Open in Colab

In [0]:

```
!nvcc --version
!pip install cupy-cuda101
```

In [0]:

```
data_path   = "./"
```

In [0]:

```
import numpy as np
import re
import nltk
from sklearn.datasets import load_files
nltk.download('stopwords')
nltk.download('wordnet')
import pickle
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split


review_data = load_files(data_path + "movie_review")
X, y = review_data.data, review_data.target

documents = []

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):
    # Remove all the special characters
    document = re.sub(r'₩₩n', ' ', str(X[sen].decode('UTF-8')))
    document = re.sub(r"₩₩₩'", "'", document)
    document = expand_contractions(document)

    document = re.sub(r'[^a-zA-z0-9₩s]', ' ', document)

    # remove all single characters
    document = re.sub(r'₩s+[a-zA-Z]₩s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'₩^[a-zA-Z]₩s+', ' ', document)

    # Substituting multiple spaces with single space
    document = re.sub(r'₩s+', ' ', document, flags=re.I)


    # Converting to Lowercase
    document = document.lower()

    # Lemmatization
    document = document.split()
    document = [stemmer.lemmatize(word) for word in document]
    document = ' '.join(document)

    documents.append(document)

# vectorizer = CountVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words=stopwor
ds.words('english'))
# X = vectorizer.fit_transform(documents).toarray()

# tfidfconverter = TfidfTransformer()
# X = tfidfconverter.fit_transform(X).toarray()

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)
```

In [0]:

```
import pickle
with open(data_path + 'documents2.txt', 'rb') as f:
      documents = pickle.load(f)
with open(data_path + 'y2.txt', 'rb') as f:
      y = pickle.load(f)
```

In [0]:

```
import pickle
with open(data_path + 'documents2.txt', 'wb') as f:
      pickle.dump(documents, f)
with open(data_path + 'y2.txt', 'wb') as f:
      pickle.dump(y, f)
```

In [0]:

```
CONTRACTION_MAP = {
"ain't": "is not",
"aren't": "are not",
"can't": "can not",
"can't've": "can not have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he would",
"he'd've": "he would have",
"he'll": "he will",
"he'll've": "he he will have",
"he's": "he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how is",
"I'd": "I would",
"I'd've": "I would have",
"I'll": "I will",
"I'll've": "I will have",
"I'm": "I am",
"I've": "I have",
"i'd": "i would",
"i'd've": "i would have",
"i'll": "i will",
"i'll've": "i will have",
"i'm": "i am",
"i've": "i have",
"isn't": "is not",
"it'd": "it would",
"it'd've": "it would have",
"it'll": "it will",
"it'll've": "it will have",
"it's": "it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she would",
"she'd've": "she would have",
"she'll": "she will",
"she'll've": "she will have",
"she's": "she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so as",
"that'd": "that would",
"that'd've": "that would have",
"that's": "that is",
"there'd": "there would",
"there'd've": "there would have",
"there's": "there is",
"they'd": "they would",
"they'd've": "they would have",
"they'll": "they will",
"they'll've": "they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what'll've": "what will have",
"what're": "what are",
"what's": "what is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where did",
"where's": "where is",
"where've": "where have",
"who'll": "who will",
"who'll've": "who will have",
"who's": "who is",
"who've": "who have",
"why's": "why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you would",
"you'd've": "you would have",
"you'll": "you will",
"you'll've": "you will have",
"you're": "you are",
"you've": "you have"
}
```

In [0]:

```
import re
def expand_contractions(text, contraction_mapping=CONTRACTION_MAP):

    contractions_pattern = re.compile('({})'.format('|'.join(contraction_mapping.keys())),
                                      flags=re.IGNORECASE|re.DOTALL)
    def expand_match(contraction):
        match = contraction.group(0)
        first_char = match[0]
        expanded_contraction = contraction_mapping.get(match)₩
                                if contraction_mapping.get(match)₩
                                else contraction_mapping.get(match.lower())
        expanded_contraction = first_char+expanded_contraction[1:]
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text
```

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords
stopword_list = stopwords.words('english')
stopword_list.remove('no')
stopword_list.remove('not')
vectorizer = CountVectorizer(stop_words=stopword_list)
X = vectorizer.fit_transform(documents).toarray()
X.shape
```

In [0]:

```
from sklearn.decomposition import TruncatedSVD
tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()
svd = TruncatedSVD(n_components=1500)
X = svd.fit_transform(X)
X.shape
```

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)
```

