# Covid-19 classification approach

**Authors:** Bita Faraji; Sindhura Vemuri; Yash Choksi

**Instructor:** Jongwook Woo

**Date: 05/11/2020**

# Lab Tutorial

Bita Faraji(bfaraji@calstatela.edu)
Sindhura Vemuri(svemuri2@calstatela.edu)
Yash Choksi (ychoksi@calstatela.edu)
05/11/2020

# Covid-19 research paper classification

## Objectives

In this hands-on lab, you will learn how to:

- Train NLP system
- Handling with JSON files in python
- Data cleaning
- Feature extraction
- Visualization

## Platform Spec

- Oracle BDCE
  - # of nodes: 3
  - Total Memory Size: 720GB
  - Python version 2.7
- Azure ML
  - Free Workspace
  - 10 GB storage
  - Single Node
- Data Bricks
  - Data Bricks subscription

- 6 GB Memory
- 0.88 Cores
- 1 DBU
- Python version 3

In this project we created from data collection to model statistics everything with python and different machine learning libraries. Machine learning libraries which we used is mainly Sci-kit Learn library.

# Goal:

In the time of such pandemic we want to thank you to every person who is working as health worker in any capacity. Due to ongoing research work going for vaccine preparation there is always need for review of older research papers. And in that case, we want to make sure we are using our time most efficiently. By keeping that as reference, we are trying to build a classification machine learning model which will predict which types of papers are there and how each paper is distinguished. Therefore, at the end of this project when any new paper is feed into our model; it can predict the category of that paper. This is classical unsupervised machine learning problem where we did not have target variables.

# Data Collection:

Data is collected from Kaggle.com and this is open research dataset. Dataset comprised of over 8 GB with many sparse data collections. All the files in data is in JSON format. Each json file is having defined JSON object with different attributes. The dataset comprised of more than 55,000 different research papers and subject of research papers is not only covid-19.
The main repository for this project is: https://github.com/yashchks87/covid_19_nlp
If you download CLI_Code.py you can execute it with Pyspark using this command
spark-submit --driver-cores 20  --driver-memory 20g CLI_code.py

And if you download file CIS5560projectFinal.html then you can get the databricks file.

# Data Cleaning:

Data cleaning is one of the most important and time-consuming part in this project because data was highly unstructured and to make it workable in python data should be converted to some specific type of dataframe or some other data structure so we can process it in python. This is glimpse of data cleaning which we did:

```python
import json
class FileReader:
    def __init__(self, file_path):
        with open(file_path) as file:
            content = json.load(file)
            self.paper_id = content['paper_id']
            self.abstract = []
            self.body_text = []
            # Abstract
            for entry in content['abstract']:
                self.abstract.append(entry['text'])
            # Body text
            for entry in content['body_text']:
                self.body_text.append(entry['text'])
            self.abstract = '\n'.join(self.abstract)
            self.body_text = '\n'.join(self.body_text)
    def __repr__(self):
        return f'{self.paper_id}: {self.abstract[:200]}... {self.body_text[:200]}...'
first_row = FileReader(all_json[0])
print(first_row)
```

Sample of data in JSON:


```json
{
    "paper_id": "0a2a28cb82e7a03af0a9fad4fd4c68c9fdac2477",
    "metadata": {
        "title": "Prediction and Evolution of B Cell Epitopes of Surface Protein in SARS-CoV-2",
        "authors": [
            {
                "first": "Jerome",
                "middle": [
                    "R"
                ],
                "last": "Lon",
                "suffix": "",
                "affiliation": {
                    "laboratory": "",
                    "institution": "South China University of Technology",
                    "location": {
                        "postCode": "510006",
                        "settlement": "Guangzhou",
                        "country": "China"
                    }
                },
                "email": ""
            },
            {
                "first": "Yunmeng",
                "middle": [],
                "last": "Bai",
                "suffix": "",
                "affiliation": {
                    "laboratory": "",
                    "institution": "South China University of Technology",
                    "location": {
                        "postCode": "510006",
                        "settlement": "Guangzhou",
                        "country": "China"
                    }
                },
                "email": ""
            },
            {
```

