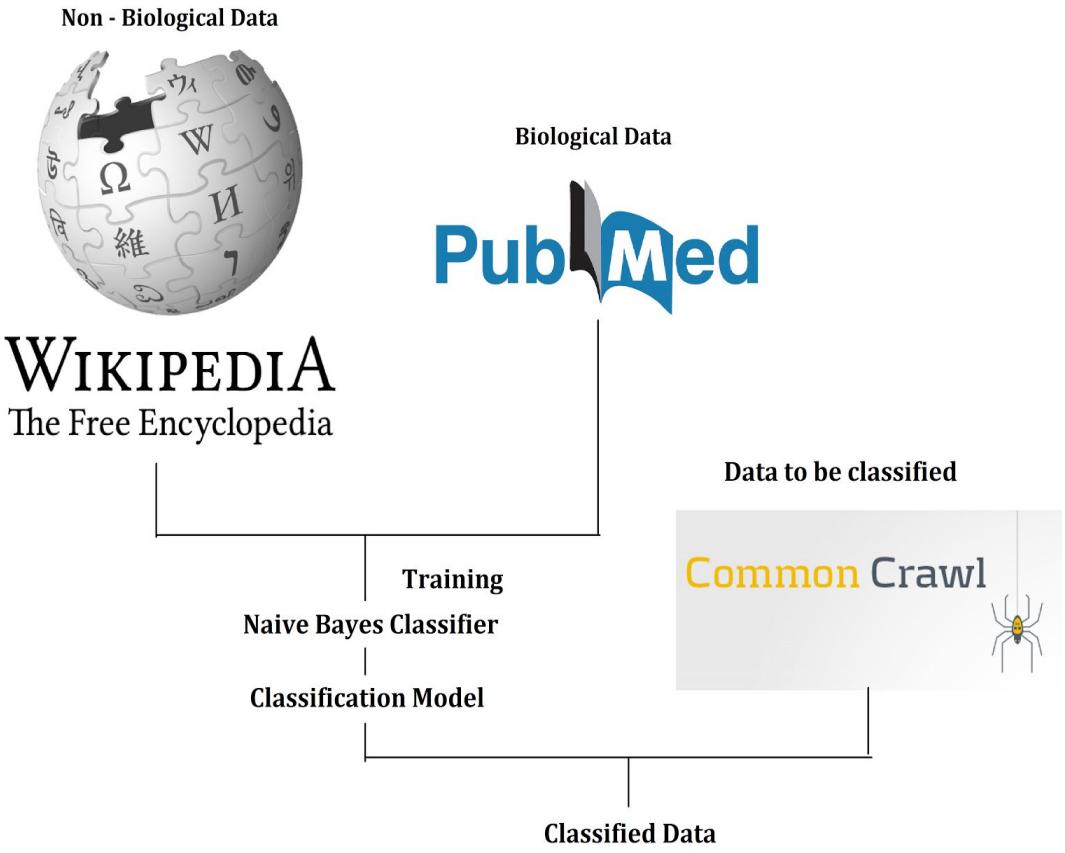
Big Data Analytics

Report on Classification of Biological Data in Common Crawl

1. Introduction

The Common Crawl is an **open repository of web crawl data** collected over last 7 years which contains several petabytes of data. It contains web page data, meta data extracts and text extracts. Academia and research constantly requires annotated or classified data which helps researchers carry out their experiments using the data. We aim to make this task easy for Biologists and anyone dealing with biological corpuses. This project aims to separate **Biological data** from other types which are not related to biology. It further aims at **classifying the Biological data into different classes** such as: human microbiome, carcinomic, infectious diseases, genetic diseases, etc. The topics in biology are quite diverse and thus the documents may fall into one or more classes. Thus the classification is not disjoint. The resulting classified data pool will serve as a central repository of all types of Biological data in an organized format.

1.1. Architecture



We use the Naive Bayes base learner as our classification model. The training data for this classifier is the English articles from Wikipedia for Non - Biological class and PubMed articles for Biological class. We first process archived common crawl files in parallel by doing the following: unpack, process, delete unpacked version. We then Classify Common Crawl data based on the classification model.
Once we have extracted all the Biology related data from Common Crawl it is now ready for further classification based on the categories in PubMed data. This classification of data into different biological categories can be achieved by using parallel classification pipelines.

