TWITTER SENTIMENT ANALYSIS

CLOUD & MACHINE LEARNING - CSCI-GA 3030-025

SRISHTI BHARGAVA, DANIEL SABBA

COURANT INSTITUTE OF MATHEMATICAL SCIENCES

NEW YORK UNIVERSITY

Table of Contents

Project Description & Overview

Project Description

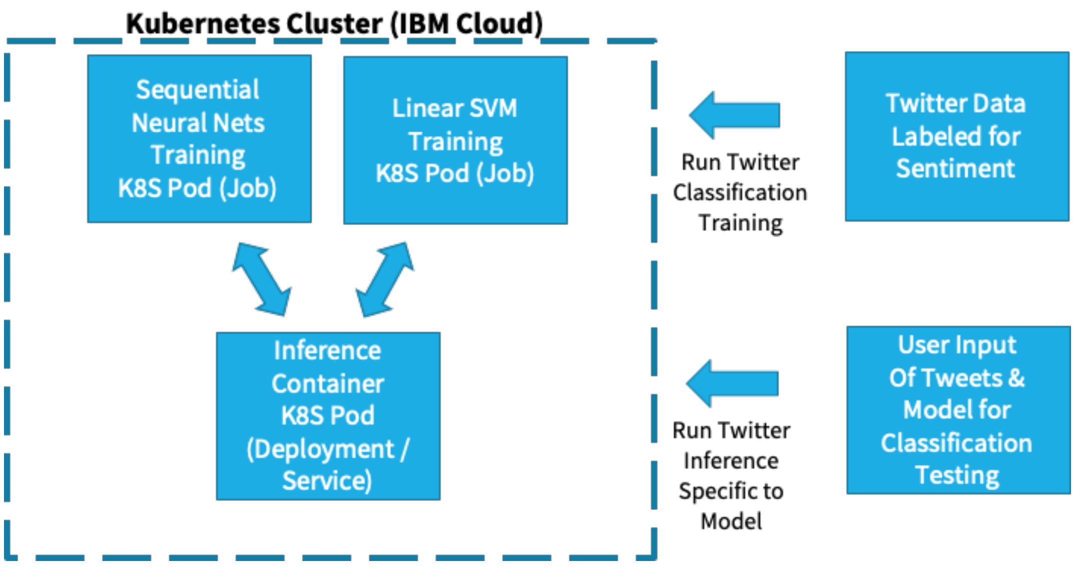
With billions of people in lockdown during the COVID19 crisis, we conjectured their perception of reality would be exclusively defined by information received through digital media. In this environment, a tool that uses machine learning algorithms to gather sentiment from this digital information would allow for uniquely precise representation of overall sentiment and could have various ancillary applications.

We designed a project that could be scaled to various models, as well as heterogeneous data sets. We identified the Kubernetes infrastructure as ideal to enable these features and devised a project architecture that would allow for user-defined tailoring to be made, as well as benefit from the scalability of cloud frameworks.

Project Architecture

1. Two training containers (jobs) run text classification training into positive, negative, neutral tweets using Deep Sequential Neural Networks and Linear Support Vector Machines.
2. An inference container (deployment) performs inference in the pre-trained models saved in volumes mounted in each training container.
3. In a web-based User Interface, a selection can be made between machine learning models, and a new tweet can be provided for inference.

BackEnd of Cloud/ML Sentiment Analysis


Kubernetes Cluster (IBM Cloud)
Sequential Neural Nets Training K8S Pod (Job)
Linear SVM Training K8S Pod (Job)
Inference Container K8S Pod (Deployment / Service)
Run Twitter Classification Training
Twitter Data Labeled for Sentiment
Run Twitter Inference Specific to Model
User Input Of Tweets & Model for Classification Testing

Inference Container & User Interface

We created an inference container as a kubectl deployment as well as a service. This allowed that container to be accessed externally. This inference container accesses different trained models saved in volumes mounted in training containers to perform inference.

The user interface allows for a new tweet to be evaluated as well as for different training models to be selected. This user interface will be hosted in a webserver and is accessing the inference container in Kubernetes external IP address and a NodePort.