This is the structure of our data and as you can see, we have to make this data to go inside every different column of our data frame. After clearing data, the data set look like:

| | paper_id | doi | abstract | body_text | authors | title |
|---|---|---|---|---|---|---|
| **0** | 71f45bcdac8e83e02ee1d8b5eab8ce5c425f5cce | 10.3390/molecules20034610 | Identifying molecular targets for eliciting br... | Developing broad-range virus-neutralizing anti... | Wang, Denong. Tang, Jin. Tang, Jiulai. Wan... | Targeti Cryptic for... |
| **1** | 02b8dea56378d11fe92d6a60ff37a34b0d6ea63e | 10.1186/ar1899 | Macrophage-like synoviocytes and fibroblast-li... | Rheumatoid arthritis (RA) is characterized by ... | Zhu, Ping. Lu, Ning. Shi, Zhan-guo. Zhou, ... | CD147 on syn in<br> |
| **2** | 2718d50101d585d27c5c817dfe2fb1dded056e1a | 10.3390/v11100958 | Feline calicivirus (FCV) can cause painful ora... | Feline calicivirus (FCV) is a common viral pat... | Spiri, Andrea Monika. Meli, Marina Luisa. <b... | Enviroı Contar Hygier |
| **3** | 1d75590848a42f84359d8ff9631fdc3145efd631 | 10.3390/ijms20071580 | The eIF4F complex is a translation initiation ... | Rotavirus is considered to be one of leading c... | Chen, Sunrui. Feng, Cui. Fang, Yan. Zhou,<... | The Eu Transla Factor |
| **4** | 7846c73c25fafce260659a9f9aefbef6b514bf15 | 10.3390/ijms20081996 | The merlin-ERM (ezrin, radixin, moesin) family... | The paralogous proteins merlin (also called ne... | Michie, Katharine A.. Bermeister, Adam. <br>... | Two Si Ezrin/F |

Now you need to uplload dataset which we filtered and created you can upload it as table in Databricks and by using that you can use in Databricks notebook. Read data from Tables:

```
csv = spark.sql('select * from azureData4')
Create table to put queries on it.
csv.createTempView('mytable3')
For checking number of rows:
spark.sql('select count(*) from mytable3').show()
```

Now to preprocess data and tokenize them converting text data into numbers:

```
# Importing libraries
from pyspark.ml.feature import RegexTokenizer, StopWordsRemover, CountVectorizer
from pyspark.ml.classification import LogisticRegression

# regular expression tokenizer
regexTokenizer = RegexTokenizer(inputCol="processed_text", outputCol="words",
pattern="\\s+")
```

```
# stop words
add_stopwords = ["http","https","amp","rt","t","c","can", # standard stop words
    "#keithlamontscott","#charlotteprotest","#charlotteriots","#keithscott"] # keywords used to
pull data)
stopwordsRemover = StopWordsRemover(inputCol="words",
outputCol="filtered").setStopWords(add_stopwords)

# bag of words count
# Here we created bag of words as 10000 because more than that will take hours to compute and
sometimes fails to.
countVectors = CountVectorizer(inputCol="filtered", outputCol="features", vocabSize=10000,
minDF=5)
```

Execute above code and you can able to rokenize them.

As we created pipeline to execute them efficiently and we can process them again and again in future:

```
# Creating pipeline of basic data cleaning and creating dataset with all features.
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors])

# Fit the pipeline to training documents.
pipelineFit = pipeline.fit(csv)
# Transform dataset with new pipelined features.
dataset = pipelineFit.transform(csv)
```

To check datatypes if each column because each column is very important for making it executable on different operations:

```
dataset.dtypes
```

As we only needed features column to use it for unsupervised algorithms:

```
d = dataset.select('features')
from pyspark.ml.clustering import KMeans
# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
modelKM = kmeans.fit(trainingData)
# Make predictions
predictionsKM = modelKM.transform(testData)
```

To measure accurzcy or efficiency of our clustering algorithm we used Euclidian distance:

```
# Evaluator for K-Means
from pyspark.ml.evaluation import ClusteringEvaluator
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictionsKM)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

But as version issue is there we used different metric and it can be found as:

```
# For Pyspark CLI
wssse = modelKM.computeCost(predictionsKM)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

As we used other algorithm for comparison we could able to make it for comparison so that we can say that which would be better:
We used Bisecting K-Means:

```
from pyspark.ml.clustering import BisectingKMeans
# Trains a bisecting k-means model.
bkm = BisectingKMeans().setK(2).setSeed(1)
modelBKM = bkm.fit(trainingData)
predictions = modelBKM.transform(testData)
```

And in the same way we can able to calculate accuracy or efficiency of both of them.
As we get less Euclidian distance with bisecting, we used that for labelling our dataset.
Now, we used 4 supervised classification algorithms to check which performs best.
Logistic Regression:
Splitting of dataset:

```
(trainingData, testData) = dataset.randomSplit([0.7, 0.3], seed = 100)
print("Training Dataset Count: " + str(trainingData.count()))
print("Test Dataset Count: " + str(testData.count()))
```

To make it useful and less complex we change column names to algo friendly.

```
from pyspark.sql.functions import *
trainingData = trainingData.select(col('prediction').alias('y'), col('features'))
```

Code to execute Logistic Regression:

```
from pyspark.ml.classification import LogisticRegression
# Build the model
lr = LogisticRegression(labelCol='y', maxIter=20, regParam=0.3, elasticNetParam=0, family =
"binomial")
# Train model with Training Data
lrModel = lr.fit(trainingData)
predictions = lrModel.transform(testData)
```

To check about our model it would be:

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
print("Test: Area Under ROC: " + str(evaluator.evaluate(see, {evaluator.metricName:
"areaUnderROC"})))
```

In this way we trained and executed other model as Decision tree classifier also:

```
from pyspark.ml.classification import DecisionTreeClassifier
# Create initial Decision Tree Model
dt = DecisionTreeClassifier(labelCol="y", featuresCol="features", maxDepth=3)
# Only for cli
# dt = DecisionTreeClassifier(labelCol="y", featuresCol="features")
# Train model with Training Data
dtModel = dt.fit(trainingData)
testData = testData.select(col('prediction').alias('y'), col('features'))
# Evaluate model
from pyspark.ml.evaluation import BinaryClassificationEvaluator
predictions = dtModel.transform(testData)
```

To evaluate:

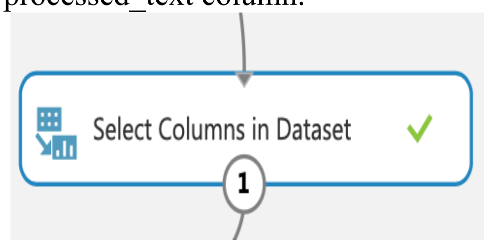
```
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
print("Test: Area Under ROC: " + str(evaluator.evaluate(see, {evaluator.metricName:
"areaUnderROC"})))
Then we performed cross-validation on logistic-regression:
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="y", maxIter=20, regParam=0.3, elasticNetParam=0, family =
"binomial")
# lr = LogisticRegression()
from pyspark.sql.functions import col