In [0]:

```
X_t = X.T[pd.merge(pd.DataFrame(X_train), pd.DataFrame(y_train,columns=list('A')), left_in
dex=True, right_index=True).corr().abs()['A'][:-1] > 0.03 ].T
X_t.shape
```

In [0]:

```
import statsmodels.api as sm
import pandas as pd
x2 = sm.add_constant(X_train)
model = sm.OLS(y_train, x2)
result = model.fit()
result = result.summary2().tables[1]
X_t2 = (X_t.T)[result['P>|t|'][1:] <= 0.01].T
result.loc[result['P>|t|'] <= 0.01]
```

In [0]:

```
import pickle
print("save")
with open(data_path + 'X_train.pk', 'wb') as f:
  pickle.dump(X_train, f)
with open(data_path + 'X_test.pk', 'wb') as f:
  pickle.dump(X_test, f)
with open(data_path + 'y_train.pk', 'wb') as f:
  pickle.dump(y_train, f)
with open(data_path + 'y_test.pk', 'wb') as f:
  pickle.dump(y_test, f)

print("save done")
```

In [0]:

```
import pickle
with open(data_path + 'X_train.txt', 'rb') as f:
      X_train = pickle.load(f)
with open(data_path + 'X_test.txt', 'rb') as f:
      X_test = pickle.load(f)
with open(data_path + 'y_train.txt', 'rb') as f:
      y_train = pickle.load(f)
with open(data_path + 'y_test.txt', 'rb') as f:
      y_test = pickle.load(f)
```

In [0]:

```
def get_init_thetas(channel_nums):
  thetas = []
  for i in range(len(channel_nums)-1):
    devi = (2 / (channel_nums[i+1] + channel_nums[i] + 1)) ** 0.5
    temp_theta = np.random.normal(0, devi, (channel_nums[i] + 1) * channel_nums[i+1])
    temp_theta = temp_theta.reshape([-1, channel_nums[i+1], channel_nums[i] + 1])
    thetas.append(temp_theta)
  return thetas

def sigmoid(z):
  return 1 / (1 + np.exp(-z))

def propagation(thetas, xs):
  temp_xs = xs
  results = []
  for theta in thetas:
    temp_xs = np.concatenate((temp_xs, np.ones([len(temp_xs), 1])), axis=1)
    results.append(temp_xs)
    temp_xs = temp_xs.reshape([temp_xs.shape[0], 1, temp_xs.shape[1]])
    num_batch = 100
    tempA = []
    tempA.append(np.sum(temp_xs[:int(len(temp_xs)/num_batch)] * theta, axis=2))
    for i in range(1, num_batch-1):
      tempA.append(np.sum(temp_xs[int(len(temp_xs)* i/num_batch):int(len(temp_xs)* (i+1)/n
um_batch)] * theta, axis=2))
    tempA.append(np.sum(temp_xs[int(len(temp_xs)* (i+1)/num_batch):] * theta, axis=2))
    temp = np.concatenate(tempA, axis=0)
    temp_xs = sigmoid(temp)
    # temp_xs = sigmoid(np.sum(temp_xs * theta, axis=2))
  results.append(temp_xs)
  return (temp_xs, results)

def objectiveFunction(labels, results, thetas, controlParameter):
  output = results[-1].flatten()
  temp = -1 * labels * np.log(output) - (1 - labels) * np.log(1 - output)
  temp = np.mean(temp)
  temp2 = 0
  n = 0
  for theta in thetas:
    temp2 += np.sum(theta**2) - np.sum(theta[0][:,-1]**2)
    n += len(theta.flatten())
  temp += controlParameter * 0.5 * temp2 / n
  return temp
```