Machine Learning Models for Sentiment Analysis

Sequential Neural Net Network

We used the tokenizer class from Keras to create a word to index dictionary. This dictionary consisted of each word from the corpus as a key, with a corresponding unique index as a value for that key. Furthermore, we used one-hot matrices to represent the text data, also experimented with vector embedding to create an embedding layer for the deep neural network model. Each epoch took more than 2 hours to train.

As we continued to fine tune model parameters, we settled with a 3-layer deep neural network consisting of dense layers, with one layer each for input and output along with a hidden layer along with 2 dropout layers to randomly remove data to help avoid overfitting.

Model Parametrization

- Activation Function: Experimented with ReLU and sigmoid. Used ReLU for the input and hidden layers. Used softmax on the output layer since it is a widely used activation function for multiclass classification
- Optimiser: Used adam optimiser to update the parameters and to minimize the cost of the neural network.
- Loss function: Used categorical Cross Entropy. This function calculates a score that summarizes the average difference between the actual and predicted probability distributions for all classes in the problem. It is the default loss function used for multi-class classification problems.

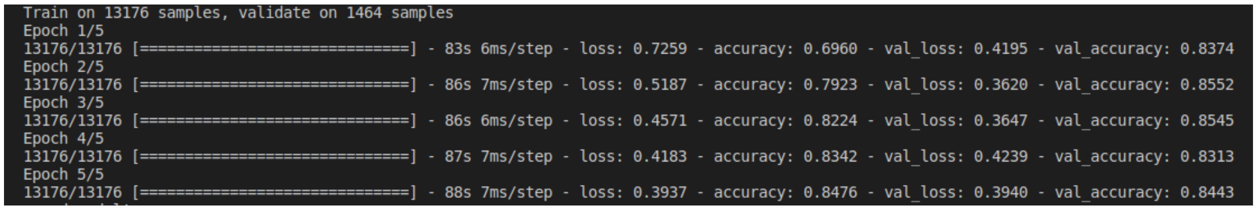
Linear Support Vector Classification

Linear SVM performs classification by finding the best hyperplane to differentiate the n classes in a n-dimensional space. The plane acts like a decision boundary for any new data point since a new data point could be classified based on which side of the plane it lies in the n-dimensional space. Performed tf-idf transformation on the data before feeding it to the model. For ease during the inference, saved states of the vectorized as well as the trained model.

Model Performance and Comparison

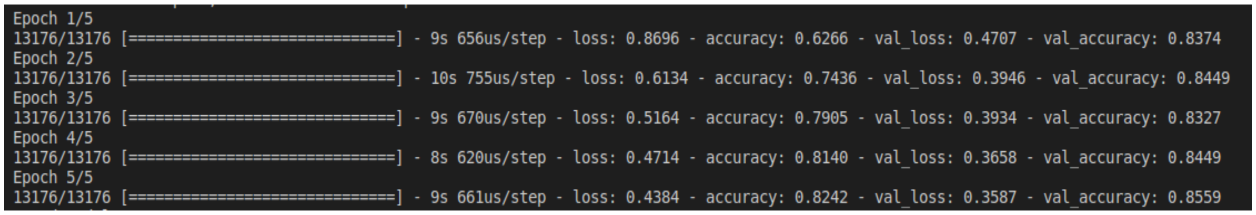
Achieved 85.2% accuracy on the Neural Net model, experimented with a bunch of epochs and batch sizes.

