CS-410 Text Information Systems – Spring 2025

HOMEWORK 3

Handed out: Apr 16, 2024 Due: May 2, 2025 at 23:59

Submission through Canvas: upload a zip file with the name HW3-yourNETID-code.zip before the deadline (above) Penalty for wrong formatting & file naming: as indicated in the Syllabus

LSTM & TRANSFORMERS FOR TEXT MINING, TOPIC MINING
[+ LEARNING TO RANK]
(100 points)

For this homework we will continue working with the News dataset from previous homework. https://github.com/mhjabreel/CharCnn_Keras/blob/master/data/ag_news_csv/test.csv

1 Word Prediction with Word2Vec and LSTM (50 points)

For this question, you will need the `test.csv` file of the News dataset from both previous homework. You will also need to install both TensorFlow (`pip install tensorflow==2.0.0-alpha0`) and Keras (`pip install keras`) – alternatively, you can use Keras from TensorFlow. After installation of the libraries, create a Jupyter Notebook to do the following: 1) Download the news data file above (`test.csv`) 2) Load it with Pandas and run the usual data preprocessing we have done so far (transform to lowercase, remove numbers, punctuation, NAs, etc. stemming is optional) and select the 3rd column as the document content 3) Modify the attached LSTM code to use a 100-dimensional time series (instead of a 3-dimensional one) and predict the next **two words** as follows. Create a 100-dimensional time series, for which you will use a word2vec encoding and create vector words of dimension 100. Predict the next two word vectors. For the final prediction, you have to indicate the actual words (in English) that were predicted. Thus, find the words that are closer in meaning to the 2 word-vectors you predicted. Specifically: 3.1) For word2vec feel free to use a pre-trained word2vec (training is only needed for LSTM) and any Python library but I suggest you use `gensim`. Once you encoded the collection using word2vec, you will have to modify the code to use these 100-dimensional time series which come from the word vectors transitions from one word to the next in the sentences of the news collection. Notice that the example code stacks the data horizontally. This is because each time series is built separately one series at a time. However, since our "time" series come from vector sequences, and thus we are building each element of the 100 sequences at once, you will have to stack the data vertically (`vstack`) and iteratively one-word vector at a time. 3.2) Modify the code to predict the next value of the skipgram word-vector sequence. To do that you will need to modify the code to use the word-vectors from the dataset. Then, you will have to partition the dataset with the entries corresponding to the known words (e.g. first $n-2$ words of a sentence) for training and the unknown words (e.g. last 2 words) for prediction. 3.3) Plot a histogram of the root mean squared error (RMSE) values of the predictions for **both** the `tanh` activation function and the `ReLU` activation function – you can plot them side by side using different colors for each histogram. This will provide an estimate of performance.

Up to this point, you are experimenting with the various documents in the collection of documents, points 3.4 and 3.5 need to be done for a selected document as indicated next: 3.4) Select the activation function (tanh or ReLU) with better distribution of RMSE and predict the last 2 word-vectors of one document. The doc you will choose for this is the one with most words in the collection. 3.5) Find the words that are closer in meaning to the word-vectors you predicted. If you are using `gensim`, you can use the function `model.wv.similar_by_vector` to obtain the words that are closer in meaning to a given word vector(s).

4) Add a boolean variable to alternatively include the stop words in the code. 5) Run the code (Word2Vec encoding and LSTM prediction) but include the stop words when the flag of the previous step is set to `True`. How do the RMSE of the predicted words compare with and without stop words (you can alternatively plot the histogram of F-scores of the matched closest words)? How does the quality of the prediction compare? Write your answers in a README file. 6) Set the flag to the option (without or with stop words) that performed better for the code you submit.

