Progress Report - Oct.6

Data Balancing

1. Undersampling - same amount of articles per country

Data Preprocessing

We might adjust this procedure after seeing our actual data:

1. **Remove url and email address**
2. **Remove named entity (NLTK & SpaCy package)**
3. Remove stopwords (NLTK (179 words) & SpaCy (326 words) & Genism (337 words))
4. **Remove special characters**
5. Remove Plural (PorterStemmer) (Not sure if we need this step in BERT)
6. Unify verb tense (WordNetLemmatizer: unify verb tense) (Not sure if we need this step in BERT)
7. Cast to lower

Exploratory Data Analysis

1. Check article distribution across peace & non-peace countries
2. Check average article length before & after preprocessing

Next Step

1. Check to see if current BERT is using lemmetizers or not
2. Practice next steps of preprocessing (removing stop words, etc.) on sample JSON files
3. Do research on how people handle differences between the U.S. and U.K. English language; find out which version of BERT to use
4. Decide whether to use stemer or not
5. Once have data: Make a copy of Philippe's S3 bucket in AWS files into students' own S3 bucket so the original data will not be affected.

Side Notes

Rule-of-5 Countries

High Peace: Australia, New Zealand, Belgium, Sweden, Denmark, Norway, Finland, Czech Republic, Netherlands, Austria

Low Peace: India, Nigeria, Iran, Sri Lanka, Kenya, Congo, Zimbabwe, Uganda, Afghanistan, Guinea

Matched-pairs Countries:

High Peace: Ghana, Chile, Canada, Denmark, Japan, Singapore, Czech Republic, United Arab Emirates

Low Peace: Congo, Brazil, United States, Latvia, China, Philippines, Russia, Iraq

Progress Report - Oct.13

Data preprocessing in BERT preprocessor

- Stopwords are usually kept for BERT, because some stopwords might carry some context, and BERT usually don't need input to be preprocessed with lemmatizing & stemming due to the same reason.
  Ex: not, nor, never are stopwords, but have strong contextual meanings.

- BERT cased
  - Tokens for a sentence with capital letters is not the same as tokens for the same sentence all in lowercase
    Ex: us != US ; for us meaning we, and US meaning the country or the currency name like US$
- BERT uncased
  - Convert everything to lowercase first, then do the tokenization

  -> We don't need 'Convert to lowercase' in our own preprocessing step since the BERT will handle it depending on which version we are going to use.

- BERT has a WordPiece Model which can handle the subword tokenization as following
  Ex: embeddings -> 'em', '##bed', '##ding', '##s'
  Note: Some words might have different word embeddings for different tense:
  'announce' vs. 'announced' are both in the BERT word bank

Practice next steps of preprocessing (removing stop words, etc.) on sample JSON files

- We will update the code and upload the file to GitHub when we have access to the data

Do research on how people handle differences between the U.S. and U.K. English language; find out which version of BERT to use

- Option 1: Convert U.K. English to U.S English and then do the U.S English BERT (or ALBERT, ELECTRA, etc)

Reference:
https://blog.tensorflow.org/2020/12/making-bert-easier-with-preprocessing-models-from-tensorflow-hub.html

Next Step

1. Can you run BERT on AWS? Is there a way to estimate the computation?
2. Take some samples from full dataset, in a logarithmic way, i.e. 1,2,4,8 (not arithmetic or linear way i.e. 1,2,3,4, megabytes) increasing it by a factor of 10 or 2 every time, running on a few small samples of data so we can get an estimate of run times and expense
3. Find which pretrained model is appropriate to use on the data we have

Progress Report - Oct 20

Run BERT in SageMaker

- Using HuggingFace with AWS SageMaker is more commonly seen than using Tensorflow, but both packages are available on SageMaker.
- AWS has a direct collaboration with HuggingFace -> Easy-to-use package to load & fine-tune in any pre-trained BERT models on HuggingFace.
- HuggingFace Models usually can be loaded as Tensorflow layers (haven't checked if this is true on SageMaker yet).
    - If it is true, then we can build only one model pipeline: use the Tensorflow package on SageMaker for the model, then insert the HuggingFace BERT model as tf layer.
    - Otherwise, we can try each package separately.

Helpful Tutorials

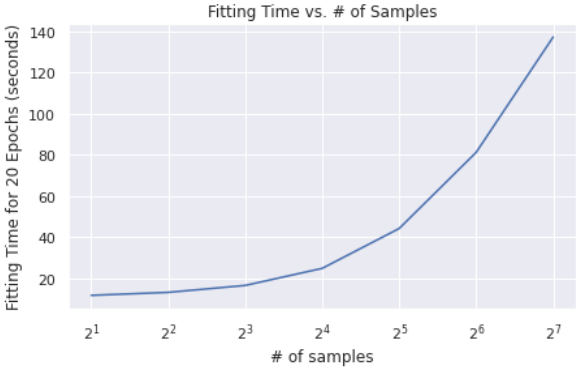
- Use BERT from HuggingFace:
    - Train a Hugging Face Transformers Model with Amazon SageMaker
- Use BERT from Tensorflow: helpful and short tutorial for beginners, covers loading data from s3 bucket, preprocessing it, storing the result file back to s3, then start a tensorflow model; might be easier to understand than the first tutorial)
    - Part 1 - Deploy TensorFlow Models on Amazon AWS SageMaker
    - Part 2 (final) - Deploy TensorFlow Models on AWS SageMaker

AWS Sagemaker Official Tutorial

https://aws.amazon.com/getting-started/hands-on/build-train-deploy-machine-learning-model-sagemaker/

BERT Computation Time Estimate

- BERT-base-uncased + one output dense layer on Colab GPU
- Average the runtime of 5 trials, 20 epochs per trial