Batch size : 3



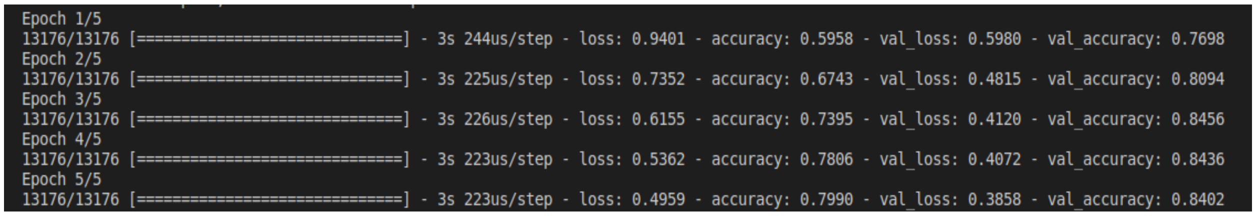
```
Train on 13176 samples, validate on 1464 samples
Epoch 1/5
13176/13176 [==============================] - 83s 6ms/step - loss: 0.7259 - accuracy: 0.6960 - val_loss: 0.4195 - val_accuracy: 0.8374
Epoch 2/5
13176/13176 [==============================] - 86s 7ms/step - loss: 0.5187 - accuracy: 0.7923 - val_loss: 0.3620 - val_accuracy: 0.8552
Epoch 3/5
13176/13176 [==============================] - 86s 6ms/step - loss: 0.4571 - accuracy: 0.8224 - val_loss: 0.3647 - val_accuracy: 0.8545
Epoch 4/5
13176/13176 [==============================] - 87s 7ms/step - loss: 0.4183 - accuracy: 0.8342 - val_loss: 0.4239 - val_accuracy: 0.8313
Epoch 5/5
13176/13176 [==============================] - 88s 7ms/step - loss: 0.3937 - accuracy: 0.8476 - val_loss: 0.3940 - val_accuracy: 0.8443
```

Batch size: 32



```
Epoch 1/5
13176/13176 [==============================] - 9s 656us/step - loss: 0.8696 - accuracy: 0.6266 - val_loss: 0.4707 - val_accuracy: 0.8374
Epoch 2/5
13176/13176 [==============================] - 10s 755us/step - loss: 0.6134 - accuracy: 0.7436 - val_loss: 0.3946 - val_accuracy: 0.8449
Epoch 3/5
13176/13176 [==============================] - 9s 670us/step - loss: 0.5164 - accuracy: 0.7905 - val_loss: 0.3934 - val_accuracy: 0.8327
Epoch 4/5
13176/13176 [==============================] - 8s 620us/step - loss: 0.4714 - accuracy: 0.8140 - val_loss: 0.3658 - val_accuracy: 0.8449
Epoch 5/5
13176/13176 [==============================] - 9s 661us/step - loss: 0.4384 - accuracy: 0.8242 - val_loss: 0.3587 - val_accuracy: 0.8559
```

Batch size: 100



```
Epoch 1/5
13176/13176 [==============================] - 3s 244us/step - loss: 0.9401 - accuracy: 0.5958 - val_loss: 0.5980 - val_accuracy: 0.7698
Epoch 2/5
13176/13176 [==============================] - 3s 225us/step - loss: 0.7352 - accuracy: 0.6743 - val_loss: 0.4815 - val_accuracy: 0.8094
Epoch 3/5
13176/13176 [==============================] - 3s 226us/step - loss: 0.6155 - accuracy: 0.7395 - val_loss: 0.4120 - val_accuracy: 0.8456
Epoch 4/5
13176/13176 [==============================] - 3s 223us/step - loss: 0.5362 - accuracy: 0.7806 - val_loss: 0.4072 - val_accuracy: 0.8436
Epoch 5/5
13176/13176 [==============================] - 3s 223us/step - loss: 0.4959 - accuracy: 0.7990 - val_loss: 0.3858 - val_accuracy: 0.8402
```

The best model performance was found with batch size as 32.

Batch Size	Time taken per epoch
3	80 seconds
32	9 seconds
100	3 seconds

As the batch size increases, the calculations in the neural network get parallelized thus leading to lesser time taken per epoch. From the above table, it is clear that time taken per epoch reduces as the batch size increases.

Achieved 80% accuracy with the Linear Support Vector Machine model.

Dataset & Model Observations

Dataset has a high number of negative tweets, about 9k out of the total 15k tweets are negative, hence the Linear SVM tends to predict some of the neutral tweets as negative.

This indicates that the algorithm is biased towards the majority class and it probably does not take the data distribution into consideration.

