Word2Vec

November 20, 2022

1 TP 2: Redes Recurrentes y Representaciones Incrustadas

1.1 2. (30 puntos extra) Perceptrón multi-capa

1.1.1 Imports

[ ]:
```
import gensim
from gensim.models import word2vec
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
import torch.nn.functional as F
from time import time
import torch.optim as optim

import re

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


# classifier imports
from sklearn.neural_network import MLPClassifier
```

1.1.2 Load the data from the SMS+Spam+Collection

https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection

```
[ ]: #Read the dataset using Pandas and delimiter as tabulation.
messages = pd.read_csv('.\smsspamcollection\SMSSpamCollection',␣
 ↪encoding='latin-1',delimiter="\t",header=None)
#Set labels on the colums to ease manipulation.
messages.columns = ["label", "text"]
messages.head()
```

```
[ ]:   label                                               text
     0   ham  Go until jurong point, crazy.. Available only …
     1   ham                      Ok lar… Joking wif u oni…
     2  spam  Free entry in 2 a wkly comp to win FA Cup fina…
     3   ham  U dun say so early hor… U c already then say…
     4   ham  Nah I don't think he goes to usf, he lives aro…
```

1.1.3 Start Preparing the data.

```
[ ]: # Replace ham with 0 and spam with 1
messages = messages.replace(['ham','spam'],[0, 1])
```

1.1.4 Gensim implementation for feature extraction.

```
[ ]: #Preprocess with built-in Gensim libraries creating a new column with the new␣
 ↪pre-processed text.
#simple_preprocess lowercases, tokenizes and de-accents and returns the final␣
 ↪tokens as unicode strings.
#We are calling the pre-processed text, text_pp.
messages['text_pp'] = messages['text'].apply(lambda x: gensim.utils.
 ↪simple_preprocess(x,deacc=True,min_len=2,max_len=15))
messages.head()
```

```
[ ]:    label                                               text  \
     0      0  Go until jurong point, crazy.. Available only …
     1      0                      Ok lar… Joking wif u oni…
     2      1  Free entry in 2 a wkly comp to win FA Cup fina…
     3      0  U dun say so early hor… U c already then say…
     4      0  Nah I don't think he goes to usf, he lives aro…

                                              text_pp
     0  [go, until, jurong, point, crazy, available, o…
     1                       [ok, lar, joking, wif, oni]
     2  [free, entry, in, wkly, comp, to, win, fa, cup…
     3     [dun, say, so, early, hor, already, then, say]
     4  [nah, don, think, he, goes, to, usf, he, lives…
```

```
[ ]: # Count amount of ham and spam.
# Where 0 is not spam and 1 spam.
messages['label'].value_counts()
```

[ ]:
```
0    4825
1     747
Name: label, dtype: int64
```

1.2 Preprocessing Messages

1.2.1 PorterStemeer to remove stopwords from the dataset

[ ]:
```
# We define an empty list to build the corpus for the word2Vec model.
corpus = []
ps = PorterStemmer()
```

[ ]:
```
#PorterStemeer to remove stopwords from the dataset and create the corpus.
for i in range(0, 5572):


    msg = messages['text_pp'][i]

    # Stemming with PorterStemmer handling Stop Words
    msg = [ps.stem(word) for word in msg if not word in set(stopwords.
 ↪words('english'))]

    # preparing Messages with Remaining Tokens
    msg = ' '.join(msg)

    # Preparing WordVector Corpus
    corpus.append(msg)
```

1.3 Word2Vec model

1.3.1 Build the Word2Vec model.

[ ]:
```
#The number of features for the model or D, according to TP document.
D=100

w2v_model = word2vec.Word2Vec(min_count=2,
                     window=3,
                     vector_size=D,
                     sample=6e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=4)
```

1.3.2 Build word2vec vocabulary with the complete dataset.

[ ]:
```
#Build word2vec vocabulary with the complete dataset.
t = time()

w2v_model.build_vocab(messages['text_pp'], progress_per=10000)
#Prints the time that it tool to build the vocabulary.
print('Time to build vocab: {} mins'.format(round((time() - t) / 60, 2)))
```

```
Time to build vocab: 0.0 mins
```