In [0]:

```
import math
def backpropa(labels, thetas, results, lr, controlParameter):
  drivated = []
  for theta in thetas:
    drivated.append(np.zeros(theta.shape))
  numOfData = len(labels)
  size = 100
  for itor in range(math.ceil(numOfData / size)):
    subLabels = labels[size * itor: size * (itor+1)]
    subResults = []
    for result in results:
      subResults.append(result[size * itor: size * (itor+1)])
    dp = []

    temp = subResults[-1].flatten() - subLabels
    temp = temp.reshape([temp.shape[0], 1, 1])
    dp.append(temp)
    temp = temp.reshape([temp.shape[0], 1])
    temp = temp * subResults[-2]
    temp = np.sum(temp, axis=0) / numOfData
    drivated[-1] = drivated[-1] + temp

    for theta_itor in reversed(range(len(thetas)-1)):
      temp = subResults[theta_itor + 1][:,:-1]
      temp = temp * (1 - temp)
      temp = temp.reshape([temp.shape[0], 1, temp.shape[1]]) * thetas[theta_itor+1][:,:,:-
1]
      dp.append(temp)
      for back_itor in reversed(range(len(dp) - 1)):
        temp_shape = list(temp.shape)
        temp_shape.insert(1,1)
        temp = temp.reshape(temp_shape)
        temp_shape = list(dp[back_itor].shape)
        temp_shape.append(1)
        temp = temp * dp[back_itor].reshape(temp_shape)
        temp = np.sum(temp, axis=2)
      temp = np.sum(temp, axis=1)
      temp_shape = list(temp.shape)
      temp_shape.append(1)
      temp = temp.reshape(temp_shape)
      temp_shape = list(subResults[theta_itor].shape)
      temp_shape.insert(1,1)
      temp = temp * subResults[theta_itor].reshape(temp_shape)
      temp = np.sum(temp, axis=0) / numOfData
      drivated[theta_itor] = drivated[theta_itor] + temp
  numOfThetas = 0
  new_thetas = []
  for theta in thetas:
    numOfThetas += len(theta.flatten())
  for itor in range(len(drivated)):
    regular = (controlParameter * thetas[itor] / numOfThetas)
    regular[0][:,-1] = 0
    temp = drivated[itor] + regular
    new_thetas.append(thetas[itor] - lr * temp)
  return new_thetas
```

In [0]:

```
import time
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
def process(thetas, train_image, train_label, test_image, test_label, channel_nums, lr, errors, accureacies, pbar, itor, controlParameter):
  if np.__name__ == 'cupy':
    train_image = np.array(train_image)
    train_label = np.array(train_label)
    test_image = np.array(test_image)
    test_label = np.array(test_label)
    for index in range(len(thetas)):
      thetas[index] = np.array(thetas[index])
  prev_error = -2
  train_error = -1
  count = 0
  while(train_error != prev_error and count < itor):
    start = time.time()
    prev_error = train_error
    (train_predict, train_results) = propagation(thetas, train_image)
    (test_predict, test_results) = propagation(thetas, test_image)
    train_predict = np.around(train_predict)
    test_predict = np.around(test_predict)
    train_accurcay = accuracy_score(train_label.tolist(), train_predict.tolist())
    test_accurcay = accuracy_score(test_label.tolist(), test_predict.tolist())
    train_error = objectiveFunction(train_label, train_results, thetas, controlParameter)
    test_error = objectiveFunction(test_label, test_results, thetas, controlParameter)
    errors[0].append(train_error.tolist())
    errors[1].append(test_error.tolist())
    accureacies[0].append(train_accurcay.tolist() * 100)
    accureacies[1].append(test_accurcay.tolist() * 100)
    thetas = backpropa(train_label, thetas, train_results, lr, controlParameter)
    count += 1
    pbar.update(1)
    # if count % 100 == 0:
    print(len(errors[0]), str(np.around(time.time() - start, 2)) + 's' ₩
          ,'/' , str(np.around(train_accurcay * 100, 2)) + "%", train_error ₩
          ,'/' , str(np.around(test_accurcay * 100, 2)) + "%", test_error)

  if np.__name__ == 'cupy':
    for i in range(len(thetas)):
      thetas[i] = np.asnumpy(thetas[i])
    test_predict = np.asnumpy(test_predict)
  return [test_predict, thetas]
```

In [0]:

```
import pickle
length = str(6210)
with open(data_path + 'accureacies/'+length+'.pk', 'rb') as f:
  accureacies = pickle.load(f)
with open(data_path + 'errors/'+length+'.pk', 'rb') as f:
  errors = pickle.load(f)
with open(data_path + 'thetas/'+length+'.pk', 'rb') as f:
  thetas = pickle.load(f)
with open(data_path + 'test_predict/'+length+'.pk', 'rb') as f:
  test_predict = pickle.load(f)

channel_nums = [len(X_train[0])]
for theta in thetas:
  print(theta.shape)
  channel_nums.append(theta.shape[1])
print(channel_nums)
```