updatedTrainingData = trainingData.select('y', col('y').alias('label'), 'features')
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# Create ParamGrid for Cross Validation
paramGrid = (ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.3, 0.5]) # regularization
parameter.addGrid(lr.elasticNetParam, [0.0, 0.1, 0.2]) # Elastic Net Parameter (Ridge = 0)
.addGrid(model.maxIter, [10, 20, 50]) #Number of iterations
.addGrid(idf.numFeatures, [10, 100, 1000]) # Number of features
.build())
# Create 5-fold CrossValidator
cv = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator,
numFolds=3)
# Run cross validations
cvModel = cv.fit(updatedTrainingData)
# this will likely take a fair amount of time because of the amount of models that we're creating
and testing
```

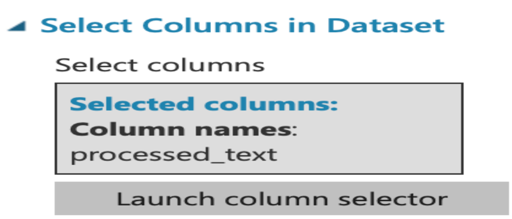
```
testData = testData.select(col('prediction').alias('y'), col('features'))
from pyspark.sql.functions import col
updatedTestData = testData.select('y', col('y').alias('label'), 'features')