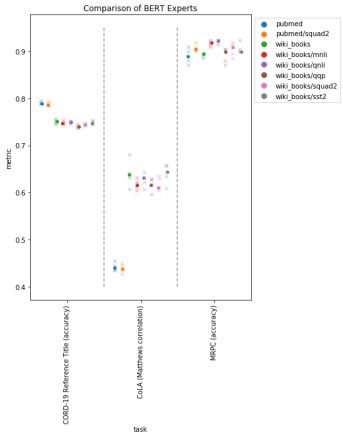
Find which pretrained model is appropriate to use on the data we have

Model	Platform	Dataset	Model Size	Vocab Size	Other Info
RoBERTa-base	HuggingFace	Pretrained on: • BookCorpus, a dataset consisting of 11,038 unpublished books • English Wikipedia (excluding lists, tables and headers) • CC-News, a dataset containing 63 millions English news articles	12-layer, 768-hidden, 12-heads, 125M parameters (vs. 110M parameters in BERT-base)	50k	SST-2 Score: 94.8 MRPC: 90.2

		crawled between September 2016 and February 2019 • OpenWebText, an open source recreation of the WebText dataset used to train GPT-2 • Stories a dataset containing a subset of CommonCrawl data filtered to match the story-like style of Winograd schemas.			
experts/bert/wiki_books/onli	Tensorflow	(Fine-tuned version of BERT-base-uncased) Pretrained on: • English Wikipedia and BookCorpus Fine-tune: • QNLI Task	110M	30k	Best performance on MRPC among other models fine-tuned for different tasks: Score of 90+
textattack/distilbert-base-uncased-ag-news	HuggingFace	Pretrained on: • Distill-BERT has a pretrain dataset same as BERT(English Wikipedia and BookCorpus) Fine-tune: • Adversarial Attack Approach on News Category Classification	66M	30k	ag_news sequence classification: 94.7

Reference:

All Tensorflow BERT Experts Performance



CORD-19 task: COVID-19 Open Research. Search question answers among a large set of COVID related scientific papers

CoLA task: The Corpus of Linguistic Acceptability. Classify sentences as grammatical or not grammatical

MRPC task: sentence semantic comparison & classification on online news articles.

- The Microsoft Research Paraphrase Corpus (MRPC) is a corpus of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent.

QNLI task: determine whether the context sentence contains the answer to the question.

SST-2 task : binary sentiment analysis/classification on IMDb movie reviews

Question:

1. Should we preprocess all of the data and save the results in a s3 bucket or should we put the preprocessing steps in the model?
    - **Pros** for saving results:
        - Save time for model training, reduce the cost on GPU usage
        - Only have to do this once, and the result would be useful for all BERT models we'd like to explore.
    - **Cons** for saving results:
        - Need more s3 bucket resources.
    - **Solution:** Saving data to S3 Storage is much cheaper. But we need to know how much data will be generated from the pipeline in order to set up the best AWS service.

Progress Report - Oct 27

Here is the link to Capstone Progress Report 1

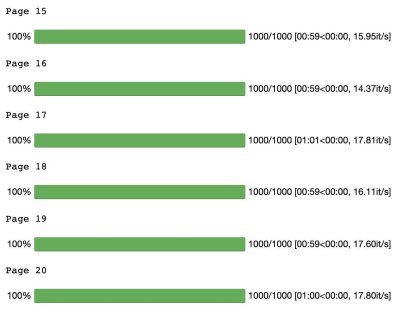
- Let us know if you have any trouble in accessing this file
- Codes for report 1 is uploaded to Github

Big Data EDA proposal (it is also mentioned in the above official progress report 1)

- Iterative EDA
  - **Step 1:** Take a random subset of real data in a logarithmic way, increasing the subset size by a factor of 10 each time
  - **Step 2:** Measure some metrics, perform analysis, and make conclusions on this subset
  - **Step 3:** Update previous conclusions if necessary, or terminate if no major updates
- Metrics to measure:
  - Article count distribution across country in the sample
  - Article length distribution comparison between high-peace and low-peace
- Once we reach a sample set where the iterative process terminates, export the result as a large .csv or .json with multiple records in the file, instead of one record per file (save time accessing the data). It could be the data we are going to study (train+test), because the data distribution is stabilized.
- We plan to include the detailed EDA on real data in report 2

Preliminary EDA observations

1. 1 min loading time per 1K .json file → 100k data take ~ 1.5 hours to load (using ml.t3.medium)



2. Article distribution across low peace countries is highly imbalanced
   Question 1: How are the articles in each folder ordered? We couldn't get a sample from 'Guinea' after scanning 100K articles from the beginning. (See image on the left)
   Question 2: Is the real data count the same as what we have in this list in previous slides? Can we get a full version of the list? (Image on the right is cropped to fit the space here)

country	filepath
Australia	76337
Austria	414
Belgium	5327
Czech Republic	572
Denmark	1338
Finland	1063
Netherlands	796
New Zealand	8920
Norway	1316
Sweden	3917

country	filepath
Afghanistan	240
Congo	526
India	91749
Iran	2368
Kenya	507
Nigeria	2706
Sri Lanka	1356
Uganda	269
Zimbabwe	279

Countries	#LN English Docs
United States	249311438
United Kingdom	74264398
India	52201503
Australia	27144954
Pakistan	26684663
Jordan	18306078
Canada	13446437
China	9476713
Russia	4979521
Japan	4252451
Germany	3776102
South Africa	3218149
New Zealand	3005247
Singapore	2911760
Thailand	2834614
Ireland	2351140