Note. To simplify this problem, you can select only one sentence from each document instead of the full document. That is, you can parse the sentences (before removing punctuation) and work only with the sentence with the most words in each document.

```python
# multivariate lstm code based on Brownlee (2020)
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
  X, y = list(), list()
  for i in range(len(sequences)):
    # find the end of this pattern
    end_ix = i + n_steps_in
    out_end_ix = end_ix + n_steps_out
    # check if we are beyond the dataset
    if out_end_ix > len(sequences):
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
in_seq3 = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
in_seq3 = in_seq3.reshape((len(in_seq3), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, in_seq3))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# covert into input/output
```

```
X, y = split_sequences(dataset, n_steps_in, n_steps_out)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(RepeatVector(n_steps_out))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(n_features)))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=300, verbose=0)
# demonstrate prediction
x_input = array([[60, 65, 125], [70, 75, 145], [80, 85, 165]])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

2 Word Prediction with Transformers - (30 points)

For this problem, you will use the *transformers* library (`!pip install transformers`) from Hugging Face. You will also use Torch and Numpy. Similarly to Problem 1, you will use the `test.csv` file of the News dataset from both previous homework. Unlike LSTM you will use pre-trained models, BERT and GPT2, and thus, no training is needed for this problem (no partitioning of the data in training and test sets, there will be only test sets). 1) Download the news data file above 2) Load it with Pandas and run the usual data preprocessing we have done so far (transform to lowercase, remove numbers, punctuation, NAs, etc. stemming is optional) and select the 3rd column as document content 3) You will use BERT and GPT2 to predict the next (one) word as follows. 3.1) Use the sample code below to create 2 functions (1 for BERT and 1 for GPT3) that use the pre-trained models to forecast the next word given a prefix sequence. 3.2) Create a function or process that iterates over the documents in the collection and use the first $n-1$ words of a sentence as a "prompt" or test sequence and the unknown word (e.g. the last word) for prediction. 3.3) Compare the predicted word with the actual word in the sentence to compute 2 performance estimates: 1 mandatory, the accuracy, and one optional, the cosine-similarity distributions with pseudo-feedback (extra credit, detailed below). Compute the accuracy for both models (number of correctly predicted words/total predictions) - this number should be low for the default BERT and a little higher for GTP2. Now add back the points (the punctuation) at the end of the sentences and run your predictions again - the performance now should be a little higher for BERT.
Note. To simplify this problem, you can select only one sentence from each document instead of the full document. That is, you can parse the sentences (before removing punctuation) and work only with the sentence with the most words in each document.

EXTRA CREDIT (5 points): Run 1 additional evaluation metric, namely, a pseudo-feedback cosine-similarity of a reference embedding. The process is as follows: use a pre-trained word2vec model (reference embedding) to encode both the real word and the predicted word (of both BERT and GPT2) and compute the cosine similarity of their word vectors. Plot the distribution/histogram of cosine similarities for both BERT and GPT2 (although this can vary); plot the histograms in the same plot using different colors for each model. Which one is better? Write your answer in the README file.

Sample code to predict next word with pre-trained BERT

```
from transformers import pipeline
model = pipeline('fill-mask', model='bert-base-uncased')
pred = model("Forecasts of Presidential [MASK]")
print("Predicted next word: ")
pred[0]['token_str']
```

Sample code to predict next word with pre-trained GPT2

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import numpy as np

tokenizer = AutoTokenizer.from_pretrained("gpt2")
model2 = AutoModelForCausalLM.from_pretrained("gpt2")

seq = "Forecasts of Presidential"

inputs = tokenizer(seq, return_tensors="pt")

input_ids = inputs["input_ids"]
for id in input_ids[0]:
  word = tokenizer.decode(id)

with torch.no_grad():
  logits = model2(**inputs).logits[:, -1, :]

pred_id = torch.argmax(logits).item()

pred_word = tokenizer.decode(pred_id)
print("Predicted next word: ")
print(pred_word)
```

3 Topic Modeling - Multinomial PMM Model and LDA(20 points)

For this problem, you can use the `test.csv` file from the News dataset. Each row corresponds to a document and the third column is the content of the documents. For this problem, you will use the `gensim` library to identify topics in the collection. The following code shows how to use LdaModel:

```
# Create the object for LDA model
lda1 = gensim.models.ldamodel.LdaModel

# Train the LDA model using the document term matrix.
ldamodel = lda1(matrix_of_doc_term, num_topics=10, id2word = D1, passes=100)
```