In [0]:

```
import numpy as np
channel_nums = [len(X_train[0]), 512 , 512, 1]
thetas = get_init_thetas(channel_nums)
errors = [[], []]
accureacies = [[], []]
```

In [0]:

```
import pickle
from tqdm.notebook import tqdm
import cupy as np
# import numpy as np
itor = 200
itor_itor = 5
pbar = tqdm(total=itor * itor_itor)
for i in range(itor_itor):
  [test_predict, thetas] = process(thetas, X_train, y_train, X_test, y_test, channel_nums,
0.025, errors, accureacies, pbar, itor, 0.025)
  print("save")
  length = str(len(errors[0])+3)
  with open(data_path + './accureacies/'+length+'.pk', 'wb') as f:
    pickle.dump(accureacies, f)
  with open(data_path + './errors/'+length+'.pk', 'wb') as f:
    pickle.dump(errors, f)
  with open(data_path + './thetas/'+length+'.pk', 'wb') as f:
    pickle.dump(thetas, f)
  with open(data_path + './test_predict/'+length+'.pk', 'wb') as f:
    pickle.dump(test_predict, f)
  print("save done", length)

pbar.close()
print(len(errors[0]), len(errors[1]), len(accureacies[0]), len(accureacies[1]))
```

In [0]:

```
print("save")
length = str(len(errors[0])+10)
with open(data_path + 'accureacies/'+length+'.pk', 'wb') as f:
  pickle.dump(accureacies, f)
with open(data_path + 'errors/'+length+'.pk', 'wb') as f:
  pickle.dump(errors, f)
with open(data_path + 'thetas/'+length+'.pk', 'wb') as f:
  pickle.dump(thetas, f)
with open(data_path + 'test_predict/'+length+'.pk', 'wb') as f:
  pickle.dump(test_predict, f)
print("save done", length)
```

```
save
save done 7210
```

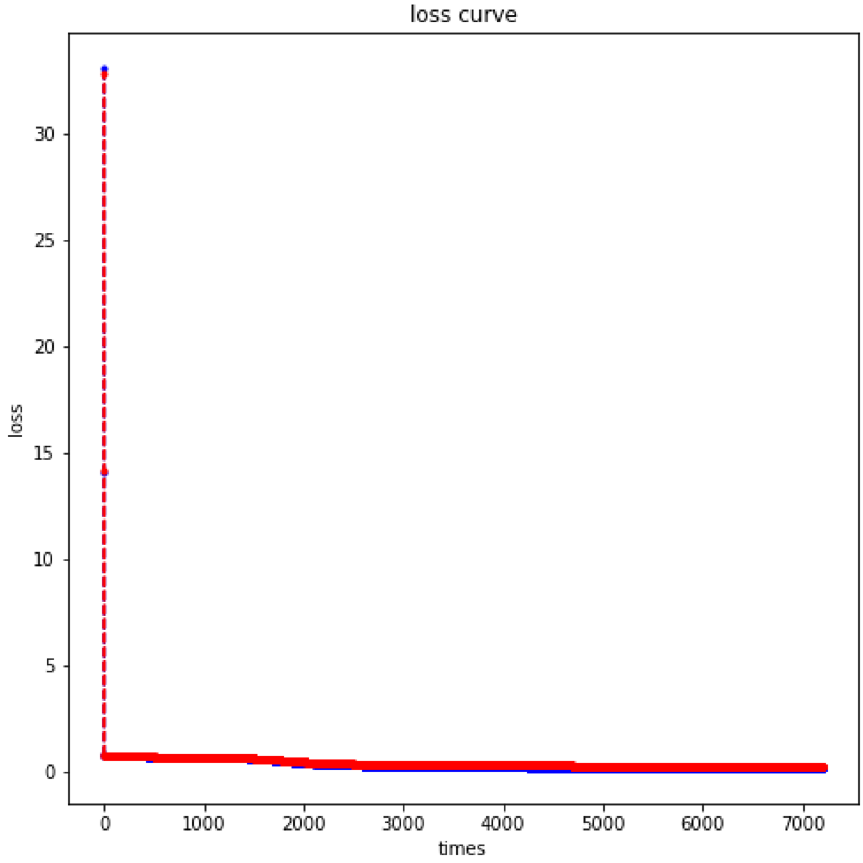
In [0]:

```
import numpy as np
(train_predict, train_results) = propagation(thetas, X_train)
(test_predict, test_results) = propagation(thetas, X_test)
train_predict = np.around(train_predict)
test_predict = np.around(test_predict)
```