# Use test set here so we can measure the accuracy of our model on new data
predictions = cvModel.transform(updatedTestData)
# cvModel uses the best model found from the Cross Validation
# Evaluate best model
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
print("Test: Area Under ROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName:
"areaUnderROC"})))
```

Once you finish all these models you can easily find that cross validated is performed much better over any other models.

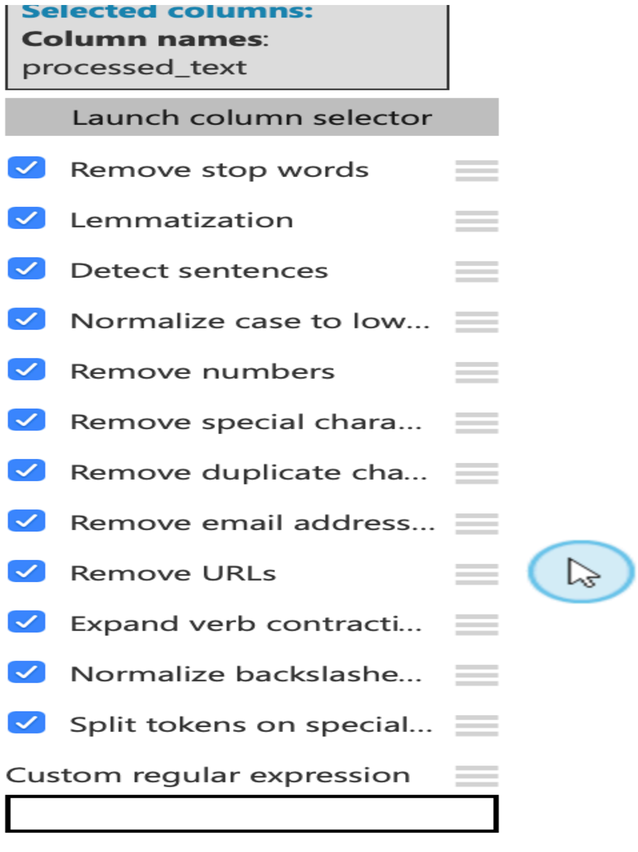
## Azure ML:

Upload dataset using local file from on your computer.
There are 2 columns there but only one column needed so use column selector and select only processed_text column.

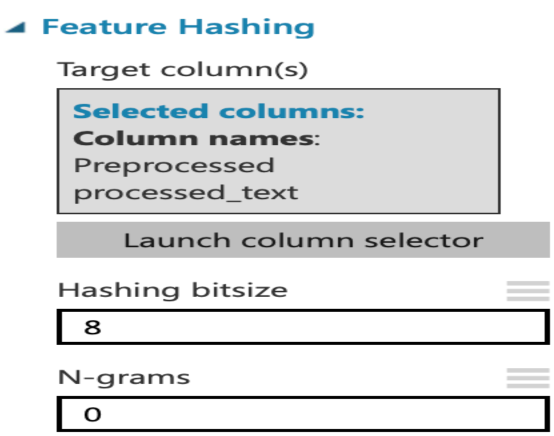




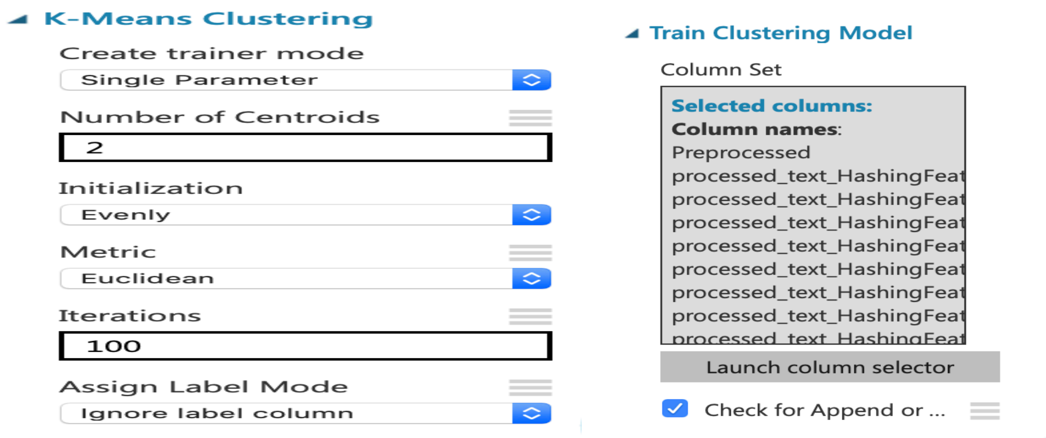
Use preprocess text to tokenize them use every property in that module:




After that convert text to numerical features and convert text to numbers:

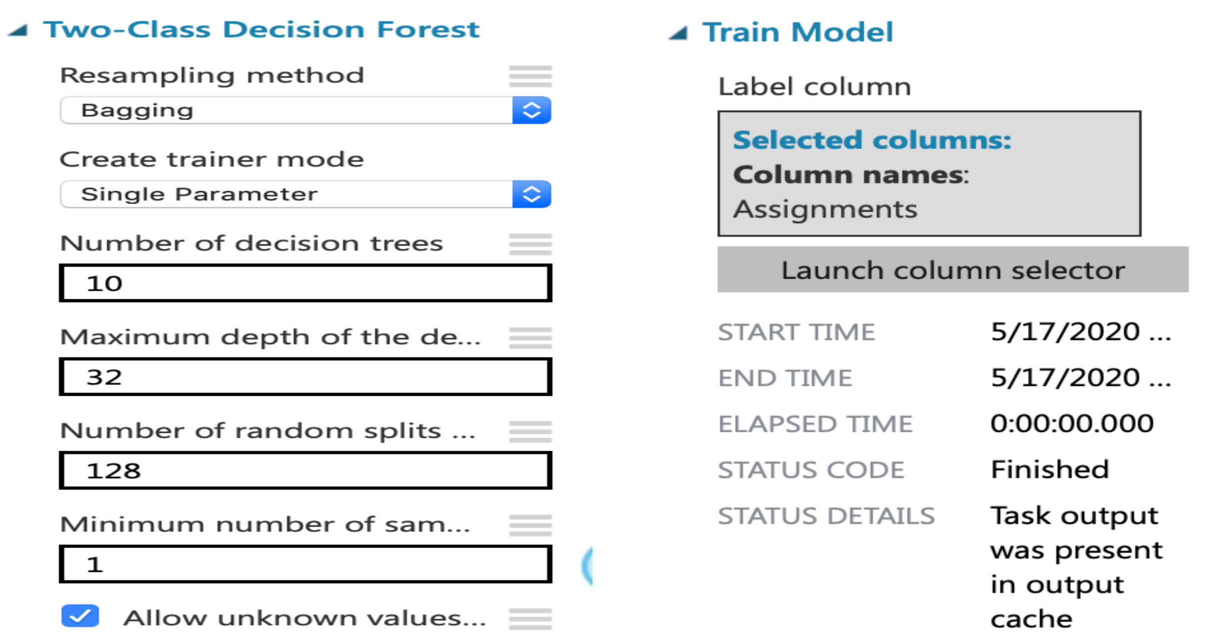