1.3.3 Train the word2vec model.

[ ]:
```
#Train the word2vec model.
t = time()

w2v_model.train(messages['text_pp'], total_examples=w2v_model.corpus_count,␣
  ↪epochs=30, report_delay=1)
#Prints the time that it took to train the model.
print('Time to train the model: {} mins'.format(round((time() - t) / 60, 2)))
```

```
Time to train the model: 0.01 mins
```

1.3.4 Testing the word2vec model.

[ ]:
```
#We can check the vectors from a specific word:
w2v_model.wv['now']
```

[ ]:
```
array([-0.19160908,  0.23981117,  0.4922505 , -0.19098581,  0.1856047 ,
       -0.25444582, -0.10798351,  0.3521298 , -0.22197844, -0.22279803,
       -0.07858622, -0.58972776, -0.15920813,  0.27845109,  0.16600876,
       -0.08639668, -0.03629502, -0.41243225,  0.22971849, -0.35526857,
       -0.25310177,  0.1740092 , -0.14529708,  0.22386938, -0.15698647,
        0.5630012 , -0.10885131,  0.07877631, -0.3082642 ,  0.09887859,
        0.19486746, -0.10290623,  0.16490392, -0.17262237, -0.27110523,
        0.2857816 ,  0.0625147 , -0.3017694 , -0.18171684, -0.68487823,
       -0.00179522,  0.22696145, -0.50293785,  0.2459044 ,  0.35869107,
        0.05961956, -0.335956  , -0.00786543, -0.17862292,  0.07066388,
       -0.2635694 , -0.12014701, -0.44177553, -0.01300936, -0.55826044,
        0.34145817,  0.5019229 , -0.27082464, -0.04179491,  0.15661003,
        0.30356234,  0.3772025 , -0.21901399,  0.13329476,  0.0425786 ,
        0.09004112,  0.00775765,  0.08436573, -0.13285293,  0.04827133,
       -0.24424356, -0.0846597 ,  0.02579471,  0.10821003,  0.12071999,
        0.1082117 ,  0.20302555, -0.16620013, -0.09685103, -0.01705716,
       -0.17135042, -0.04118485, -0.1622989 ,  0.06429579,  0.30334675,
        0.1394438 , -0.31852353,  0.07122932,  0.3088901 , -0.06764276,
        0.21078314,  0.07743206,  0.14414512,  0.02918453,  0.18463081,
        0.3439598 , -0.03584324, -0.29142895,  0.04754777,  0.20154236],
```

```
        dtype=float32)
```

[ ]:
```python
# Another word2vec test to find similar words.
w2v_model.wv.most_similar('film',topn=5)
```

[ ]:
```
[('lives', 0.9979659914970398),
 ('ben', 0.9979487657546997),
 ('jeans', 0.9979398250579834),
 ('jason', 0.9979063868522644),
 ('bloody', 0.9979048371315002)]
```

[ ]:
```python
# Get the index from a word.
w2v_model.wv.key_to_index["film"]
```

[ ]:
```
996
```

1.3.5 Feature extraction function for point 2.1.

2 2.2 MLP implementation.

2.1 Data iterator

In order to get ready the training phase, first, we need to prepare the way how the sequences will be fed to the model. For this purpose, PyTorch provides two very useful classes: Dataset and DataLoader. The aim of Dataset class is to provide an easy way to iterate over a dataset by batches.

Taken from the provided "Natural_disaster_NLP_LSTM.ipynb" file.

