

    result_x = {"train": [], "val": [], "test": []}
    result_y = {"train": [], "val": [], "test": []}

    # generate sequences a day at a time
    for i, day in enumerate(per_day):
        # if we have less than 8 datapoints for a day we skip over the
        # day assuming something is missing in the raw data
        total = day["solar.total"].values
        if len(total) < 8:
            continue
        if i >= next_val:
            current_set = "val"
            next_val = i + int(len(per_day) / val_size)
        elif i >= next_test:
            current_set = "test"
            next_test = i + int(len(per_day) / test_size)
        else:
            current_set = "train"
            max_total_for_day = np.array(day["solar.total.max"].values[0])
        for j in range(2, len(total)):
            result_x[current_set].append(total[0:j])
            result_y[current_set].append([max_total_for_day])
            if j >= time_steps:
                break
    # make result_y a numpy array
    for ds in ["train", "val", "test"]:
        result_y[ds] = np.array(result_y[ds])
    return result_x, result_y

```

```

jupyter CNTK_106B_LSTM_Timeseries_with_IOT_Data (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 O
def merged = df_merged[["solar.current", "solar.total",
                       "solar.current.max", "solar.total.max"]]
# we group by day so we can process a day at a time.
grouped = df_merged.groupby(df_merged.index.date)
per_day = []
for _, group in grouped:
    per_day.append(group)

# split the dataset into train, validation and test sets on day boundaries
val_size = int(len(per_day) * val_size)
test_size = int(len(per_day) * test_size)
next_val = 0
next_test = 0

result_x = {"train": [], "val": [], "test": []}
result_y = {"train": [], "val": [], "test": []}

# generate sequences a day at a time
for i, day in enumerate(per_day):
    # if we have less than 8 datapoints for a day we skip over the
    # day assuming something is missing in the raw data
    total = day["solar.total"].values
    if len(total) < 8:
        continue
    if i >= next_val:
        current_set = "val"
        next_val = i + int(len(per_day) / val_size)
    elif i >= next_test:
        current_set = "test"
        next_test = i + int(len(per_day) / test_size)
    else:
        current_set = "train"
        max_total_for_day = np.array(day["solar.total.max"].values[0])
    for j in range(2, len(total)):
        result_x[current_set].append(total[0:j])
        result_y[current_set].append([max_total_for_day])
        if j >= time_steps:
            break
# make result_y a numpy array
for ds in ["train", "val", "test"]:
    result_y[ds] = np.array(result_y[ds])
return result_x, result_y

```

jupyter CNTK\_106B\_LSTM\_Timeseries\_with\_IOT\_Data (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

## Data caching

For routine testing we would like to cache the data locally when available. If it is not available from the cache locations we shall download.

```
In [5]: # We keep upto 14 inputs from a day
TIMESTEPS = 14

# 20000 is the maximum total output in our dataset. We normalize all values with
# this so our inputs are between 0.0 and 1.0 range.
NORMALIZE = 20000

X, Y = generate_solar_data("https://www.cntk.ai/jup/dat/solar.csv",
                           TIMESTEPS, normalize=NORMALIZE)

using cache for https://www.cntk.ai/jup/dat/solar.csv
```

### Utility for data fetching

`next_batch()` yields the next batch for training. We use variable size sequences supported by CNTK and batches are a list of numpy arrays where the numpy arrays have variable length.

A standard practice is to shuffle batches with each epoch. We don't do this here because we want to be able to graph the data that is easily interpretable.

```
In [6]: # process batches of 10 days
BATCH_SIZE = TIMESTEPS * 10

def next_batch(x, y, ds):
    """get the next batch for training"""

    def as_batch(data, start, count):
        return data[start:start + count]

    for i in range(0, len(x[ds]), BATCH_SIZE):
        yield as_batch(X[ds], i, BATCH_SIZE), as_batch(Y[ds], i, BATCH_SIZE)
```

### Understand the data format

jupyter CNTK\_106B\_LSTM\_Timeseries\_with\_IOT\_Data (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

### Understand the data format

You can now see the sequence we are going to feed to the LSTM. Note if we have less than 8 datapoints for a day we skip over the day assuming something is missing in the raw data. If we get more than 14 data points in a day we truncate the readings.