Todo next

- ☑ ~~Get the EDA report on the real dataset~~

- [x] ~~Export the subsampled data as one large .csv or .json file (or as train file and test file)~~
- [x] ~~Preprocess the data and store the result~~
- [x] ~~Fine-tune RoBERTa model using cleaned-up dataset and store the result model~~

Progress Report - Nov 3

Solve the small files problem:

- **Problem:** The dataset we received last week is stored as many individual .json files with < 20KB each. This would cause a **small file problem**, which means the program would experience a huge computation latency in the opening/closing (or using http GET method to fetch the content) of each file while extracting the data. Also, there are **many columns in each file that we don't need**, which further adds more computation time on allocating the memory blocks for those extra data every time we want to load it in a notebook. As we explored last week, it took 1min to read 1K data from s3 using the basic ml.t2.medium notebook instance on SageMaker. So the time estimation to load in 2M data points is 32 hours, which is super long and unstable. If we want the program to finish within a reasonable time range, then we could not get the desired amount of data with the current loading speed.

- **Solution:** we decided to read in a large sample of dataset we need at once, and save them as a large .csv for later re-use. But we still experience the file opening/closing/fetching latency. We made several adjustments below to the codes to optimize the file loading time:
  1. **Use the ujson package instead of json.** This can speed up the json reading time by ~2.5x. Benchmark Statistics Source: https://artem.krylysov.com/blog/2015/09/29/benchmark-python-json-libraries/
  2. **Multi-threading the process**: This can speed up the reading process by ~12x.

```
In [77]: time_start=time.time()
         c = Counter()
         with Pool() as p:
             for page in page_iterator:
                 for file in page['Contents']:
                     filepath = file['Key']
                     obj = client.get_object(Bucket=data_bucket_name, Key=filepath)
                     data = obj['Body'].read().decode('utf-8')
                     stringio_data = io.StringIO(data)
                     io_data = stringio_data.readlines()
                     json_data = list(map(js.loads, io_data))
                     c[json_data[0]['source']['location']['country']] += 1
         time_end=time.time()
         print(time_end-time_start)

         47.53581094741821

In [78]: time_start=time.time()
         file_names = []
         for page in page_iterator:
                 for file in page['Contents']:
                     filepath = file['Key']
                     file_names.append(filepath)
         pool = ThreadPool(24)

         def runcommand(filepath):
             obj = client.get_object(Bucket=data_bucket_name, Key=filepath)
             data = obj['Body'].read().decode('utf-8')
             stringio_data = io.StringIO(data)
             io_data = stringio_data.readlines()
             json_data = list(map(js.loads, io_data))
             c[json_data[0]['source']['location']['country']] += 1

         c = Counter()
         for i in range(len(file_names)):
             pool.apply_async(runcommand, args=(file_names[i],))
         pool.close()
         pool.join()
         time_end=time.time()
         print(time_end-time_start)

         5.995022058486938
```

  3. **Run the notebook on a larger instance:** Throughout the optimized data loading process above, all available threads are working at its highest IO capacity. But we are frequently raising http GET requests which consumes lots of network bandwidth, and it's time consuming for the incoming data to allocate memory blocks in the notebook RAM until we export them in batch. So we switched to using a larger notebook instance that might cost more budgets but can assist the job getting done faster. We finally used a ***ml.m5.4xlarge*** instance with 16 CPUs and 64 GB RAM to perform 2 million data loading, compression, and dumping to S3 bucket. This instance gives more CPUs and available threads, allowing us to run more programs in parallel. It also has a larger memory space, allowing a faster memory allocation process per data record. It also provides double-sized network bandwidth, allowing us to load/dump files faster across S3 and SageMaker. In conclusion, this adjustment can speed up the program from all perspectives.
     More on SageMaker notebook instances spec.: https://aws.amazon.com/ec2/instance-types/

- **Final Results:**
  - The optimized program and newly selected instance took ~40 min to load 1M data points, (2.5 seconds per 1K data points). That is a ~24x speed up overall.

- The gathered data are now stored at the ***compress-data-sample*** bucket as **compressed1m_lp.csv** and **compressed1m_hp.csv**.

EDA Result on word count

- We finally sampled 1 Million data from each peacefulness folder. (2 Millions in total)

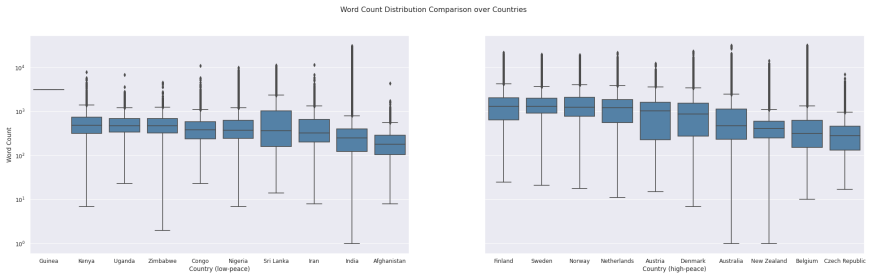
country	sample_perc	avg_wordCount	wordCount_CI_95
Afghanistan	0.2328	221.809708	(213.6184, 230.001)
Congo	0.4985	489.747041	(477.5319, 501.9622)
Guinea	0.0001	3121.000000	(nan, nan)
India	91.7565	422.663411	(421.0335, 424.2933)
Iran	2.3680	484.964443	(479.5853, 490.3436)
Kenya	0.5157	607.522397	(594.3597, 620.6851)
Nigeria	2.7022	579.226482	(571.1824, 587.2705)
Sri Lanka	1.3659	831.227542	(810.2969, 852.1582)
Uganda	0.2902	575.023777	(560.7757, 589.2719)
Zimbabwe	0.2701	566.279156	(551.8202, 580.7381)