Once text converted to numbers train clustering algorithm to train it:




After that combine all data together and make it one as our labelled data and the column name of that labeled data would be assigned.
Then we train the model 4 supervised ML models and we compared them also:

Two class decision forest:



We analyzed feature importance module to check which columns has maximum importance columns:

CIS5560 Final › Permutation Feature Importance › Feature importance

| rows | columns |
|---|---|
| 256 | 2 |

view as

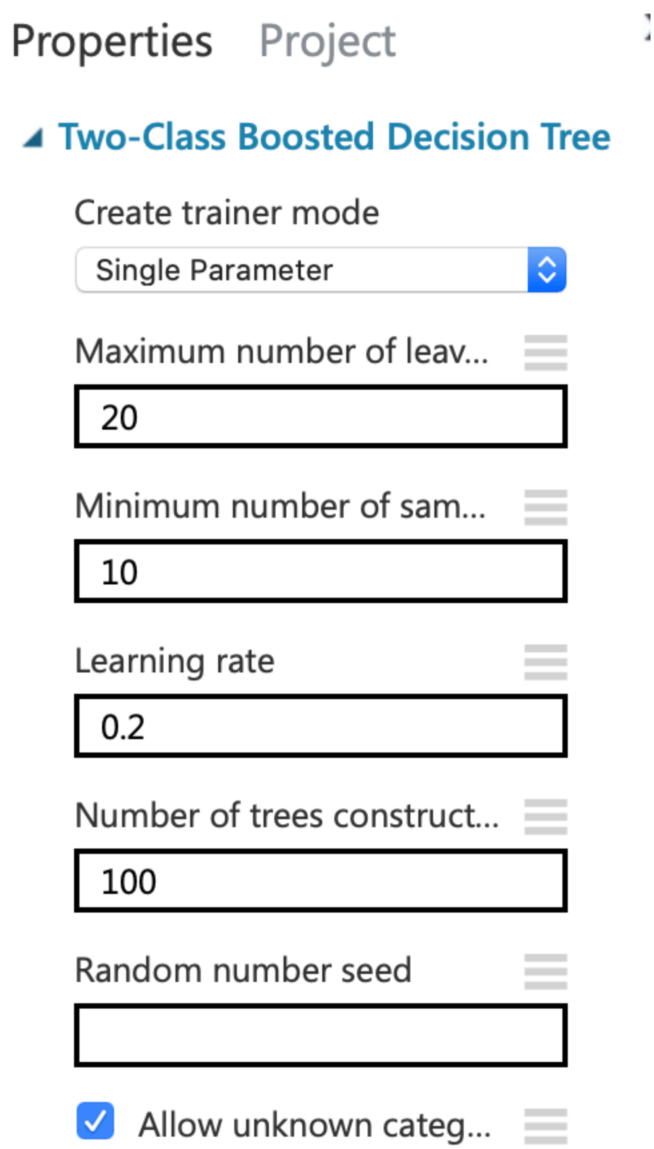
| Feature | Score |
|---|---|
| Preprocessed processed_text_HashingFeature_23 | 0.022222 |
| Preprocessed processed_text_HashingFeature_171 | 0.022222 |
| Preprocessed processed_text_HashingFeature_1 | 0 |
| Preprocessed | |

These 2 columns have most important features than any other columns.
Score of model with different metrics:

| True Positive | False Negative | Accuracy | Precision | Threshold | AUC |
|---|---|---|---|---|---|
| 14 | 4 | 0.889 | 0.933 | 0.5 | 0.932 |