```
In [7]: X['train'][0:3]
Out[7]: [array([ 0.        ,  0.0006985], dtype=float32),
         array([ 0.        ,  0.0006985,  0.0033175], dtype=float32),
         array([ 0.        ,  0.0006985,  0.0033175,  0.010375 ], dtype=float32)]
```

```
In [8]: Y['train'][0:3]
Out[8]: array([[ 0.23899999],
               [ 0.23899999],
               [ 0.23899999]], dtype=float32)
```

## Model Creation (LSTM network)

Corresponding to the maximum possible 14 data points in the input sequence, we model our network with 14 LSTM cells, 1 cell for each data point we take during the day. Since the input sequences can be between 8 and 14 data points per sequence, we take the advantage of CNTK support for variable sequences as input to a LSTM so we can feed our sequences as-is with no additional need for padding.

The output of the neural network is the total output for the day and each sequence for a given day has the same total output.

For example:

```
1.7,11.4 -> 10300
1.7,11.4,67.5 -> 10300
1.7,11.4,67.5,250.5 ... -> 10300
1.7,11.4,67.5,250.5,573.5 -> 10300
```

The outputs from the LSTMs are fed into a dense layer and we randomly dropout 20% of the values to not overfit the model to the training set. The output of the dense layer becomes the prediction our model generates.

Notice: We lost the timestamp altogether; in our model only the sequences of readings matter.

Our LSTM model has the following design:

```

In [9]: #Specify the internal-state dimensions of the LSTM cell
H_DIMS = 15
def create_model(x):
    """Create the model for time series prediction"""
    with C.layers.default_options(initial_state = 0.1):
        m = C.layers.Recurrence(C.layers.LSTM(H_DIMS))(x)
        m = C.sequence.last(m)
        m = C.layers.Dropout(0.2)(m)
        m = C.layers.Dense(1)(m)
    return m

```

**Training**

Before we can start training we need to bind our input variables for the model and define what optimizer we want to use. For this example we choose the adam optimizer. We choose squared\_error as our loss function.

```

In [10]: # input sequences
x = C.sequence.input_variable(1)

# create the model
z = create_model(x)

# expected output (label), also the dynamic axes of the model output
# is specified as the model of the label input
l = C.input_variable(1, dynamic_axes=z.dynamic_axes, name="y")

# the learning rate
learning_rate = 0.005
lr_schedule = C.learning_parameter_schedule(learning_rate)

# loss function
loss = C.squared_error(z, l)

# use squared error to determine error for now
error = C.squared_error(z, l)

# use adam optimizer
momentum_schedule = C.momentum_schedule(0.9, minibatch_size=BATCH_SIZE)
learner = C.fsadagrad(z.parameters,

```

In this tutorial we will walk through several complete CNTK examples, showing

- how to describe a network in our network-description language BrainScript;
- how to set the learning parameters; and
- how to read in, and write out, data

Logistic regression is a simple model for performing binary classification. Given some real-valued d-dimensional example  $x = (x_1 \dots x_d)'$ , we want to output one of two labels: 0 or 1, true or false, spam or not-spam, etc. So, we want to learn a function  $y = f(x; w, b)$  where  $y$  is in  $\{0, 1\}$  and  $w = (w_1 \dots w_d)$  and  $b$  are parameters that we will learn that describe the relationship between  $x$  and  $y$ .

Without CNTK, we need to choose an optimization procedure and solve the derivatives of a cost function with respect to the parameters we want to learn. Seeing logistic regression as a probabilistic model, we can maximize the likelihood of the data. This turns out to be the same as minimizing the cross-entropy cost function, which for logistic regression is also known as the 'logistic loss function'.

jupyter CNTK\_106B\_LSTM\_Timeseries\_with\_IOT\_Data (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

```
In [12]: plt.plot(loss_summary, label='training loss');
```

Let us validate the training validation and test dataset. We use mean squared error as measure which might be a little simplistic. A method that would define a ratio how many predictions have been inside a given tolerance would make a better measure.