NonPeace

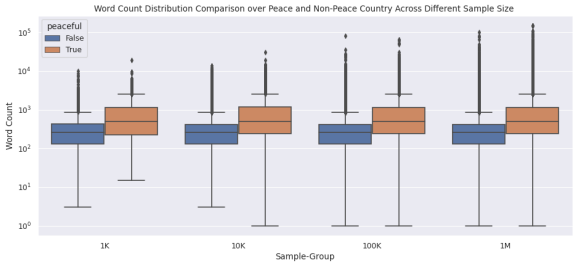
country	sample_perc	avg_wordCount	wordCount_CI_95
Australia	76.0521	884.916874	(882.2838, 887.55)
Austria	0.4210	1240.742993	(1200.2169, 1281.2691)
Belgium	5.3665	703.747545	(683.1177, 724.3774)
Czech Republic	0.5935	415.039596	(402.2787, 427.8005)
Denmark	1.3845	1281.044854	(1252.5792, 1309.5105)
Finland	1.0927	1807.282694	(1767.3396, 1847.2258)
Netherlands	0.8204	1566.497075	(1530.0766, 1602.9176)
New Zealand	9.0610	526.680709	(522.3605, 531.0009)
Norway	1.2701	1851.772459	(1815.7198, 1887.8251)
Sweden	3.9382	1688.450307	(1673.9758, 1702.9249)

Peace

- **Observation:** As we have concluded before, the data distribution in each peaceful group is highly imbalanced between countries. In the non-peace country group, non-Indian data only makes up less than 10% of data, and we only have 1 data point from Guinea among the 1 million samples. In the peace country group, though the balance is slightly better, we could still see that most of the data come from Australia.
  - **Adjustment to data balancing Approach:** Initially, we would like to balance the data by taking the same amount of articles from each country, so that the outcome can have data balance between countries within each peace group as well as across peace groups. However, due to the long computation time of iterating over the single .json files in order to find a datapoint like Guinea, such an approach is not very realistic. So we decided to go by our second option, which is to **take the same amount of articles from the peaceful group and the non-peaceful group**.



- **Observation:** Guinea is an outlier country. It only has one sample in the dataset. There is no obvious pattern between the average and the variance of word count per country in general. Most of the article length is within 1000 words.



- **Observation:** The median of word count in each peace group doesn't change much as we include more samples to our dataset. But the variances of two groups increased and more large outliers appear in the dataset. (The y-axis is log based. So that the boxplot is easier to look at.)

		wordCount							
		count	mean	std	min	25%	50%	75%	max
group	peaceful								
1K	False	1000.0	480.064000	839.020144	3.0	130.75	259.0	425.25	9797.0
	True	1000.0	884.981000	1199.921103	15.0	224.75	494.5	1130.00	18908.0
10K	False	10000.0	436.083400	750.234108	3.0	128.00	257.0	413.00	13869.0
	True	10000.0	918.899500	1249.580898	1.0	239.00	497.0	1170.00	30324.0
100K	False	100000.0	434.750620	807.295570	1.0	131.00	258.0	415.00	82669.0
	True	100000.0	904.128400	1271.521732	1.0	238.00	495.0	1146.00	64822.0
1M	False	1000000.0	435.602821	791.469142	1.0	131.00	259.0	416.00	101327.0
	True	1000000.0	906.523503	1315.915010	1.0	238.00	493.0	1146.00	149973.0

- **Observation:** This is a detailed stats table of the boxplot above to further validate our previous observation. We raised a hypothesis that high-peace countries usually have longer articles compared to low-peace countries.
  - **Suggestions to the BERT tokenization pipeline:** Most of BERT tokens took 512 tokens as max input length. Using that number can allow information in the data to be preserved to the maximum. However, we might use more [PAD] tokens in articles from non-peace countries than peace countries. Though BERT would apply an attention mask to ignore the [PAD] tokens, it is still an unknown question of whether the [PAD] tokens would have any effects in BERT's prediction result, because deep learning models are hard to explain. So we might consider to use a smaller input length to 1) confound this input length differences in peace vs. non-peace countries, and 2) to speed up the BERT training time on large dataset

Compare between High Peace and Low Peace

Hypothesis 1: High-peace countries tend to have longer English articles than low-peace countries

- H0: Distribution of article length from high-peace country = Distribution of LP article length from low-peace country
- H1 Distribution of article length from high-peace country > Distribution of LP article length from low-peace country

```
In [68]: scipy.stats.wilcoxon(hp_df.wordCount, y=lp_df.wordCount,
                              zero_method='wilcox', correction=False, alternative='greater', mode='auto')
Out[68]: WilcoxonResult(statistic=374641694823.0, pvalue=0.0)
```

Conclusion: The alternative hypothesis is true. Articles from high-peace countries are significantly longer than articles from low-peace countries.

- **Observation:** To validate our hypothesis above, we performed a wilcoxon ranked distribution test**.** Based on the test result, we accept the hypothesis that the high-peace country articles have significantly more words than the low-peace country ones.