[ ]:
```python
from torch.utils.data import Dataset
from torch.utils.data import DataLoader

class DatasetMaper(Dataset):
        '''
        Handles batches of dataset
        '''
        def __init__(self, x, y):
                """
                Inits the dataset mapper
                """
                self.x = x
                self.y = y

        def __len__(self):
                """
                Returns the length of the dataset
                """
                return len(self.x)
```

```
        def __getitem__(self, idx):
                """
                Fetches a specific item by id
                """
                return self.x[idx], self.y[idx]
```

2.1.1 Function extract_features_dataset

```
[ ]: # The number of features for the model or D, according to TP document. And is␣
     ↪set on previous steps while creating the word2vec model.

     w2v_model = word2vec.Word2Vec(min_count=2,
                          window=3,
                          vector_size=D,
                          sample=6e-5,
                          alpha=0.03,
                          min_alpha=0.0007,
                          negative=20,
                          workers=4)

     # Function requested on point 2.1, to generate a dataset using a defined amount␣
     ↪of features (D).
     def extract_features_dataset(model, preprocesed_dataset, max_length_words=100,␣
     ↪num_features=20,batch_size=64):

         # We use the CountVectorizer to convert the collection of text messages to␣
     ↪a matrix of token counts.
         cv = CountVectorizer(max_features=num_features)
         # And create our x array that will be used later on the MLP model.
         x = cv.fit_transform(corpus).toarray()

         # We built our y, using the labels from the dataset.
         y = messages['label']
         # Then transform the labels and prepare y for later use on MLP model.
         le = LabelEncoder()
         y = le.fit_transform(y)


         # Load training data
         # Taken from the provided "Natural_disaster_NLP_LSTM.ipynb" file.
         # We will split the dataset in training and testing sets, will be later␣
     ↪feed to the dataloder.
         xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.20, )

         # create data loaders
         training_set = DatasetMaper(xtrain, ytrain)
         test_set = DatasetMaper(xtest,ytest)
```

```
    loader_training = DataLoader(training_set, batch_size=batch_size)
    loader_test = DataLoader(test_set, batch_size=batch_size)

    return loader_training, loader_test

#Grabs the dataset after calling the function to built with the desired␣
 ↪features.
loader_training, loader_test =␣
 ↪extract_features_dataset(w2v_model,messages['text_pp'],num_features=D)
```

2.2 Create the MLP Model

[ ]:
```
def create_MLP_model():
    # Model creation with neural net Sequential model
    model=nn.Sequential(nn.Linear(D,70), # 1 Layer
                        nn.ReLU(),          # Activation function ReLu.
                        nn.Linear(70,70),  # 2 Layer
                        nn.ReLU(),        # Activation function ReLu.
                        nn.Linear(70,2),   # 3 Layer
                        nn.Sigmoid() #Output activation function Sigmoid.
                       )
    return model

#Calls the functions to create the model.
mlp_model = create_MLP_model()

#Error function, selectec Cross Entropy as per documented in the pdf file.
criterion = nn.CrossEntropyLoss()

#Prints the details of the model.
print("MLP model")
print(mlp_model)
```

```
MLP model
Sequential(
  (0): Linear(in_features=100, out_features=70, bias=True)
  (1): ReLU()
  (2): Linear(in_features=70, out_features=70, bias=True)
  (3): ReLU()
  (4): Linear(in_features=70, out_features=2, bias=True)
  (5): Sigmoid()
)
```