```
In [13]: # validate
def get_mse(X, Y, labeltxt):
    result = 0.0
    for xi, yi in next_batch(X, Y, labeltxt):
        eval_error = trainer.test_minibatch({x : xi, l : yi})
        result += eval_error
    return result/len(X[labeltxt])
```

```
In [14]: # Print the train and validation errors
for labeltxt in ["train", "val"]:
    print("mse for {}: {:.6f}".format(labeltxt, get_mse(X, Y, labeltxt)))
```

```
mse for train: 0.000082
mse for val: 0.000080
```

```
In [15]: # Print the test error
labeltxt = "test"
print("mse for {}: {:.6f}".format(labeltxt, get_mse(X, Y, labeltxt)))
```

the label dimension is now 3 instead of 1;

- we take out the Sigmoid() node;
- we replace the Logistic() learning criterion by CrossEntropyWithSoftmax() which will add the softmax layer and use cross entropy as the objective function; and
- we replaced the SquareError() evaluation node with ErrorPrediction().

jupyter CNTK\_106B\_LSTM\_Timeseries\_with\_IOT\_Data (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

```
In [15]: # Print the test error
labeltxt = "test"
print("mse for {}: {:.6f}".format(labeltxt, get_mse(X, Y, labeltxt)))
```

```
mse for test: 0.000080
```

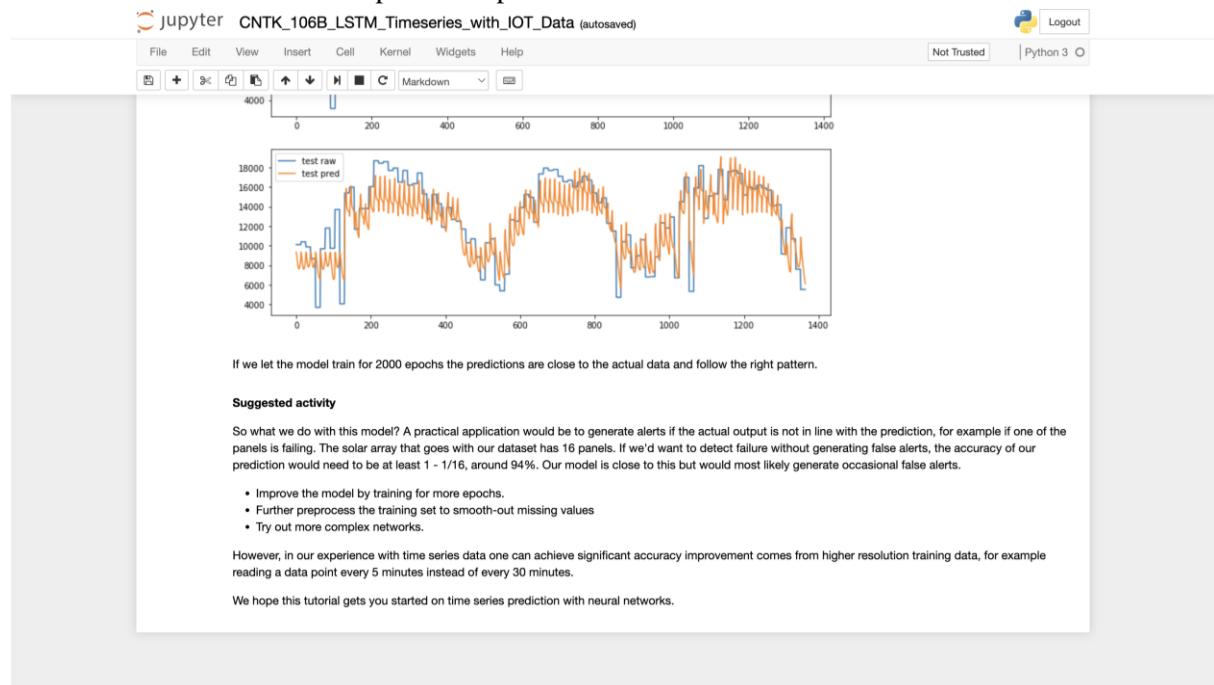
### Visualize results

Our model has been trained well, given the train, validation and test errors are in the same ball park. Predicted time series data renders well with visualization of the results. Let us take our newly created model, make predictions and plot them against the actual readings.