Preprocess and results and store it to .csv

- **Optimizing the preprocess pipeline:**

- The preprocessing pipeline also has the same computation time problem as in the data loading pipeline. The two main contributors of this computation time are the americanize function and the SpaCy nlp NER process. While the NER function is written within the scipy package and there's nothing left we could do on our side, the americanize function has a large optimization potential. We optimized the loop as follows to make it 11x faster.

```
def americanize(string):
    '''
    Convert all British English spelling to American spelling in the input string
    source: https://stackoverflow.com/questions/42329766/python-nlp-british-english-vs-american-english
    -- Modified the code from source to meet our special requirement here.
    Input:
      sentence: a text string
    Return:
      string: a string with all words in English spelling
    '''
    for british_spelling, american_spelling in british_to_american_dict.items():
        string = re.sub(r'\b'+british_spelling+'\b', american_spelling, string, flags=re.IGNORECASE)

    return string

%timeit americanize(data_hp.content.values[0])
```

```
335 ms ± 2.29 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
british_pattern = '|'.join([r'\b'+word+r'\b' for word in british_to_american_dict.keys()])

def americanize(string):
    '''
    Convert all British English spelling to American spelling in the input string
    source: https://stackoverflow.com/questions/42329766/python-nlp-british-english-vs-american-english
    -- Modified the code from source to meet our special requirement here.
    Input:
      sentence: a text string
    Return:
      string: a string with all words in English spelling
    '''
    for british_word in re.findall(british_pattern, string, flags=re.IGNORECASE):
        string = re.sub(r'\b'+british_word+r'\b', british_to_american_dict[british_word], string, flags=re.IGNORECASE)

    return string

%timeit americanize(data_hp.content.values[0])
```

```
29.9 ms ± 22 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

- We also utilize the multi-processing to speed up the SpaCy nlp NER process by ~1.5x on a 2 CPUs basic notebook instance. But it still took 7 seconds to process 100 articles, and the processing time per article will increase as the article length increases. As we have more data in the samples, we will get more and more longer articles that need to be preprocessed (as shown in the previous boxplots). So we switched to the optimized notebook instance **(ml.c5.18xlarge**) which has 72 CPUs and got this preprocess done in 1 hour in total, which is an overall speedup of 19x.

```
start_time = time.time()
data_lp.content.apply(preprocess)
end_time = time.time()

end_time - start_time
```

```
11.004391431808472
```

```
# ! pip install pandarallel
from pandarallel import pandarallel

pandarallel.initialize()

start_time = time.time()
data_lp.content.parallel_apply(preprocess)
end_time = time.time()

end_time - start_time
```

```
INFO: Pandarallel will run on 2 workers.
INFO: Pandarallel will use Memory file system to transfer data between the main process and workers.