1.2. Pipeline

The text that was extracted from the Wikipedia and PubMed articles is further processed to strip the punctuation marks and any insignificant characters. It is then Tokenized based on the white space characters such as spaces, tabs, new lines etc. We have also used N-gram features. We then apply the Countvectorizer module to get term and/or feature frequencies. Our initial implementation also included Inverse Document Frequency, but as directed, we have removed the IDF module. Finally, we train the NaiveBayes classifier to obtain the model. Hence, the resulting Pipeline is shown below:

Preprocessed Articles → Punctuation Stripper → Tokenizer → N-Gram Features → CountVectorizer → NaiveBayes

1.2.1 Classification of Non-biological data from Biological data

For this part, we have used unigram features instead of bigram or trigram. Unigram features give good performance for large data sets and using bigram, trigram or four-gram for the initial classification would have been computationally expensive. There was no significant difference in the accuracy of classification between unigram and other N - gram feature extraction models when used in combination with Naive Bayes on small datasets when we initially tested our strategy. Hence, we stuck with unigram features for this task. All the other pipeline elements remain the same.

1.2.2. Classification of Biological Data into other Bio related categories

For this part, we have used the trigram and four-gram features as the datasets under consideration for training were small and it gave a better performance and accuracy results over test data. All the other pipeline elements remain the same. We have further classified the biological data into Microbial and Cancer. The microbiome subset of data was too small, so we started off with microbe data.

1.3. Technologies

The following technologies have been used:

- AWS s3 for storage
- Globus for file transfer
- Apache Spark for big data processing
- PySpark for ML pipelines
- We would like to use MongoDB to store the results. For now we stick with AWS s3.

2. Running the Code and Experiments

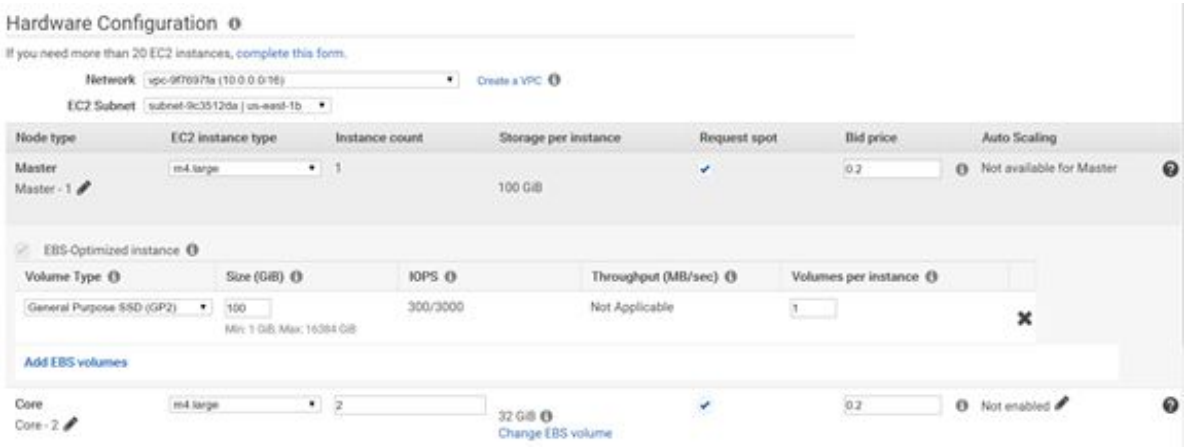
2.1. Create a cluster by using AWS EMR

1. Select "go to advanced options"

2. Select latest Release.
   Select "Spark" and "Hadoop"
3. Enter configuration below -
   [{"classification":"spark","properties":{"maximizeResourceAllocation":"true"}}]
   This configuration will set the spark configuration file automatically, to fully utilize all resources while running spark job.

4. In Hardware Configuration, select suited instance type for Master and Core.

When testing or running demo, create a small cluster. We have used “m4.large” for testing, one for master, 2 for core. Use “memory optimized” instance for real run. Please select “add EBS volumes” if you need more space to store data. Also select “Request spot” to save money.



5. Full cluster name, tags and key pair. Done.

2.2. Separating Biological Articles from Non-Bio (bioExtractor)

2.2.1 Dataset

All Pubmed and Wikipedia data has been uploaded to S3(s3://project-snow-crash). All uncompressed Pubmed data is 27GB, and we have selected 12GB of pubmed which is the same size as English wikipedia data. This has been done to get a balanced dataset. The data has been split into training and test datasets by taking an 80% - 20% split.

