Course 52017 – Project Summary

Yoel Graumann

Nofar Gerassi

Introduction

In this project we decided to perform document classification on the arXiv PDF dataset. We wanted to see if we could extract decent features out of pdf files and perform multiclass classification using different models. At first, we wanted to extract features from all the available data in the cluster, but running our preliminary code caused us to reevaluate our decision. Basically, running our preliminary code on a single pdf file takes a long time, and we found it ineffective to perform the same analysis over all the data, when we could come to the same conclusions when running the code on a subset of the data. Additionally, we ended up running our code locally, mainly because we liked the freedom of running our code when we wanted to, and we did not have to wait for the server / cluster to run it.

Just as an example, we ended up running our final code locally on 1,750 PDF files, which took about 3 hours, running on the following Computer specifications:

- Processor 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz, 3110 Mhz, 4 Core(s), 8 Logical Processor(s)
- Installed Physical Memory (RAM) - 16.0 GB

To reiterate, we decided to perform multiclass classification, we ended up picking the following 5 categories: Astrophysics, Computer Science, Mathematics, Physics and Statistics. We ended up looking at the main categories of the arXiv data, and not on the subcategories. We picked these 5 categories because these were the subjects, we are most interested in, personally.
We sampled 350 papers of each category (without replacement), such that 1,750 pdf files were processed and “converted” into features.
Our project is split into 2 Jupyter Notebooks,
Part 1 performs the preprocessing, and part 2 performs the modelling and inference.

Preliminaries

When starting the project, we also used the json metadata file which was provided in the cluster. We used this data to keep track of the count of each pdf category in the data.
After some coding, we found out that we had, in our local 2003 data, 1,071 Astrophysics pdf files, 4,129 Computer Science pdf files, 3,178 Mathematics pdf files, 1,338 Physics pdf files and 380 Statistics pdf files. We wanted to sample equally from every category, so we have a balanced dataset. One way of getting our dataset to be balanced is by performing "undersampling". Undersampling is typically a method of simply using most of the "rare events" (statistics pdf in our case in our case) and reducing the number of "abundant events" (the other categories). The main advantage of undersampling is that we can correct imbalanced data to reduce the risk of our analysis and models skewing towards the majority classes. Another practical advantage is that we require less storage, and our runtime is sped up by a lot, since we do not use all the provided data. On the other hand, we are losing potentially important data. But since the processing step takes so much time for a single pdf file, we decided to go with under sampling.

Preprocessing step

The preprocessing step is performed inside the first notebook.
We decided to use the PyPDF2 library to preprocess the data. Pypdf2 is an open-source python pdf library capable of splitting, merging cropping and transforming the pages of the pdf files. We mainly used it to retrieve the text metadata from the pdf files.
One main problem with the provided pdf files is that it holds the class / category in the metadata. In other words, when extracting the text using the pypdf2 library, the metadata has the category of the file itself inside the text. To make this point clearer, let's show a few examples:

Yan Han*
Brain Machine Interface Lab
The University of Texas at Austin
Austin, Texas

Ahmed H Tewfik
Brain Machine Interface Lab
The University of Texas at Austin
Austin, Texas

***Abstract*—In this paper we demonstrate predicting electroencephalography (EEG) features from acoustic features using recurrent neural network (RNN) based regression model and generative adversarial network (GAN). We predict various types of EEG features from acoustic features. We compare our results with the previously studied problem on speech synthesis using EEG and our results demonstrate that EEG features can be generated from acoustic features with lower root mean square error (RMSE), normalized RMSE values compared to generating acoustic features from EEG features (ie: speech synthesis using EEG) when tested using the same data sets.**

***Index Terms*—electroencephalography (EEG), deep learning**

I. Introduction

Electroencephalography (EEG) is a non invasive way of measuring electrical activity of human brain. EEG sensors are placed on the scalp of a subject to obtain the EEG recordings. The references [1]–[3] demonstrate that EEG features can be used to perform isolated and continuous speech recognition where EEG signals recorded while subjects were speaking or listening, are translated to text using automatic speech recognition (ASR) models. In [4] authors demonstrated synthesizing speech from invasive electrocorticography (ECoG) signals using deep learning models. Similarly in [2], [5] authors demonstrated synthesizing speech from EEG signals using deep learning models. In [2], [5] authors demonstrated results using different types of EEG feature sets. Speech synthesis and speech recognition using EEG features might help people with speaking disabilities or people who are not

process which requires the use of specialized EEG sensors and amplifiers, thus having a computer model which can generate EEG features from acoustic features might also help with speeding up the EEG data collection process as it is much easier to record speech or audio signal, especially for the task of collecting EEG data for performing speech recognition experiments.