Plot

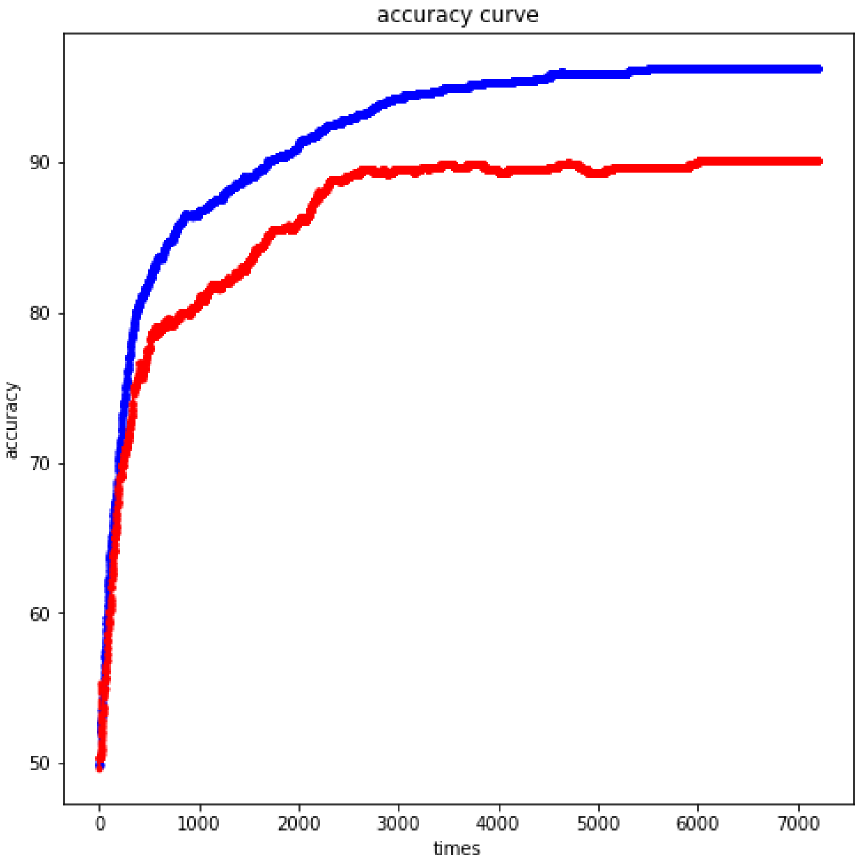
In [0]:

```
import matplotlib.pyplot as plt

time_points = range(len(errors[0]))
plt.figure(figsize=(8, 8))
plt.title("loss curve")
plt.xlabel("times")
plt.ylabel("loss")
plt.plot(time_points, errors[0], 'b.--')
plt.plot(time_points, errors[1], 'r.--')
plt.show()
```



In [0]:

```
import matplotlib.pyplot as plt

time_points = range(len(accureacies[0]))
plt.figure(figsize=(8, 8))
plt.title("accuracy curve")
plt.xlabel("times")
plt.ylabel("accuracy")
plt.plot(time_points, accureacies[0], 'b.--')
plt.plot(time_points, accureacies[1], 'r.--')
plt.show()
```



In [0]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Training results")
print(confusion_matrix(y_train,train_predict))
print(classification_report(y_train,train_predict))
print( accuracy_score(y_train, train_predict))
print("-------------------------------------------------------------")
print("Testing results")
print(confusion_matrix(y_test,test_predict))
print(classification_report(y_test,test_predict))
print(accuracy_score(y_test, test_predict))
```

```
Training results
[[669  29]
 [ 23 679]]
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       698
           1       0.96      0.97      0.96       702

    accuracy                           0.96      1400
   macro avg       0.96      0.96      0.96      1400
weighted avg       0.96      0.96      0.96      1400

0.9628571428571429
-------------------------------------------------------------
Testing results
[[272  30]
 [ 29 269]]
              precision    recall  f1-score   support

           0       0.90      0.90      0.90       302
           1       0.90      0.90      0.90       298

    accuracy                           0.90       600
   macro avg       0.90      0.90      0.90       600
weighted avg       0.90      0.90      0.90       600

0.9016666666666666
```