Data	Path in S3	source	Size
Pubmed test data (20%)	s3://project-snow-crash/pubmed-test	from pubmed full dataset	100M
Wikipedia test data (20%)	s3://project-snow-crash/wiki-test	From wikipedia full dataset	100M
Pubmed training data (A-F) for biology (80%)	s3://project-snow-crash/pubmed-AF	comm_use.*.txt.tar.gz	12G
All training English data from wikipedia for non - biology (80%)	s3://project-snow-crash/enwiki	We have used the English data from lorelei project *enwiki-latest-pages-articles-multistream.xml extracted by* WikiExtractor.py	11.1G

2.2.2. Run test on PC.

We have installed Spark on the PC (windows 10, i7-3630QM, 8GB RAM). Spark combined with jupyter notebook can give an interactive way for testing and demo.

1. create one separate anaconda python environment for spark.
   conda create -n spark python=3.5 anaconda
2. Get into this environment
   activate spark
3. Set environment variable for pyspark
   PYSPARK_DRIVER_PYTHON : jupyter
   PYSPARK_DRIVER_PYTHON_OPTS : notebook
4. Run pyspark, this will start jupyter notebook.
   pyspark
5. Open bioExtractor.ipynb and run.

(P.T.O.)

2.2.3. Use part of the full data set for testing.

1. Upload test data from S3 to HDFS

    hadoop distcp s3n://project-snow-crash/wiki-test/* hdfs://ec2-54-209-223-251.compute-1.amazonaws.com:8020/user/hadoop/wiki-test/
    hadoop distcp s3n://project-snow-crash/pubmed-test/* hdfs://ec2-54-209-223-251.compute-1.amazonaws.com:8020/user/hadoop/pubmed-test/

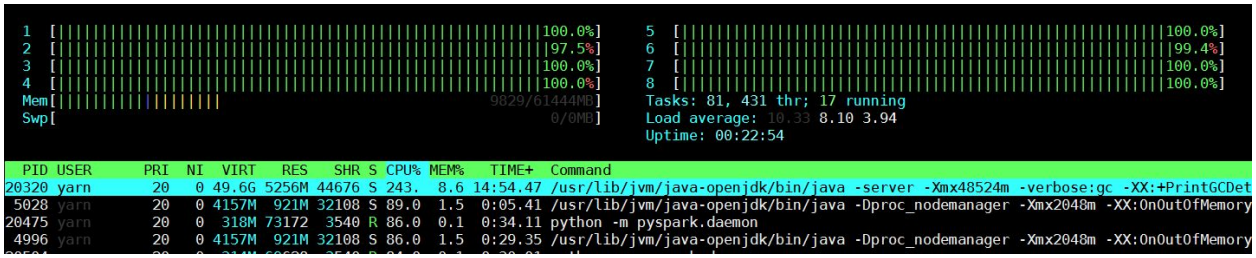
2. Submit spark program, it will print the error rate, confusion matrix and a sample test.

    spark-submit bioClassify.py

2.2.4. Run on full data set

1. Upload test data from S3 to HDFS, Notice that it's really slow to upload pubmed dataset to HDFS, We think that's because there are too many small files. Combine some files before upload maybe a good choice.

We use this script to combine small files. These also influence the CPU utilization. After we combine all the files the CPU utilization increase from 50% to 100%.



combineFiles.sh

```
#!/bin/bash
for FOLDER in $(ls /mnt/pubmed-AF) ; do
        echo $FOLDER
        cat /mnt/pubmed-AF/$FOLDER/*.txt >>
/mnt/pubmed-AF-combine/$FOLDER.txt
done
```

Upload commands

    hadoop distcp s3n://project-snow-crash/enwiki /* hdfs://ec2-54-209-223-251.compute-1.amazonaws.com:8020/user/hadoop/enwiki/
    hadoop distcp s3n://project-snow-crash/pubmed-AF-combine /* hdfs://ec2-54-209-223-251.compute-1.amazonaws.com:8020/user/hadoop/pubmed-AF-combine/

For large dataset, stage result are much large, which can exceed the default spark.driver.maxResultSize(1g). We set it to 0, which means unlimited. config("spark.driver.maxResultSize", "0")