6.90094780921936
```

- The preprocessed data are now stored at the ***compress-data-sample*** bucket as **processed_train.csv** and **processed_test.csv**.

- For the convenience of data reading for the next step, we also saved a **shuffled** .json version of the processed data: **processed_train.json** and **processed_test.json**, where each line in the file is a json object like below:

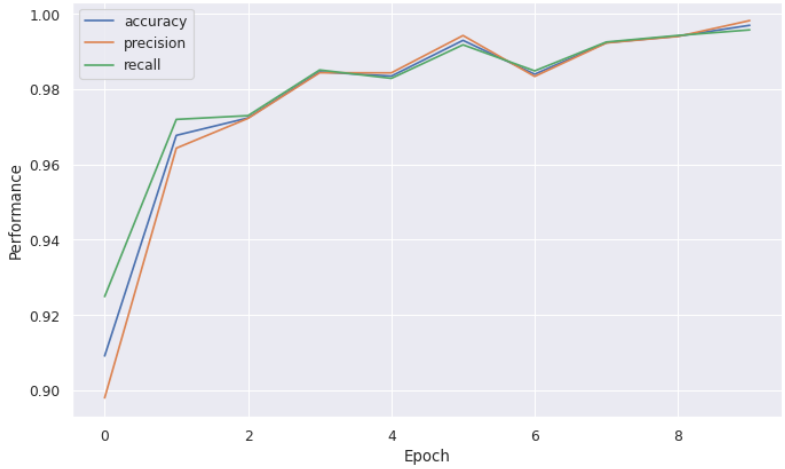
> {'title': xxxx,'content': xxxx,'wordCount': xxxx,'country':xxxx,'content_cleaned':xxxx}
> {'title': xxxx,'content': xxxx,'wordCount': xxxx,'country':xxxx,'content_cleaned':xxxx}
> ...

Fine-tune RoBERTa Model (logistic regression on embedding)

The entire sample dataset has

- 100K random data split in 8:2 training:validation ratio, trained for 1 epoch.

    - Entire process took ~30 minutes on ***ml.p3.2xlarge*** instance -> Training the entire 1M dataset takes 5 hours per epoch.
    - Performance on training:
        - Loss: 10.9%
        - Accuracy (threshold 0.5): 95.9%
        - Precision: 95.5%
        - Recall: 96.33%
    - Performance on Validation:
        - Loss: 6.1%
        - Accuracy (threshold 0.5): 97.6%
        - Precision: 98%
        - Recall: 97.9%
- 10K random data split in 8:2 training:validation ratio, trained for 10 epoch.
    - Slightly faster than the previous trial. (Maybe due to reusing data in cache). Took 21 minutes to finish training + evaluation.
    - Performance on training:
        - Loss: 1.17%
        - Accuracy (threshold 0.5): 99.7%
        - Precision: 99.8%
        - Recall: 99.6%



    - Performance on Validation:
        - Loss: 19.29%
        - Accuracy (threshold 0.5): 96.53%
        - Precision: 97.77 %
        - Recall: 95.25 %

- Conclusion: We prefer to use a larger dataset and train less epoch to reduce overfit and expose the model to more data patterns/variations.

Architecture Report

To Do: Write an overall description of
1) what AWS services to use (e.g. Sagemaker, EC2, ElasticSearch)

- 10 Sagemaker notebook ml.t2.medium instances
    - Two notebook instances for each member, one in east-1 and one in east-2
    - ~5GB EBS per notebook instances
- S3 (See part 4) for more details)

2) how many GPUS and if possible, running time would be needed, or how you are conceptualizing the resources you would need.
1 GPU for 200 hours.

3) what NLP software will be using (e.g. ROBERTA)
Roberta, SpaCy, NLTK, Tensorflow, HuggingFace (sagemaker.huggingface, transformers from huggingface)

4) how much total storage in S3 you will use
Right now we think we will use about 2TB amount of storage. Depending on further data of other countries, we may use more than that amount.

5) if classifiers (e.g. logistic regression, random forest, SVM), how many and which you will use.
Logistic regression, Random Forest, SVM, XGBoost

Next Step:

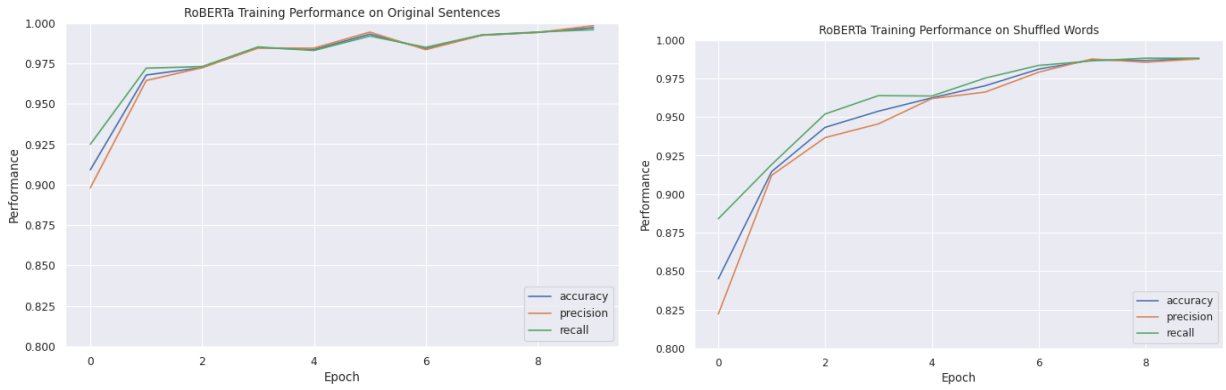
1. Shuffle the text, run BERT on this shuffled data
2. Try different model
3. Try to balance test set

Progress Report - Nov 10

Roberta Fine-tuning Result on Shuffled Sentence Comparison

Training mode	Dataset	Loss	Accuracy	Precision	Recall
100 K for 1 epoch	Train	10.9%	95.9%	95.55%	96.33%
	Validation	6.1%	97.6%	98%	97.2%
	Train (shuffled words)	19.96%	92.19%	91.11%	93.53%
	Validation (shuffled words)	16.85%	93.56%	97.64%	89.30%
10 K for 10 epochs	Train	1.17%	99.7%	99.8%	99.6%
	Validation	19.29%	96.53%	97.77%	95.25%
	Train (shuffled words)	4.62%	98.77%	98.76%	98.81%
	Validation (shuffled words)	34.99%	92%	95.41%	88.43%

10 epochs training performance comparison



- Performance improves slower than unshuffled
- Larger loss and significantly smaller recall comparing to unshuffled

Models' Performance on unshuffled data

- Run time for getting the embedding is a little long. We are still trying to solve this issue. So for now, we took a small subset of the training and testing data, constructed and tested the xgboost and svm on that subset to produce the performance stats (Accuracy) below. The precision and recall is not listed in the table, but can be found in the appendix below the table.

	RoBERTa (Logistic)	XGboost	SVM
Train (80K)	98.4%	99.1%	98.66%
Test (10K)	98.2%	98.2%	98.23%

Test (Minority Group Only, 1633 data points)	95.7%	95.9%	95.77%

Appendix: Detailed Performance Stats per model

xgb

Overall Train Classification Rate: 99.1%

Predicted Observed	NonPeace	Peace
NonPeace	99% (39558)	1% (357)
Peace	1% (359)	99% (39726)

Overall Test Classification Rate: 98.2%

Predicted Observed	NonPeace	Peace
NonPeace	98% (4995)	2% (83)