| **False Positive** | **True Negative** | **Recall** | **F1 Score** | | |
| 1 | 26 | 0.778 | 0.848 | | |
| **Positive Label** | **Negative Label** | | | | |
| 1 | 0 | | | | |

In the same way we did for Two-Class Boosted decision tree:

And the properties of that model are:



Finally, we created 3 visualizations:

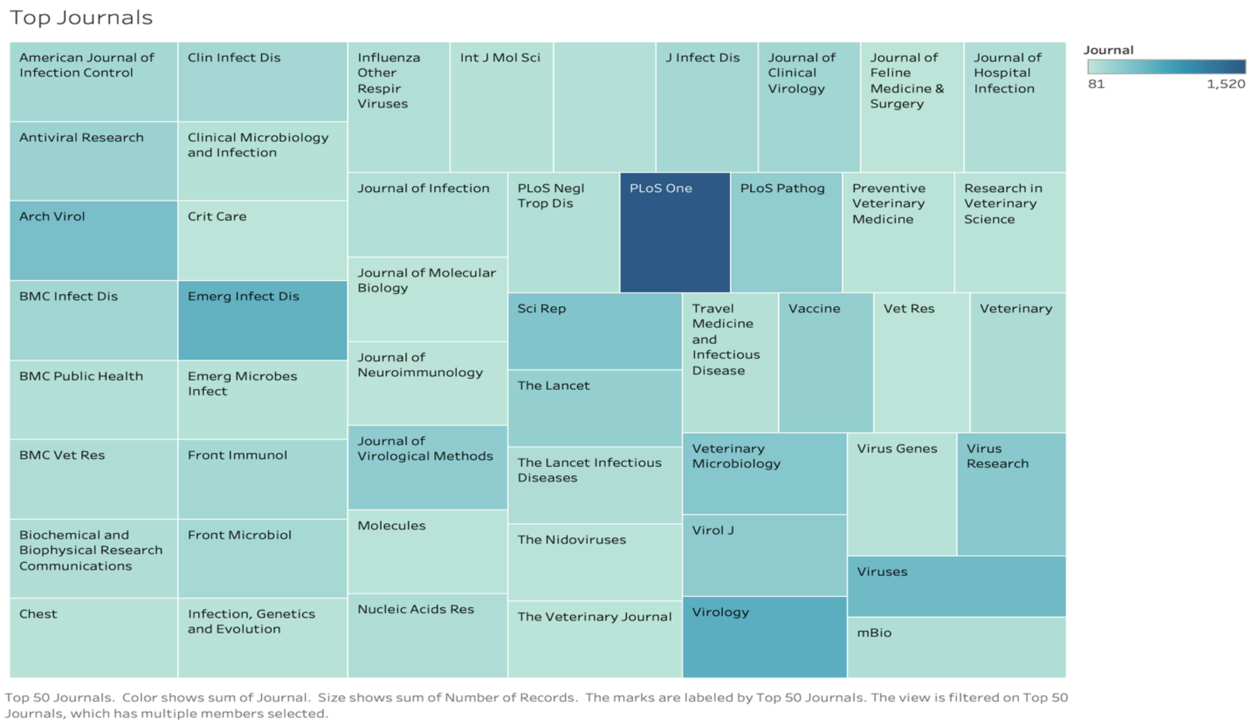



Figure 1. Top journals in which most of the articles are published. Like PLoS One is top in this category.

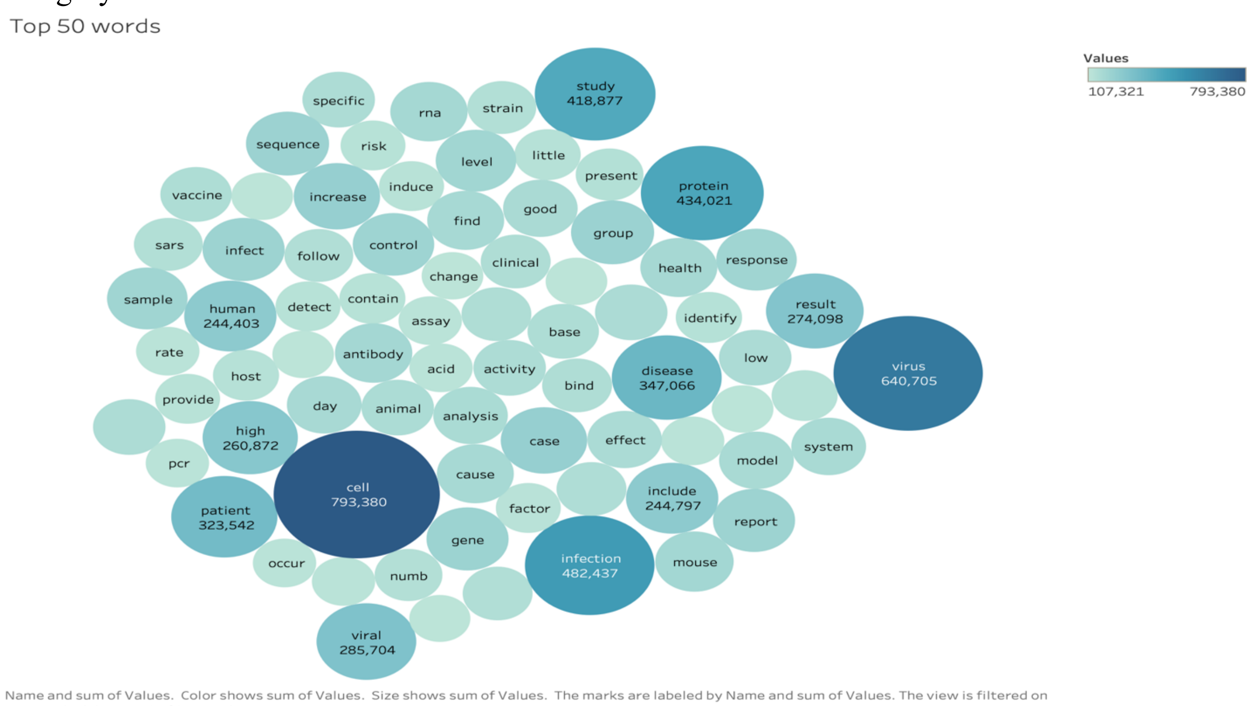



Figure 2. Top 50 most used words in all the journals.

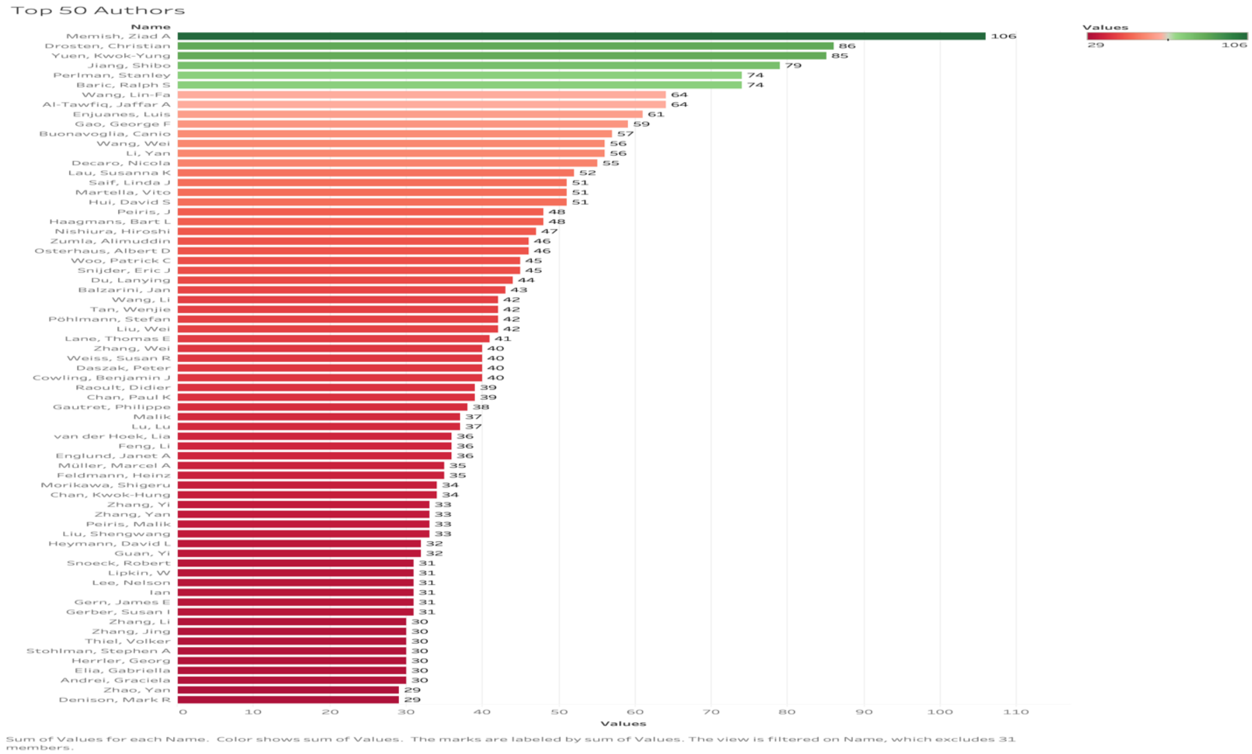



Figure 3. Top 50 authors.

All the visualizations are created in Tableau software.

## References and Github link


1. https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge/
2. https://www.kdnuggets.com/2019/05/guide-natural-language-processing-nlp.html
3. https://towardsdatascience.com/natural-language-processing-with-pyspark-and-spark-nlp-b5b29f8faba
4. https://github.com/yashchks87/covid_19_nlp