`matrix_of_doc_term` can be computed with the `doc2bow` function by providing a document from the corpus to the function. The matrix is built by computing `doc2bow` for every document in the corpus. The steps are:

```
1. create the dictionary D1 with gensim.corpora.Dictionary (use the collection)
2. compute the matrix_of_doc_term by computing D1.doc2bow for every document
in the collection
```

Finally, use `ldamodel` to plot:

a. The 5 most relevant terms for one of the topics
b. Plot a matrix comparing the distance/difference of the 10 topics. Use the Kullback-Leibler divergence and 50 words. e.g. `ldamodel.diff(ldamodel, distance='kullback_leibler', num_words=50)`. You can use any library for the plots but `Matplotlib` or `ggplot` are recommended.

4 EXTRA CREDIT - Logistic Regression for Learning to Rank with PageRank, HITS, and TF-IDF (25 points)

Create a function that trains a learning-to-rank algorithm for classifying web-pages as relevant or not, using logistic regression, as follows. Use the set of pages you downloaded with the web crawler you build in HW2 to build a graph of URL links. Out of the ten pages, select a query or topic for which *approximately* only half of the pages will be relevant and half irrelevant. If the pages you downloaded for HW2 are not easy to label this way, choose and download a different set. You will use this manual identification of relevant and non-relevant pages as your ground truth of labels as indicated below. Use the content of the HTML page as content of the document itself, index the pages as you wish (e.g. titles, file prefixes, numbers from 1 to #_of_pages, etc.) but number them (for the matrix representations of PageRank and HITS) and use the numbers as the indices of the nodes, and use the hyperlinks as edges in the graph. Specifically, use the page id (e.g. a number or similar) as the node and the hyperlink (`<a ...></a>`) as an edge between two pages. For instance if the page `www.nytimes.com` links to your last page, say `https://www.nytimes.com/2023/11/10/technology/personalized-ai-agents.html` you can index the first as node 1 and the second as node 10 and so on, then, you could assign a link between nodes 1 and 10 (alternatively, 0 and 9). You will use the labeling of *relevant* pages you created manually (indicated above) and build a dataset as follows:

Relevant?	PageRankScore	HITSScore	TF_IDF
$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$
$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$
…			
$y^{(10)}$	$x_1^{(10)}$	$x_2^{(10)}$	$x_3^{(10)}$

Table 1: Dataset

Compute both HITS and PageRank scores (you don't have to implement either PageRank nor HITS and you can use any library of your choice), in addition to the TF-IDF score of the pages. Arrange the three scores for each page and for the cases neded consider a set of words of your interest as shown in Table 1. For instance, you can use: [reuters, stocks, friday, investment, market, prices] as words that are relevant for the topic `financial markets`.

Then, use Scikit-learn (link) to *apply* a logistic regression classifier (no coding from scratch is expected for logistic regression) to classify documents that are relevant from those that are irrelevant using the dataset you just created. Thus, you are building a learning-to-rank program that will consider not only the content of the pages but the hierarchy of importance of the documents to rank them. Partition your data into training and test sets by using 60% pages as training and 40% as testing. Run the prediction of relevance for your pages and report the precision and recall curve and F1-scores (for both training and test sets). Make sure that the partitions have an equal representation of each of the two types of pages in it (relevant and not relevant).

WHAT TO TURN IN

Mandatory (Graded) elements/files within the zip file:

1) The Jupyter notebook with the solutions to the problems (please, annotate & comment your code) and

2) a README file with the answers to the questions in the problems

3) (Extra Credit) For the Learning-To-Rank question: In addition to the notebook (can be a separate notebook but it is better to use the same), submit a PDF with a plot of the results and a one-paragraph description of what you found most interesting.

Optional (no points will be awarded for this!):

You can also include in the README file **a URL link to your Colab file** where you can show what you did in case that helps check what you did. However, this is just to verify outputs, and the code in your Collab repositoty will not be considered for grading. Only the submitted code will be graded.

REFERENCES

Brownlee, J. How to Develop LSTM Models for Time Series Forecasting. (2020).