THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

The Global Diffusion of Low Carbon Technologies

Report by
Deyu Kong
Rehapriadarsini Manikandasamy
The George Washington University

Author Note

This report was prepared for DATS 6501 - Data Science Capstone Project supervised by Professor Dr. Edwin Lo, George Washington University and Katherine Anne Stapleton, World Bank Group.

TABLE OF CONTENTS

INTRODUCTION

An increase in pace and scale of adoption of low-carbon technologies (LCTs) will help in accelerating the goal of net-zero emissions. According to the International Energy Association (2021), existing technologies should be able to achieve over 70 percent of the cut in global emissions which will help in reaching the net-zero by mid of the century. Recent academic literature has provided limited evidence on the diffusion of LCTs instead they are more focussed on their invention and innovation [1]. It is also widely discussed that low-carbon patenting growth has not accelerated sufficiently over the past decade. Hence it is important to analyze the diffusion of LCTs for the following reasons.

- To answer a critical policy question: how to bring about rapid enough progress in deployment of LCTs?
- Understanding diffusion is key to answering this question

This project is developed as a collaborative effort with the World Bank Group which was aiming to fill the gap on LCT diffusion by shedding light on the extent to which low carbon technologies have started to impact labor markets and boardroom discussions. The group first followed a growing literature that infers the spread of new technologies through their footprint in the demand for new tasks or skills in the text of job adverts [2]. As a result of this study, the World Bank Group has performed the following tasks in analyzing the job advertisements.

- Manually extracted keywords from the job titles by Disruptive tech group
- Expand list using synonyms, hypernyms and hyponyms for these words from WordNet, Dictionary.com and google trends and keep only ’bigrams’ [3]
- Results in 250 bigrams which classify into 51 technologies

Problem Statement:

Using natural language processing techniques to discover synonymous phrases and word combinations that have been missed during the research by the World Bank Group team. As part of implementation to find a solution to the given problem, natural language processing techniques like zero-shot classification and bigram counter generator were used throughout the project.

DESCRIPTION OF THE DATASET

To support the research on the commercial adoption of various low-carbon technologies (LCTs), the World Bank Group uses data scraped online based on job postings from Burning Glass Technologies (BGT). The dataset now includes the United States, in addition to the countries previously included (the UK, 29 European countries, Canada, Australia, New Zealand, and Singapore) and covers 51 Technologies. The online postings on a daily basis, sourced from > 40,000 online job boards and company websites from 2010 to 2022 (median 52 million posts per year) are used for the research.

The dataset is provided in the form of XML files zipped according to each year. These XML files contain job postings containing elements like jobID, job descriptions, date posted etc., which would later be reduced according to the usage in the analysis by picking up the required data elements.


```
<?xml version="1.0" encoding="iso-8859-1"?>
<Jobs>
<Job>
  <JobID>311017520</JobID>
  <CleanJobTitle>Pulmonologist/Critical Care Physician</CleanJobTitle>
  <JobDomain>www.resumes2work.com</JobDomain>
  <CanonCity>Missoula</CanonCity>
  <CanonCountry>USA</CanonCountry>
  <CanonState>MT</CanonState>
  <JobDate>2010-01-01</JobDate>
  <JobText>From: Company:  Providence Health &amp;amp; Services ( ) Job Reference ID: 21228810  Category: other  Duration:  City, ST:  Missoula, Montana  Country:  United Stat
ana. Our 247-bed tertiary center has been named a Top 100 Hospital by Thomson (formerly Solucient). We house the world-renowned International Heart Institute of Montana an
e-inspiring days, starry nights and fresh mountain air. Providence Health and Services, a not-for-profit network of hospitals, clinics and physician partners in Alaska, Ca
  <JobURL>http://www.resumes2work.com/job.php?id=35860314</JobURL>
  <PostingHTML></PostingHTML>
  <Source>Company from Job Board</Source>
  <JobReferenceID>21228810</JobReferenceID>
  <Email></Email>
  <CanonEmployer>Providence Health &amp;amp; Services</CanonEmployer>
  <Latitude>46.8575</Latitude>
  <Longitude>-114.042</Longitude>
  <CanonIntermediary></CanonIntermediary>
  <Telephone>604-806-9090</Telephone>
  <CanonJobTitle>Critical Care Physician</CanonJobTitle>
  <CanonCounty>Missoula</CanonCounty>
  <DivisionCode></DivisionCode>
  <MSA>33540: Metropolitan Statistical Area</MSA>
  <LMA>MT303354</LMA>
  <InternshipFlag>0</InternshipFlag>
  <ConsolidatedONET>29106900</ConsolidatedONET>
  <CanonCertification>
    <CanonCertification name="Board Certified/Board Eligible" type="Certification" />
  </CanonCertification>
  <CanonSkillClusters>Health Care: Emergency and Intensive Care;Specialized Skills|Specialized Skills|Specialized Skills</CanonSkillClusters>
  <CanonSkills>
    <CanonSkill name="Critical Care" clusterName="Health Care: Emergency and Intensive Care;Specialized Skills" />
    <CanonSkill name="Writing" clusterName="Specialized Skills" />
    <CanonSkill name="Allergy Consultation" clusterName="Specialized Skills" />
  </CanonSkills>
```