2.3 Train MLP Model

[ ]:
```
#Training function, using as reference the notebook shared in class.
def train_model(model, criterion, epochs = 15, lr = 0.01, is_MLP = False):

    time0 = time()
    running_loss_list= []
    epochs_list = []
    optimizer = optim.SGD(model.parameters(), lr= lr, momentum=0.9)
    for e in range(epochs):
        running_loss = 0

        #go for every batch
        for x_batch, y_batch in loader_training:

            x = x_batch.type(torch.FloatTensor)
            y = y_batch.type(torch.LongTensor)

            # Flatenning
            if(is_MLP):
              x = x.view(x.shape[0], -1)

            # defining gradient in each epoch as 0
            optimizer.zero_grad()
            # modeling for each text batch
            output = model(x)

            # calculating the loss
            loss = criterion(output, y)

            # This is where the model learns by backpropagating
            loss.backward()

            # And optimizes its weights here
            optimizer.step()

            # calculating the loss
            running_loss += loss.item()

        else:
            print("- Epoch {} - Training loss: {}".format(e, running_loss/
 ↪len(loader_training)))

    print("\nTraining Time (in minutes) =",(time()-time0)/60)
    return model

print("### Training MLP model")
```

```
mlp_model = train_model(mlp_model, criterion, epochs = 10, lr = 0.1, is_MLP =␣
↪True)
```

```
### Training MLP model
- Epoch 0 - Training loss: 0.5033365990434374
- Epoch 1 - Training loss: 0.44918989496571676
- Epoch 2 - Training loss: 0.4486369963203158
- Epoch 3 - Training loss: 0.4484444060495922
- Epoch 4 - Training loss: 0.44834481009415217
- Epoch 5 - Training loss: 0.44828164151736666
- Epoch 6 - Training loss: 0.44823518991470335
- Epoch 7 - Training loss: 0.4481964307171958
- Epoch 8 - Training loss: 0.4481601906674249
- Epoch 9 - Training loss: 0.44812259461198534

Training Time (in minutes) = 0.011097991466522216
```

2.4 Test the model

```
[ ]: #Test model function, using as reference the notebook shared in class.
def test_model(testloader, model, verbose = True):

    from sklearn import metrics

    #Variables later used to calculate F1 scores.
    correct_rate, false_negative_rate, all_count = 0, 0, 0
    predictions = []
    true_labels = []

    for x_batch, y_batch in loader_test:

      x = x_batch.type(torch.FloatTensor)
      y = y_batch.type(torch.LongTensor)

      for i in range(len(y)): # Iterate over targets.
        text = x[i].view(1, D)
        with torch.no_grad():
            logps = model(text)
        ps = torch.exp(logps)
        probab = list(ps.cpu().numpy()[0])
        pred_label = probab.index(max(probab)) # Get predcition for current␣
↪iteration.
        true_label = y_batch.cpu().numpy()[i] # Get expected label from current␣
↪iteration.
        true_labels.append(true_label)
        predictions.append(pred_label)
```

```
        if (true_label == pred_label): correct_rate += 1 # Adds to correct_rate␣
↪if the prediction is correct.
        else:
          if (pred_label == 0): false_negative_rate += 1 # False negatives␣
↪count.

        all_count += 1

    if (verbose):
      #Prints summary of the testing.
      print("Messages Tested =", all_count)
      print("True Positive Tests =", correct_rate)
      print("False Positive Tests =", (all_count - correct_rate) -␣
↪false_negative_rate)
      print("False Negative Tests =", false_negative_rate)
      print("\nModel Accuracy (Average) =", np.round((correct_rate/
↪all_count)*100,4),"%")

    #Printing the F1-Scores using the sklearn metrics.
    print("\nF-1 Scores:")
    print(metrics.classification_report(true_labels, predictions))
    print(metrics.confusion_matrix(true_labels, predictions))

    # Printing the Overall Accuracy of the model
    F1_Score=metrics.f1_score(true_labels, predictions,␣
↪average='weighted',zero_division=0)
    print('\nAccuracy of the model on Testing Sample Data:', round(F1_Score,2))

    return correct_rate, false_negative_rate, all_count

#Show tests results.
print("Testing MLP model")
res = test_model(loader_training, mlp_model)
```

```
Testing MLP model
Messages Tested = 1115
True Positive Tests = 968
False Positive Tests = 0
False Negative Tests = 147

Model Accuracy (Average) = 86.8161 %

F-1 Scores:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       968
           1       0.00      0.00      0.00       147
```

```
    accuracy                           0.87      1115
   macro avg       0.43      0.50      0.46      1115
weighted avg       0.75      0.87      0.81      1115

[[968   0]
 [147   0]]

Accuracy of the model on Testing Sample Data: 0.81

C:\Users\jcord\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\jcord\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\jcord\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1334: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```