```
In [16]: # predict
f, a = plt.subplots(2, 1, figsize=(12, 8))
for j, ds in enumerate(['val', "test"]):
    results = []
    for x_batch, _ in next_batch(X, Y, ds):
        pred = z.eval({x : x_batch})
        results.extend(pred[:, 0])
    # because we normalized the input data we need to multiply the prediction
    # with SCALER to get the real values.
    a[j].plot((Y[ds] * NORMALIZE).flatten(), label=ds + ' raw');
    a[j].plot(np.array(results) * NORMALIZE, label=ds + ' pred');
    a[j].legend();
```

The softmax function is a generalization of the logistic function. It maps a vector of real values to a probability distribution. It is widely used for multi-class classification problems. In this context, if we

are trying to predict the class of an instance out of K possible classes, the model would compute a K-dimensional vector whose components represent the confidence scores for each class.



## PART 6

Get familiar with Deep Learning using TensorFlow; execute and document the following exercise

The data consists csv file of a GRE exam score in the range 0 to 800, a grade point average in the range 0 to 4.0 and the rank of the student's undergrad institution in the range 4 to 1 (top). The Admission Decision is binary.

A very simple Tensorflow Logistic Regression example.

We will build a logistic regression model of some (fake) graduate school admissions decisions. The data consists csv file of a GRE exam score in the range 0 to 800, a grade point average in the range 0 to 4.0 and the rank of the student's undergrad institution in the range 4 to 1 (top). The Admission Decision is binary.

There are six columns in the data. Ignore column 0, the GRE, GPA and school rank are in columns 1 through 3. Column 4 is an admission decision that is only based on the rank of the school: only students with a rank 1 school get in and others not. Column 5 contains a more interesting (but still silly) admission model where you get in if your GRE score is 800 or your rank is 1. In the example below we use the former model. It is easy for the system to learn it perfectly.

The csv file is called dat.csv and it is here [https://1drv.ms/u/s!AkRG9Zk\\_lOUag5IX1xzlv7rP9t5FFg](https://1drv.ms/u/s!AkRG9Zk_lOUag5IX1xzlv7rP9t5FFg)

```
In [1]: import tensorflow as tf
import numpy as np
import csv

/home/ec2-user/anaconda3/envs/amazonel_tensorflow2_p27/lib/python2.7/site-packages/OpenSSL/crypto.py:12: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
    from cryptography import x509

In [2]: sess=tf.compat.v1.InteractiveSession()

In [3]: #sess = tf.InteractiveSession()

In [4]: train_datas = []
train_labels = []
test_datas = []
test_labels = []
i = 0
scale = np.array([[0.01, 1.0, 1.0]])
with open('/home/ec2-user/SageMaker/dat.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        if i < 300:
            train_datas.append([np.float32(x) for x in row[1:4]])
            #print np.float(row[0])
            train_labels.append([np.float32(row[4])])
        else:
            test_datas.append([np.float32(x) for x in row[1:4]])
            test_labels.append([np.float(row[4])])
        i += 1

train_datas.append(np.float32(x) for x in row[1:4])
train_labels.append([np.float32(row[4])])
```

```
In [6]: train_datas = []
train_labels = []
test_datas = []
test_labels = []
i = 0
scale = np.array([[0.01, 1.0, 1.0]])
with open('/home/ec2-user/SageMaker/dat.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        if i < 300:
            train_datas.append([np.float32(x) for x in row[1:4]])
            #print np.float(row[0])
            train_labels.append([np.float32(row[4])])
        else:
            test_datas.append([np.float32(x) for x in row[1:4]])
            test_labels.append([np.float(row[4])])
        i += 1

In [7]: test_data = np.array(test_datas, dtype=np.float32)
test_label = np.array(test_labels, dtype=np.float32)
train_data = np.array(train_datas, dtype=np.float32)
train_label = np.array(train_labels, dtype=np.float32)