Fig 1: Snippet of data from one of the XML file

DESCRIPTION OF ALGORITHMS

Zero-shot Classification:

Zero-shot classification is a NLP technique which helps to map appropriate labels to a piece of text which is more similar to the topic modeling technique. This mapping is irrespective of the text domain and aspect. For example, it can be a topic, emotion or event described by the label [4]. Fig. 2 provided below shows how a zero-shot classifier can classify sentences based on the different label aspects. The output of a zero-shot classifier is a dictionary consisting of: Labels - the list of all the candidate labels used for prediction; Scores -the list of the probability scores corresponding to the labels; Sequence -the original sequences/text used for the prediction.

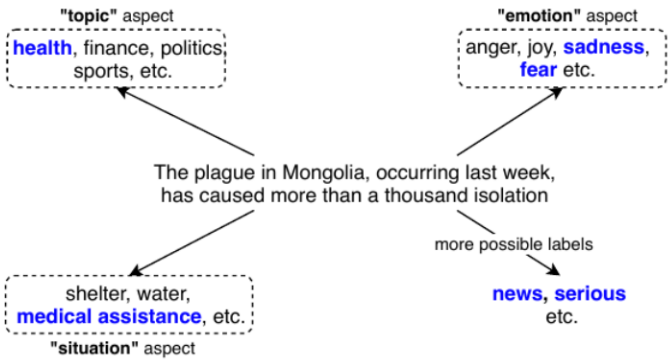

Fig 2: Zero-shot classification based on aspects [Image source: aixplain.com]

The zero-shot classification pipeline uses a pre-trained model on natural language inference to determine the relevancy amongst a set of given candidate names and sequence of text. This is one of the most convenient classifier models which works without the need for labeled training data. However, for a given sequence, the method requires each possible label to be fed through the large NLI model separately. Thus for N sequences and K classes, a total of N*K forward passes through the model are required. This requirement slows inference considerably, particularly as K grows [5].

Bigrams:

N-gram is a sequence of the N-words in the modeling of NLP. The N-gram modeling predicts most occurring words that can follow the sequences. The model is the probabilistic language model which is trained on the collection of the text [6]. Fig. 3 below shows how n-grams can be generated with the given sentence sequence. In the Bigram Language Model, we find bigrams, which are two words coming together in the corpus(the entire collection of words/sentences) [7]. The bigram model approximates the probability of a word given all the previous words by using only the conditional probability of one preceding word.

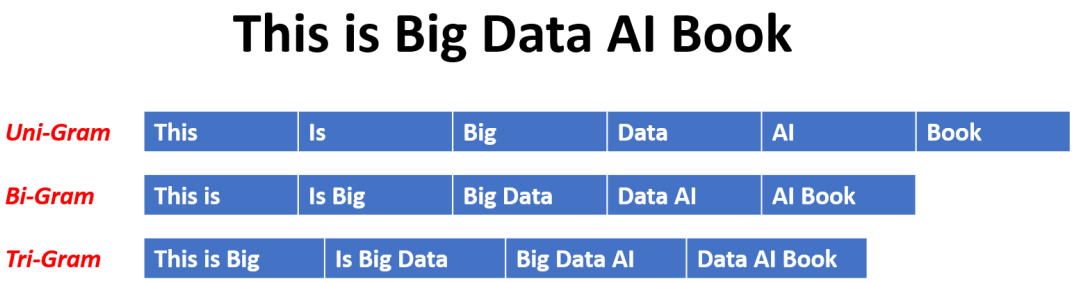

Fig 3: N-gram formation [Image source: Devopedia]