In [6] authors demonstrated medical time series generation using conditional generative adversarial networks [7] for toy data sets. Other related work include the reference [8] where authors demonstrated generating EEG for motor task using wasserstein generative adversarial networks [9]. Similarly in [10] authors generate synthetic EEG using various generative models for the task of steady state visual evoked potential classification. In [11] authors demonstrated EEG data augmentation for the task of emotion recognition. Our work focuses only on generating EEG features from acoustic features.

We first performed experiments using the model used by authors in [5] and then we tried performing experiments using generative adversarial networks (GAN) [12]. In this work we predict various EEG feature sets introduced by authors in [2] from acoustic features extracted from the speech of the subjects as well as from acoustic features extracted from the utterances that the subjects were listening.

Our results demonstrate that predicting EEG features from acoustic features seem to be easier compared to predicting acoustic features from EEG features as the root mean square error (RMSE) values during test time where much lower for

being modeled. In this work, we study Rayleigh-Bénard convection (RBC) with mixed ("FT") temperature boundary conditions. We aim to understand how FT boundaries change the nature of the convective solution compared to the traditional choice of fixing the temperature at the top and bottom of the domain ("TT"). We find that simulations with FT boundaries experience a long thermal rundown while simulations with TT boundaries do not. In the relaxed state, the mean behavior of an FT simulation corresponds to an equivalent simulation with TT boundaries. The fast evolution of TT simulations can therefore be taken advantage of to rapidly relax FT simulations. We show that these findings carry over to more complicated problems through a brief examination of rotating convection. We furthermore find that FT boundaries introduce minor asymmetries into the flow fields, with the fixed flux boundary producing more extreme events than the fixed temperature boundary, but that these asymmetries do not appreciably affect the bulk flows. We conclude that thermal relaxation occurs in two stages: (1) changes to the experimental energy reservoir and (2) changes to the stratification of the experiment. Through the proper choice of boundary conditions or initial conditions, the first of these stages can be bypassed, and the second stage seems to be irrelevant for RBC.

I. Introduction

Convection is a crucial heat transport mechanism in the atmospheres and interiors of stars and planets. Numerical simulations are a commonly-used tool in studies of geophysical or astrophysical convection. These studies range from examinations of convection in the simplified Boussinesq approximation [1–3] to highly complex "dynamo simulations" which include magnetism and atmospheric density stratification [4, 5]. Regardless of complexity, numerically simulated convection is fundamentally driven by some combination of imposed boundary conditions and internal heating profiles [6]. In studies of Boussinesq convection, the standard choice is to hold constant the temperature difference across the domain by fixing the temperature at the upper and lower boundaries. However, a common choice of thermal boundary conditions in astrophysical convection [7–13] is to fix the flux entering the domain through the bottom boundary and to fix the value of a thermodynamic quantity (e.g., temperature or entropy) at the top boundary. We are unaware of any study which has examined the consequences of imposing the "mixed" boundaries that are frequently favored in astrophysical convection studies.

In this work, we examine how the choice of using "mixed" boundaries affects the evolved nonlinear convective state in the simplest possible model: Rayleigh-Bénard convection (RBC) under the Boussinesq approximation. In RBC, temperature is the only thermodynamic quantity and throughout this work we will refer to the choice of fixing the flux at the bottom and temperature at the top as "FT" boundary conditions. We will refer to the common choice of fixing temperature at both boundaries as "TT" boundaries, and fixing the flux at both boundaries as "FF" boundaries[1]. It is generally assumed that FT and FF boundaries should fundamentally behave similarly [6], and the behavior of FF boundaries is well-known [14, 15]. FF and TT boundaries exhibit the same scaling of convective heat transport (quantified by the Nusselt number, Nu) as a function of increased convective driving (quantified by the Rayleigh number, Ra) [15]. It is natural to assume that FT simulations should follow the same Nu vs. Ra scaling laws as FF and TT boundary conditions. However, FT boundaries introduce complexities into the convective solution which neither FF nor TT boundaries are exposed to. First, the evolved mean temperature of a simulation with FT

These 2 pdf files were picked randomly, as you can see in the picture, they all have the category / class itself inside the pdf file, specifically on the left side. Although this is not the case for all PDF files in the database, this is the case in many pdf files provided, and when extracting the metadata using PyPDF2 we had the category as part of the pdf text. In order to fix this problem, we create a function which will scan the PyPDF2 output for the class and we removed it from the text. This task is completed by the function called "remove_category" in our code.