In [8]: test_data[0:20]
```

```
Out[8]: array([[640. ,  3.3 ,  2. ],
   [660. ,  3.6 ,  3. ],
   [400. ,  3.15,  2. ],
   [680. ,  3.98,  2. ],
   [220. ,  2.83,  3. ],
   [580. ,  3.46,  4. ],
   [540. ,  3.17,  1. ],
   [580. ,  3.51,  2. ],
   [540. ,  3.13,  2. ],
   [440. ,  2.98,  3. ],
   [560. ,  4. ,  3. ],
   [660. ,  3.67,  2. ],
   [660. ,  3.77,  3. ],
   [520. ,  3.65,  4. ],
   [540. ,  3.46,  4. ],
   [520. ,  3.65,  4. ],
   [540. ,  3.46,  4. ]])
```

jupyter tensorflow Last Checkpoint: 37 minutes ago (autosaved) 

File Edit View Insert Cell Kernel Widgets Help Trusted conda\_amazonel\_tensorflow2\_p27 O nbdiff

```
In [9]: train_label[0:20]
Out[9]: array([0.,  
           [0.],  
           [1.],  
           [0.],  
           [0.],  
           [0.],  
           [1.],  
           [0.],  
           [0.],  
           [0.],  
           [0.],  
           [1.],  
           [1.],  
           [0.],  
           [0.],  
           [0.],  
           [0.],  
           [0.],  
           [0.],  
           [1.]), dtype=float32)
```

first define the placeholders and variables

```
In [12]: import tensorflow.compat.v1 as tf
In [15]: tf.disable_v2_behavior()
WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/amazonel_tensorflow2_p27/lib/python2.7/site-packages/tensorflow_core/python/compat/v2_compat.py:65: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed in a future version.  
Instructions for updating:  
non-resource variables are not supported in the long term
```

jupyter tensorflow Last Checkpoint: 37 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_tensorflow2\_p27 O

In [15]: `tf.disable_v2_behavior()`

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/amazonel\_tensorflow2\_p27/lib/python2.7/site-packages/tensorflow\_core/python/compat/v2\_compat.py:65: disable\_resource\_variables (from tensorflow.python.ops.variable\_scope) is deprecated and will be removed in a future version.  
Instructions for updating:  
non-resource variables are not supported in the long term

In [17]: `x = tf.placeholder(tf.float32, shape=(None,3))  
y = tf.placeholder(tf.float32, shape =(None,1))  
  
# Set model weights  
W = tf.Variable(tf.zeros([3, 1]))  
b = tf.Variable(tf.zeros([1]))`

In [18]: `training_epochs = 400000  
batch_size = 100  
display_step = 1000`

In [19]: `pred = tf.sigmoid(tf.matmul(x, W) + b) # Softmax  
cost = tf.sqrt(tf.reduce_mean((y - pred)**2/batch_size))  
  
opt = tf.train.AdamOptimizer()  
optimizer = opt.minimize(cost)  
  
# Initializing the variables old version is initialize_all_variables. New version  
# has global_variable_initializer.  
  
#init = tf.global_variables_initializer()  
init = tf.initialize_all_variables()`

WARNING:tensorflow:From /home/ec2-user/anaconda3/envs/amazonel\_tensorflow2\_p27/lib/python2.7/site-packages/tensorflow\_core/python/util/tf\_should\_use.py:198: initialize\_all\_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.  
Instructions for updating:  
Use `tf.global\_variables\_initializer` instead.

In [20]: `def get_batch2(batch_size, x, y):  
 indices = np.random.choice(299, batch_size)  
 return x[indices], y[indices]`

jupyter tensorflow Last Checkpoint: 37 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_tensorflow2\_p27 O

In [20]: `def get_batch2(batch_size, x, y):  
 indices = np.random.choice(299, batch_size)  
 return x[indices], y[indices]`

In [21]: `sess.run(init)  
# Training cycle  
for epoch in range(training_epochs):  
 avg_cost = 0.  
 total_batch = int(len(train_data)/batch_size)  
 # Loop over all batches  
 for i in range(total_batch):  
 batch_xs, batch_ys = get_batch2(batch_size, train_data, train_label)  
 # Fit training using batch data  
 _, c = sess.run([optimizer, cost], feed_dict={x:batch_xs,y:batch_ys})  
 # Compute average loss  
 avg_cost += c / total_batch  
 # Display logs per epoch step  
 if (epoch+1) % display_step == 0:  
 print "Epoch:", '%04d' % (epoch+1), "cost=", str(avg_cost)  
  
print "Optimization Finished!"`

Epoch: 383000 cost= nan  
Epoch: 384000 cost= nan  
Epoch: 385000 cost= nan  
Epoch: 386000 cost= nan  
Epoch: 387000 cost= nan  
Epoch: 388000 cost= nan  
Epoch: 389000 cost= nan  
Epoch: 390000 cost= nan  
Epoch: 391000 cost= nan  
Epoch: 392000 cost= nan  
Epoch: 393000 cost= nan  
Epoch: 394000 cost= nan  
Epoch: 395000 cost= nan  
Epoch: 396000 cost= nan  
Epoch: 397000 cost= nan  
Epoch: 398000 cost= nan  
Epoch: 399000 cost= nan

jupyter tensorflow Last Checkpoint: 38 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_tensorflow2\_p27 O

Epoch: 398000 cost= nan  
Epoch: 399000 cost= nan  
Epoch: 400000 cost= nan  
Optimization Finished!

define a function in python to compute sigmoid of an array.