EXPERIMENTAL SETUP

AWS instance setup:

1. Log in to AWS account and launch console.
2. Launch virtual machine
3. Select AMI if already exists
4. GPU – g3.4xlarge
5. Generate a key pair and download the key in .pem version
6. Launch the instance

GCP instance setup:

1. Launch Google cloud console
2. Launch compute engine
3. Create a project and click create intents
4. Select the following options in the instance creation tab:
5. Zone – US central-a-zone
6. Series- N1
7. Machine type – n1-standard-8(30GB)
8. GPU – nvidia-teslaT4/P4/V100
9. Boot disk – ubuntu, version – 20.04, size – 300
10. Add the public key generated using Puttygen software

For windows,

1. Create a SSH session in mobaxterm
2. Host: External IP from any one of the instances
3. Name: ubuntu
4. Add the private key downloaded from the instances
5. Click ok.
6. Create a folder in the cloud

PyCharm IDE integration:

1. Create a project
2. Tools -> Deployment -> Configuration -> Add SFTP
3. SSH configuration: Host- IP address from instances, name: ubuntu, authorize using the private key pair.
4. Test the connection
5. In mappings, map the remote local path and the cloud folder's path
6. Deployment -> Configuration -> Automatic upload
7. Setup the Interpreter
8. Python interpreter -> SSH interpreter -> Existing server configuration -> Choose the cloud -> click ok.

PySpark on Azure Databricks:

PySpark is the python API for Apache spark which is an open source tool used for distributed computing consisting of libraries for large scale and real time big data processing. Since this project relied heavily on processing large volumes of textual data, the team employed PySpark on Azure Databricks platform to process and run NLP tasks like bigram counter generator using 'Spark-nlp' libraries by JohnSnowLabs.

Azure Databricks Setup

1. Login to portal.azure.com
2. Setup resource group
3. Launch Databricks Workspace
4. Compute menu item on the sidebar -> Create compute -> choose *11.3 LTS (includes Apache Spark 3.3.0, Scala 2.12)* Runtime -> choose appropriate worker type and number (For this project, the team used 2 to 8 *Standard_DS3_v2* machines with 14G memory and 4 Cores CPU) -> Set auto termination time -> Create Cluster
5. Compute menu item on the sidebar -> 'All' tab -> select cluster created -> 'libraries' tab -> Install new -> 'Maven' in Library Source tab -> put 'com.johnsnowlabs.nlp:spark-nlp_2.12:4.2.1' for installing Spark-nlp library -> Install
6. Put 'com.databricks:spark-xml_2.12:0.15.0' for Spark-xml library in the same location and install
7. Account name on top right -> Admin Console -> Workspace settings -> turn on DBFS File Browser
8. Account name on top right -> User Setting -> Git Integration -> setup your Github credentials
9. Repos menu item on the sidebar -> Add Repo -> use https URL to clone this project repo
10. Data menu item on the sidebar -> DBFS tab -> put all xml files in /data folder
11. Go to *spark_counter* in Repo -> project folder -> Script, Click on the square button on top right, choose the cluster created and Start.