An important side note is that we are using PyPDF2 to turn ALL the pdf pages in the file into one string, then we process it. Another thing to note here is that we ignored all latex / plots in the pdf itself. We only extracted text data from the pdfs.

Once the category leak has been dealt with, we move on to cleaning the text itself and tokenizing, lemmatizing etc. Our first step is parsing the string using beautifulsoup library, this library makes it easy for us to process the data in the next steps. In second step we removed / replaced certain characters / strings that we found to be uninformative. We then turned the string into lower case.

We then performed tokenization, which simply turns every word in our string into a different token, we are breaking down sentences into their building blocks. We used the already built nltk library to help us with tokenization. The NLTK tokenization turned our single big string made of all the pdf pages into a list of substrings. We decided to NOT include words that appeared less than twice in the text. This was done to remove a lot of "noise" which the PyPDF2 library extracted from the pdf metadata.

Our final step in the text cleaning process is the lemmatization step, we lemmatized all tokenized words in our data. Lemmatization is an NLP preprocessing step which groups together different inflected forms of the same word. In other words, we reduce a word to its root form, called a lemma. For example, the word laughing, laughed, laughs will all be identified as the word "laugh".

Some advantages of lemmatization:

- Improved efficiency in text processing, as it reduces the dimensionality of the data by reducing words to their base form.
- Better representation of the text data, as it allows for better clustering and classification - which is what we're doing here.
- Lemmatization can help with comprehension of the meaning of the text data, as it can help identify the underlying concepts and themes in the text (in our case the category).

Even though lemmatization is more resource intense, we decided to run lemmatization over stemming. Mainly because lemmatization is a linguistically motivated procedure while stemming is more of an approximation and from what we read online, the results are worse. Note that we decided to remove words that appeared in our premade *stopwords* list. *Stopwords* are words which are commonly used, such as "the" "is" "a" etc. By removing these unimportant words, we allow the model to focus on the more important words instead. Once we were done with lemmatization, we started vectorization.

Vectorization:

We tried many vectorization techniques before we ended up settling for TFidvectorizer.
At first we tried to use the doc2vec algorithm.
The doc2vec algorithm is like the popular word2vec algorithm. But This model is used to create a vectorized representation of a group of words, taken collectively as a single unit. It doesn't give the simple average of the words in the sentence.
We believe that a very big weakness of this model, at least for our use case, was that we had to train the model on the pdf files. We had to feed the model the pdf files we wanted to extract features from. This added more steps for our code and increased our project runtime by a lot. Another reason we ended up not using doc2vec was because the performance of the classification models which used the features from doc2vec were subpar. We had an accuracy of about 50% using logistic regression when using doc2vec features. Therefore, we gave up on this vectorization method. We got similar results for word2vec and gave up on it too.
Another vectorization method we tried was to download the premade embedding from huggingface (https://huggingface.co/). Hugging face is an American company that develops tools for building applications using ML/ AI. They host on their website many pretrained transformers. What we tried to do is use a pretrained model to turn our words into embeddings. Specifically, we used this model. This model is intended to be used as a sentence and paragraph encoder. Given an input text, it outputs a vector which captures the semantic information. Sadly, using the output vector from this pretrained model did not work well when we tried to perform classification using our logistic regression model. Again, we got accuracy of about 50%.
Finally, we ended up using TF-IDF vectorizer – This algorithm measures the originality of a word by comparing the number of times a word appears in a document with the number of documents the word appears in. This vectorizing method ended up looking the most promising as we will see in our model part. All our models were fit using the features created by tfidf vectorizer. Once we got the features, we downloaded them as a pickle file and moved on to part 2 of our project.

Models

Using our pickle data, we fitted 3 different models: Logistic Regression, Multilayer perceptron and naïve bayes.
All three models had an accuracy of above 80%. This part of the project was a lot easier than the first one. Overall, the models were working well. The comparisons of the models are provided in the pdf file of part 2.