2. Submit spark program, it will print the error rate, confusion matrix and a sample test. Change the data path in code before submit.

    spark-submit bioClassify.py

2.2.5. Test result for bioExtractor:

	Small test dataset	Full dataset
Accuracy	Test Error : 0.0306813399814	Test Error : 0.0788781354741
Run time	Less than 10 minutes on: master(m4.large), 2 core(m4.xlarge)	Less than 20 minutes on: master(r3.2xlarge), 10 core(r3.2xlarge)

Confusion matrix for small test dataset

	bio	other	total
bio	86193	2100	88293
other	2712	65833	68545

Confusion matrix for full dataset

	bio	other	total
bio	9402063	650493	7659957
other	746637	6913320	10052556

When the amount of data for training and testing was increased, the error increased from 3% (small test set) to 7.8% on the full data set. This is likely due to wikipedia articles containing some documents related to biology.

2.3. Microbe-Cancer Classifier

2.3.1. Dataset

The bash script to extract all cancer and microbe data from PubMed full dataset is findMicroCancer.sh:

```
#!/bin/bash
for FILE in $(find /mnt/pubmed-all -iname "*cancer*") ; do
   cp $FILE /mnt/pubmed-sub/cancer
done

for FILE in $(find /mnt/pubmed-all -iname "*microbiol*") ; do
   cp $FILE /mnt/pubmed-sub/microbiol
done
```

Another bash script to extract data as “other”, which is nearly same in size as cancer and microbes:

```
#!/bin/bash
for FOLDER in $(ls /mnt/pubmed-all) ; do
   #echo $FOLDER
   if echo $FOLDER | grep -iq cancer; then
      echo "find cancer"
      continue
   fi
   if echo $FOLDER | grep -iq microbiol; then
      echo "find microbiol"
      continue
   fi
   COUNT=0
   for FILE in $(ls /mnt/pubmed-all/$FOLDER) ; do
      COUNT=$((COUNT+1))
      #echo /mnt/pubmed-all/$FOLDER/$FILE
      cp /mnt/pubmed-all/$FOLDER/$FILE /mnt/pubmed-sub/other
      if [ "$COUNT" -gt 4 ]; then
         #echo $COUNT
         break
      fi
   done
done
```

We also combined all the files.

```
cat /mnt/cancer/*.txt >> /mnt/cancer.txt
cat /mnt/microbiol/*.txt >> /mnt/microbiol.txt
cat /mnt/other/*.txt >> /mnt/other.txt
```

Data	Path in S3	source	Size
small cancer data	s3://project-snow-crash/pubmed-sub-small/cancer-small	part of full dataset	141KB
small microbial data	s3://project-snow-crash/pubmed-sub-small/microbiol-small	part of full dataset	148KB
small other data	s3://project-snow-crash/pubmed-sub-small/other-small	part of full dataset	185KB

Cancer data	s3://project-snow-crash/ pubmed-sub/cancer	Use findMicroCancer.sh extract from full dataset(pubmed-all)	698M
microbial data	s3://project-snow-crash/ pubmed-sub/microbiol	Use findMicroCancer.sh extract from full dataset(pubmed-all)	552M
Other medical data	s3://project-snow-crash/ pubmed-sub/other	Use findOther.sh extract from full dataset(pubmed-all)	690M

2.3.2. Run

Same as above, here ngramMicroCancer.ipynb can be used for testing on PC. Use same command to upload the data from S3 to HDFS, and submit the python code.

2.3.3. Test result for Microbe-Cancer classifier:

	Small test dataset	Full dataset
Accuracy	Test Error : 0.230483271375	Test Error : 0.412268087653
Run time	Less than 10 minute on: master(m4.large), 2 core(m4.xlarge)	Less than 20 minutes on: master(r3.2xlarge), 10 core(r3.2xlarge)

Confusion matrix for small test dataset

	microbial	cancer	other	total
microbial	69	3	47	119
cancer	0	63	64	127
other	6	4	282	292

Confusion matrix for full dataset

	microbial	cancer	other	total
microbial	222007	16391	282559	520957
cancer	17743	256881	282782	557406
other	47922	42102	504066	594090

When the amount of data for training and testing was increased, the error increased from 23% (small test set) to 41% on the full data set.