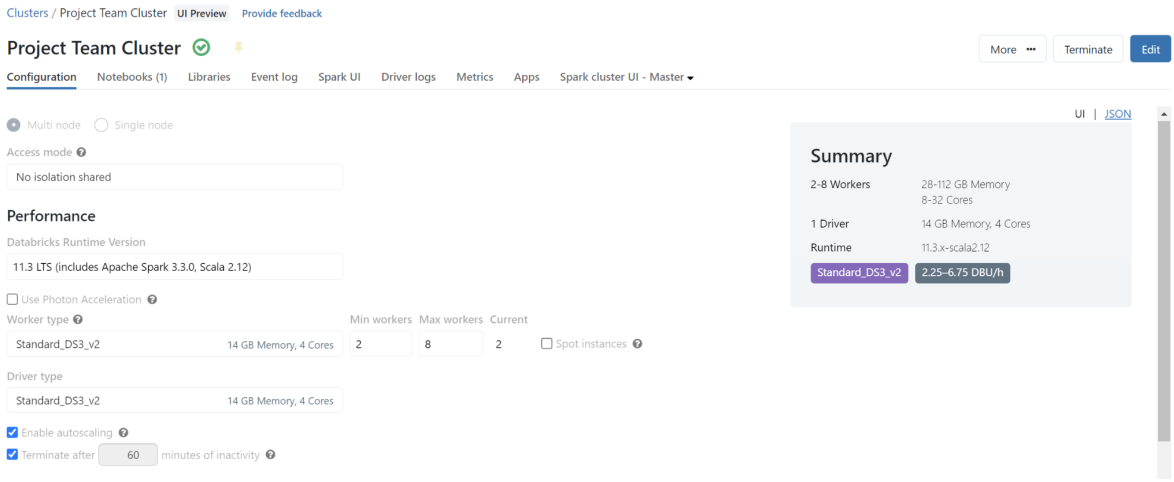

Fig 4: Cluster created on Azure Databricks

Zero-Shot Classification:

The zero-shot classification model is implemented using the transformer model pipeline generated by Hugging Face.

```
with open(f'{OR_PATH}/candidates.txt', 'r') as f:
    cand_txt = f.read()
candidates = cand_txt.split('\n')
# %%
classifier = pipeline("zero-shot-classification")


print(classifier(text, candidate_labels=candidates,))
```

Fig 5: Code snippet for zero-shot implementation

The pipeline can use any model trained on an NLI task, by default bart-large-mnli. It works by posing each candidate label as a "hypothesis" and the sequence which we want to classify as the "premise" [8]. A list of candidate names focussing on LCTs are generated to be fed into the model to generate the probability scores (to find the best match of low-carbon technology related content) on the given text sequence.

```
biodegradable and compostable materials
greenhouse gas capture
combustion technologies
efficient electrical power generation
emissions reductions in production
end-user energy management
energy efficient solutions
fuels of non-fossil origin
improved thermal performance
new energy vehicles
nuclear
renewable energy
smart grids
waste management
```

Fig 6: List of labels used

The zero-shot classifier works in such a way that it processes the given job description and creates a text sequence. The text sequence is then fed to the pipeline and compared with a given list of labels to find the probability of presence of aspect/context of the given labels. The probability score generated for each label will help in understanding the presence or absence of LCTs related in the given job description. This will be helpful to answer our problem statement of finding the spread of low-carbon technology adoptions for the given companies.

Bigram Counter:

After starting the PySpark cluster, the data needs to be read through DBFS. Considering the computational complexity of the data, only a limited number of XML files are used to generate the bigram keywords for the project. The data is processed into a data frame consisting of all the elements from the XML file initially and later the columns with job descriptions are segregated for further analysis.

```
df_raw.printSchema()
```

```
root
 |-- BGTOcc: string (nullable = true)
 |-- BGTSubOcc: string (nullable = true)
 |-- CIPCode: string (nullable = true)
 |-- CanonCertification: struct (nullable = true)
 |    |-- CanonCertification: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- _VALUE: string (nullable = true)
 |    |    |    |-- _name: string (nullable = true)
 |    |    |    |-- _type: string (nullable = true)
 |-- CanonCity: string (nullable = true)
 |-- CanonCountry: string (nullable = true)
 |-- CanonCounty: string (nullable = true)
 |-- CanonEmployer: struct (nullable = true)
 |    |-- _StockTicker: string (nullable = true)
 |    |-- _VALUE: string (nullable = true)
 |-- CanonIntermediary: string (nullable = true)
 |-- CanonJobHours: string (nullable = true)
 |-- CanonJobTitle: string (nullable = true)
 |-- CanonJobType: string (nullable = true)
 |-- CanonMaximumDegree: string (nullable = true)
 |-- CanonMinimumDegree: string (nullable = true)
 |-- CanonOtherDegrees: string (nullable = true)
 |-- CanonPostalCode: string (nullable = true)
 |-- CanonPreferredDegrees: string (nullable = true)
 |-- CanonRequiredDegrees: string (nullable = true)
 |-- CanonSkillClusters: string (nullable = true)
 |-- CanonSkills: struct (nullable = true)
```