```
In [22]: def sigmoid(z):  
    res = np.zeros(len(z))  
    for i in range(len(z)):  
        y = 1.0/(1.0+np.exp(-z[i]))  
        if y > 0.5:  
            res[i] = 1  
        else:  
            res[i] = 0  
    return res
```

```
In [23]: p = sigmoid(pred.eval({x:test_data}))  
q = y.eval({y:test_label})
```

how accurate is this. In other words what fraction of the cases have  $p = q$ .

```
In [24]: def accurate(p,q):  
    total = 0.0  
    numones = 0.  
    numzeros = 0.  
    for i in range(len(p)):  
        if p[i] == 0:  
            numzeros += 1  
        else:  
            numones += 1  
        if p[i] == q[i]:  
            total +=1  
    print 'accuracy = '+str(total/len(p))  
    print 'ratio of 1s ='+str(numones/len(p))  
    print 'ratio of 0s ='+str(numzeros/len(p))
```

jupyter tensorflow Last Checkpoint: 38 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted | conda\_amazonel\_tensorflow2\_p27 O

```
In [25]: accurate(p,q)  
accuracy = 0.84  
ratio of 1s =0.0  
ratio of 0s =1.0
```

```
In [26]: def softmax(v):  
    k = len(v)  
    res = np.zeros(k)  
    tot = 0.0  
    for i in range(k):  
        res[i] = np.exp(v[i])  
        tot += res[i]  
    for i in range(k):  
        res[i] = res[i]/tot  
    return res
```

```
In [27]: def sigmoid(x):  
    y = 1.0/(1.0+np.exp(-x))  
    if y > 0.5:  
        return 1  
    else:  
        return 0
```

if we wish, we may look at the trained W and b arrays.

```
In [28]: w =W.eval()  
In [29]: w  
Out[29]: array([[[nan],  
                 [nan],  
                 [nan]], dtype=float32)
```

```
In [30]: lb =b.eval()  
In [31]: lb
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter tensorflow Last Checkpoint: 38 minutes ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Status Bar:** Trusted | conda\_amazonel\_tensorflow2\_p27 O
- Code Cells:**
  - In [30]: `lb = b.eval()`
  - In [31]: `lb`
  - Out[31]: `array([nan], dtype=float32)`
  - In [32]: `for i in range(20):  
 print sigmoid(np.matmul(train_data[i], w)+lb)`
  - Output of In [32]:  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0

There are six columns in the data. Ignore column 0. the GRE, GPA and school rank are in columns 1 through 3. Column 4 is an admission decision that is only based on the rank of the school: only students with a rank 1 school get in and others not. Column 5 contains a more interesting (but still silly) admission model where you get in if your GRE score is 800 or your rank is 1. in the example below we use the former model. It is easy for the system to learn it perfectly.