Peace	2% (95)	98% (4827)

Overall Test (Minority Group) Classification Rate: 95.9%

Predicted Observed	NonPeace	Peace
NonPeace	90% (420)	2% (20)
Peace	10% (47)	98% (1146)

SVM

Train:

	precision	recall	f1-score	support
NonPeace	0.9868	0.9864	0.9866	39915
Peace	0.9864	0.9868	0.9866	40085
accuracy			0.9866	80000
macro avg	0.9866	0.9866	0.9866	80000
weighted avg	0.9866	0.9866	0.9866	80000

Test:

	precision	recall	f1-score	support
NonPeace	0.9806	0.9846	0.9826	5078
Peace	0.9841	0.9799	0.9820	4922
accuracy			0.9823	10000
macro avg	0.9823	0.9823	0.9823	10000
weighted avg	0.9823	0.9823	0.9823	10000

Test (Minority Country):

	precision	recall	f1-score	support
NonPeace	0.8972	0.9523	0.9239	440
Peace	0.9820	0.9598	0.9708	1193
accuracy			0.9577	1633
macro avg	0.9396	0.9560	0.9473	1633
weighted avg	0.9591	0.9577	0.9581	1633

BERT (logistic):

Overall Train Classification Rate: 98.4%

Predicted Observed	NonPeace	Peace
NonPeace	98% (39335)	1% (580)
Peace	2% (705)	99% (39380)

Overall Test Classification Rate: 98.2%

Predicted Observed	NonPeace	Peace
NonPeace	98% (5001)	2% (77)
Peace	2% (108)	98% (4814)

Overall Test (Minority Group) Classification Rate: 95.7%

Predicted Observed	NonPeace	Peace
NonPeace	90% (419)	2% (21)
Peace	10% (49)	98% (1144)

Progress Report - Nov 17

LIME Explainer

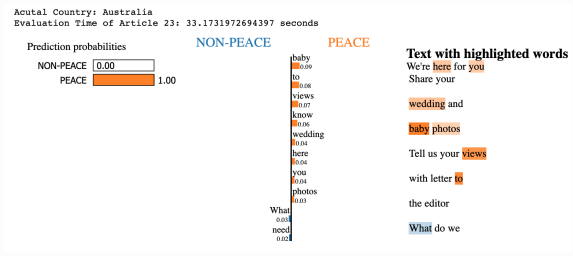
https://lime-ml.readthedocs.io/en/latest/lime.html#lime.lime_text.LimeTextExplainer.explain_instance

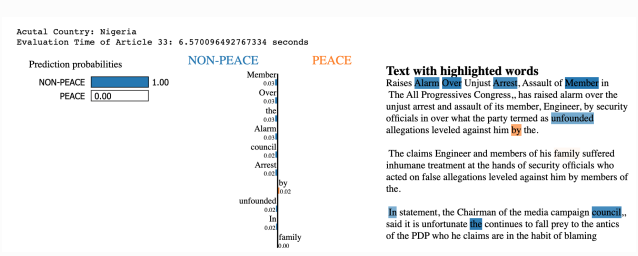
- **Algorithm:** *"First, we generate neighborhood data (25 for the sake of computation time) by randomly hiding features from the instance. We then learn locally weighted linear models on this neighborhood data to explain each of the classes in an interpretable way"*
- **Explain Option:** **"***if True (bag of words), will perturb input data by removing all occurrences of individual words or characters. Explanations will be in terms of these words. Otherwise, will explain in terms of word-positions, so that a word may be important the first time it appears and unimportant the second."*
- The important words for **unshuffled** are picked **based on the surrounding context,** rather than bag-of-words. So a word that's picked could not be interpreted on its own. We would have to read the entire sentence in order to evaluate its interpretation. On the other hand, the important words for **shuffled** data are picked by **bag-of-words** approach since the context is not important to us. We can see what key word the model is using
- **The model occasionally picks up some location/country names and makes predictions based on that, but it is also making predictions based on other vocabs or the context those vocabs are in.**

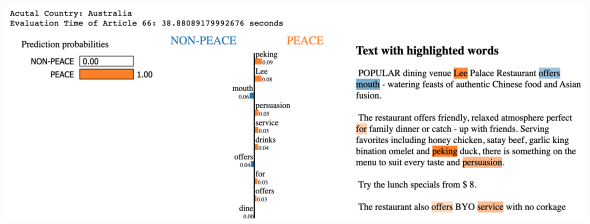
Some Interesting examples showing the some peace indicators model (RoBERTa trained on all countries) learned:

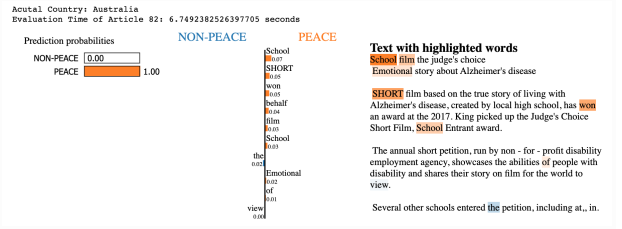
The model is learning some patterns that are related to the sentiment or topic of this article.

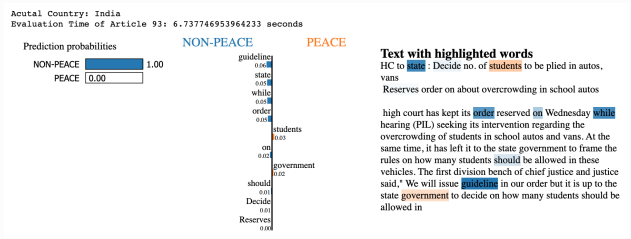
In Training:



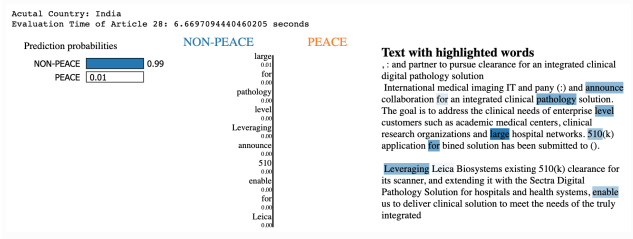


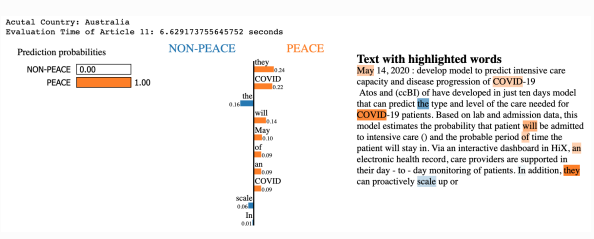


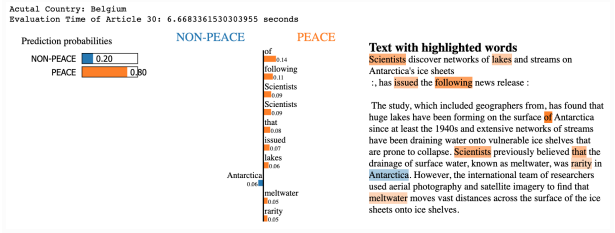




In Testing Data:







Train on minority country & Validate on India and Australia (reverse the order gives no huge performance change)

- 80K training, 20K validation for 2 epochs

	Loss	Accuracy	Precision	Recall	F1
Train	0.05	0.98	0.99	0.99	0.99
Validation	1.54	0.69	0.62	0.83	0.71

Same procedure on shuffled:

	Loss	Accuracy	Precision	Recall	F1
Train	0.17	0.94	0.94	0.98	0.95
Validation	0.88	0.45 (0.55)	0.45	1.0	0.62

Conclusion:

- If we split the training & validation by country, we can **minimize the effect of anything learned based on country names** in the performance on validation data.
- This new result shows that the **previous model's high performance could be a result from learning the country names that are not totally cleaned**.

Progress Report - Nov 24 & Dec 01

1. **Lime Paper:** https://arxiv.org/pdf/1602.04938.pdf (section 3.3: Sampling for Local Exploration)

2. Recall from last time that the performance drop in original vs. shuffled data when splitting by country. We found 2 causes for that within our processing pipeline
    a. Imbalance Training Data
    b. Overfit with 2 epochs

   which caused the model to focus more on positive data and produce a high recall and low accuracy we saw last week.

3. Train on articles from all countries but limit number of articles from Australia and India
   Distribution of countries:

```
Out[20]: {'Afghanistan': 103,
          'Zimbabwe': 134,
          'Uganda': 159,
          'Austria': 214,
          'Congo': 265,
          'Kenya': 271,
          'Czech Republic': 284,
          'Netherlands': 428,
          'Australia': 500,
          'India': 500,
          'Finland': 544,
          'Norway': 670,
          'Denmark': 704,
          'Sri Lanka': 706,
          'Iran': 1173,
          'Nigeria': 1326,
          'Sweden': 1925,
          'Belgium': 2656,
          'New Zealand': 4476}
```

   Output:

		Loss	Accuracy	Precision	Recall	F1
Original Sentence	Train	0.18	0.93	0.94	0.97	0.95
	Validation	0.11	0.96	0.96	0.98	0.97
Shuffled Sentence	Train	0.27	0.89	0.90	0.96	0.93
	Validation	0.19	0.93	0.94	0.96	0.95

4. Repeat 3) 10 times, each time the entire dataset contains the same set of 10K data from minority countries and a random set of Australian & Indian data, then perform a random 8:2 train-validation split.

		Loss	Accuracy	Precision	Recall	F1
Original Sentence	Train	0.2056 (0.1787, 0.2325)	0.9146 (0.9019, 0.9273)	0.9080 (0.8954, 0.9205)	0.9230 (0.9067, 0.9394)	0.9154 (0.9026, 0.9281)
	Validation	0.1239 (0.1133, 0.1346)	0.9535 (0.9494, 0.9577)	0.9620 (0.9534, 0.9707)	0.9440 (0.9297, 0.9585)	0.9527 (0.9485, 0.9570)
Shuffled Sentence	Train	0.3197 (0.3034, 0.3359)	0.8599 (0.8528, 0.8670)	0.8402 (0.8280, 0.8525)	0.8901 (0.8843, 0.8959)	0.8643 (0.8583, 0.8704)
	Validation	0.2120 (0.1970, 0.2271)	0.9145 (0.9083, 0.9207)	0.9136 (0.8915, 0.9357)	0.9165 (0.8902, 0.9429)	0.9139 (0.9079, 0.9200)

Here are the overall validation performance statistics when removing the two causes from 2) and including the results from 3). We also calculated the performance metrics for positive and negative classes separately, and the confidence interval for each metric.

(+: positive class; -: negative class)
(all training & validation is class balanced, only the Country Balanced Random Split is country balanced)

		Accuracy	Precision +	Recall +	F1 +	Precision -	Recall -	F1 -
Original Sentence	Random Split	94.68 (94.13, 95.22)	93.19 (91.70, 94.69)	96.48 (95.56, 97.40)	94.78 (94.31, 95.25)	96.39 (95.53, 97.25)	92.88 (91.10, 94.65)	94.57 (93.94, 95.20)
	Split By Country	71.88 (70.17, 73.58)	67.64 (66.49, 68.79)	84.17 (78.04, 90.29)	**74.77 (72.12, 77.43)**	80.02 (75.59, 84.45)	59.59 (55.42, 63.76)	**67.87 (66.04, 69.69)**
	Country Balanced Random Split	95.35 (94.94, 95.77)	96.20 (95.34, 97.07)	94.40 (92.97, 95.85)	95.27 (94.85, 95.70)	94.60 (93.19, 96.01)	96.30 (95.40, 97.21)	95.42 (95.00, 95.84)
Shuffled Sentence	Random Split	90.74 (90.05, 91.43)	90.25 (88.80, 91.70)	91.46 (89.34, 93.58)	90.80 (90.06, 91.53)	91.45 (89.60, 93.30)	90.02 (88.19, 91.84)	90.67 (89.97, 91.37)
	Split By Country	68.72 (67.16, 70.28)	64.77 (63.41, 66.14)	82.47 (77.04, 87.90)	**72.38 (70.35, 74.42)**	76.94 (71.86, 82.02)	54.97 (50.43, 59.51)	**63.57 (61.22, 65.93)**
	Country Balanced Random Split	91.45 (90.83, 92.07)	91.36 (89.15, 93.57)	91.65 (89.02, 94.29)	91.39 (90.79, 92.00)	92.00 (89.86, 94.13)	91.17 (88.20, 94.14)	91.46 (90.63, 92.29)

1. When the country in validation is in training, the model could perform well
2. F1 + is always higher than F1 - when split by country, which could mean that the model is learning peace data better when split by country.