Fig 7: Snippet of data frame's schema

Load the following libraries to start building the pipeline for bigram counter.

```
import pyspark.sql.functions as f
from sparknlp.pretrained import PretrainedPipeline
import sparknlp
from sparknlp.base import *
from sparknlp.annotator import *
```

Fig 8: Libraries used

The bigram counter is generated by creating a DocumentAssembler() which takes in the job description as the input. The input is then processed to find the tokens/keywords in the given text sequence. These tokens are used to find the words coming together based on the conditional probability amongst each other.

```
document_assembler = DocumentAssembler() \
    .setInputCol("JobText")

tokenizer = Tokenizer() \
    .setInputCols(["document"]) \
    .setOutputCol("token")

bigrams = NGramGenerator() \
            .setInputCols(["token"]) \
            .setOutputCol("bigrams") \
            .setN(2)

pipeline = Pipeline(stages=[
    document_assembler,
    tokenizer,
    bigrams,
])
```

Fig 9: Bigram counter pipeline

The following is the flowchart representation of steps involved in the generation of bigrams.

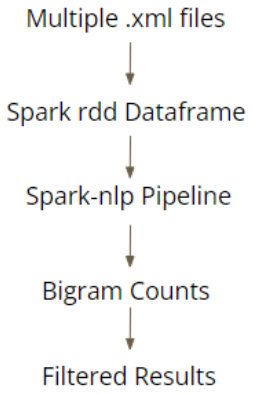

Fig 10: Steps involved in generating bigrams

RESULTS

Zero-shot classification:

	labels	scores
0		0.331111
1	improved thermal performance	0.098071
2	energy efficient solutions	0.088838
3	combustion technologies	0.084017
4	efficient electrical power generation	0.056562
5	end-user energy management	0.048045
6	fuels of non-fossil origin	0.045543
7	renewable energy	0.044551
8	emissions reductions in production	0.039654
9	smart grids	0.036689
10	greenhouse gas capture	0.034671
11	biodegradable and compostable materials	0.032794
12	new energy vehicles	0.030038
13	nuclear	0.014877
14	waste management	0.014538

Fig 11: Probability scores for labels

The job description processed under the zero-shot classifier generated the probability scores for each label used. Analyzing the result generated above, the probability scores for labels listed from 1 to 14 have scores less than 0.1 which shows the processed job description doesn't have context related to the given labels and Label 0 denotes presence of no technology which has a high probability of 0.33. In this way, each job description could be processed to find which label has more probability that shows the presence/absence of relevant content (related to LCTs) in the text.

Though the zero-shot classification is the most advanced technique it takes more time to process each text. One of the primary limitations of using this technique is it takes too long to inference. While processing the text on a local machine it takes 3 to 4 minutes long to generate probabilities while with a powerful Nvidia V100 GPU on a GCP instance it takes 35 seconds to generate the scores for one job. As the number of candidate labels and amount of job text increases it would take more time to generate the result. Hence, the team decided to move to traditional NLP methods like bigram counter to find the keyword combinations and synonyms.

Bigram counter:

After loading xml files for the first two weeks in 2010, namely *US_XML_AddFeed_20100101_20100107.xml* and *US_XML_AddFeed_20100108_20100114.xml* in which contains more than 103,000 entries of job advertisement, it took 24 minutes on the cluster to construct bigram counter dictionary.

```
+-----------+------+
|     bigram| count|
+-----------+------+
|   this job|204512|
| job poster|161537|
|   . Please|141359|
|        > >|122641|
|       , or|119409|
|      , and|111425|
|   , please|105321|
| Location :| 89634|
|       to :| 89340|
| about this| 86973|
| services ,| 85372|
|     do not| 84491|
|     Date :| 83978|
|       to a| 83409|
|        � �| 82456|
|  Please do| 81654|
|     or any| 81382|
| , products| 81372|
```

Fig 12: Bigram count dictionary

The generated bigrams are filtered using the threshold and core keyword setup to find the synonymous bigrams.

```
1  re_approx.filter(f.col("bigram").contains('energy')).show(50, truncate=False)
```

▸ (2) Spark Jobs

```
|energy management       |66   |
|with energy             |61   |
|energy conservation     |51   |
|energy efficient        |45   |
|energy environment      |45   |
|high-energy personality |39   |
|positive energy         |35   |
|High-energy and         |35   |
|your energy             |33   |
|high-energy environment |33   |
|and high-energy         |30   |
|high-energy team        |30   |
|for high-energy         |28   |
|in energy               |28   |
|energy industry         |28   |
|hard-working High-energy|27   |
|energy individual       |27   |
|solar energy            |26   |
|energy-smart ,          |24   |
|our energy-smart        |24   |
|energy savings          |24   |
```

Fig 13: Bigrams generated for the keyword

To test code pipeline as a proof of concept, the team used ‘renewable energy’ bigram from the existing keyword list with threshold of **30%**, **50%** and **70%** and ‘core’ keyword of ‘**energy**’ as hyperparameters. The results are as follows:

Threshold	**30%**	**50%**	**70%**
Bigrams Filtered-in	8	14	30
New Bigrams Found	energy management(66)	energy management(66) energy conservation(51) energy efficient(45)	energy efficiency(112) energy management(66) energy conservation(51) energy efficient(45) energy-smart ,(24) energy savings(24)

Table 1: New Bigrams Found Using Different Threshold

Obviously, using a wider threshold helps identify more potentially relevant bigrams currently unaccounted for, but brings in more ‘noise’ to the fold in the process. Meanwhile choosing the ‘core’ keyword to help the filtering process is more of an art than a science. The general rule-of-thumb is using the noun in the existing bigram and losing the adjective, or using one more relevant to that concept. The hyperparameter and the eyeballing process after filtering requires significant human attention and could not yet be automated. But this shows the code pipeline works as intended, and helped identify bigrams not yet included in the keyword search list.

SUMMARY AND CONCLUSION

As a result of the research, new bigrams were generated for the keywords associated with the LCTs. Implementing NLP techniques helped in reducing the manual efforts in finding word phrases and synonyms for the given problem statement. Though advanced NLP techniques were used to find the best match of keywords the project has its own limitations based on the following.

- Processing Power: The methods required high computational power to process the text documents. For example, it took 24 mins for two XML files in PySpark.
- Hyperparameters: More research needs to be involved in finding the accurate threshold & “core” keyword to find the best combination of words.
- Punctuation removal / Lower-case could increase the chance of model performance
- Manual Screening: Selecting meaningful ones form filtered results

Future Scope:

- Loop through existing bigrams and append new ones automatically
- Usage of Cosine/Vector Similarity between tokens/bigrams
- Implement Zero-shot classification pipeline in spark-nlp
- Beefier GPU and more budget on adapting technologies

REFERENCES

1. Popp, David. 2002. “Induced Innovation and Energy Prices.” THE AMERICAN ECONOMIC REVIEW,92(1): 60
2. Acemoglu, Daron, David Autor, Jonathon Hazell, and Pascual Restrepo. 2022. “Artificial Intelligence and Jobs: Evidence from Online Vacancies.” Journal of Labor Economics, 40(S1): S293–S340.
3. Bloom, Nicholas, Tarek Alexander Hassan, Aakash Kalyani, Josh Lerner, and Ahmed Tahoun. 2021. “The Diffusion of Disruptive Technologies.” National Bureau of Economic Research Working Paper 28999.
4. https://www.section.io/engineering-education/how-to-implement-zero-shot-classification-using-python/
5. https://github.com/huggingface/transformers/blob/main/examples/research_projects/zero-shot-distillation/README.md
6. https://www.kdnuggets.com/2022/06/ngram-language-modeling-natural-language-processing.html
7. https://www.educative.io/answers/what-is-a-bigram-language-model
8. https://discuss.huggingface.co/t/new-pipeline-for-zero-shot-text-classification/681
9. Gary King, Patrick Lam, and Margaret Roberts. 2017. “Computer-Assisted Keyword and Document Set Discovery from Unstructured Text.” American Journal of Political Science, 61, 4, Pp. 971-988.

APPENDIX

Github Repo Link:

All the project codes and documents related to the project are available in the following github repo.

https://github.com/herrzilinski/Capstone2022