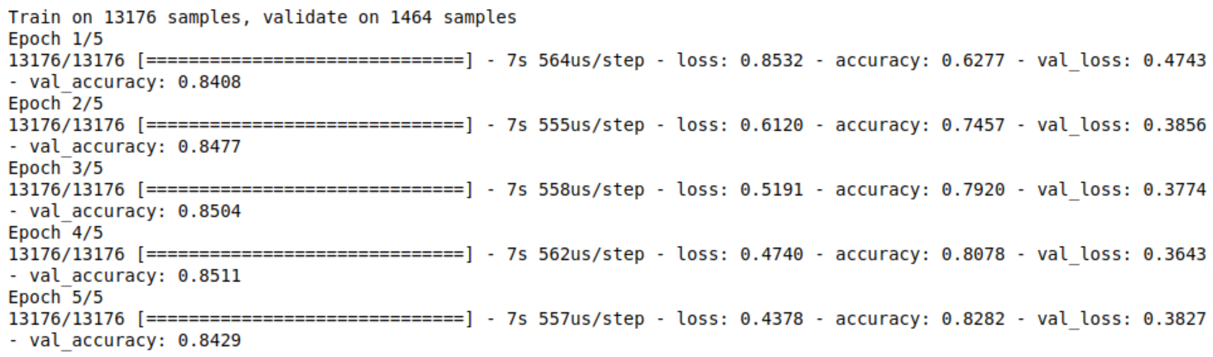
The Linear SVM model gives a high prediction accuracy over the majority class, and maintains a reasonable accuracy for the minority classes (neutral and positive)

On the other hand, the neural network model that we built in this project for performing sentiment analysis does a better job at handling an unbalanced dataset. The categorical cross-entropy loss function that is used in our Sequential neural network helps to maximise the performance of the model while taking into account the imbalances in the dataset.

Performance comparison between different cloud infrastructures

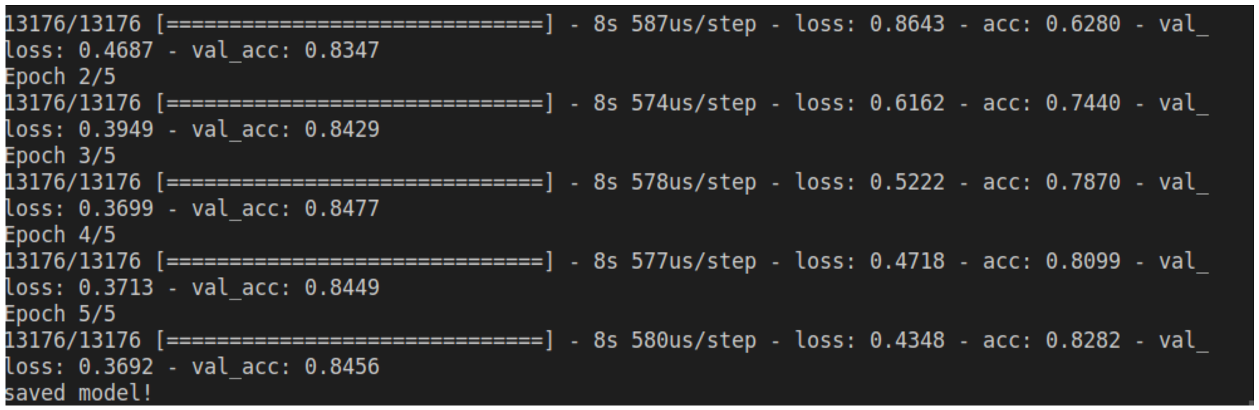
We trained the deep neural network model on both IBM Cloud and Amazon Web Services. Typically, which machine learning platform is best, majorly depends on the user’s requirements and final goals. However, in our situation we saw very similar training performance on both the platforms. Time taken for each epoch for the same hyperparameters was approximately the same. AWS took slightly less time per epoch to train the model. However, training on IBM cloud achieved a slightly higher accuracy of 84.5% as compared to 84.2% on AWS.

AWS



```
Train on 13176 samples, validate on 1464 samples
Epoch 1/5
13176/13176 [==============================] - 7s 564us/step - loss: 0.8532 - accuracy: 0.6277 - val_loss: 0.4743
- val_accuracy: 0.8408
Epoch 2/5
13176/13176 [==============================] - 7s 555us/step - loss: 0.6120 - accuracy: 0.7457 - val_loss: 0.3856
- val_accuracy: 0.8477
Epoch 3/5
13176/13176 [==============================] - 7s 558us/step - loss: 0.5191 - accuracy: 0.7920 - val_loss: 0.3774
- val_accuracy: 0.8504
Epoch 4/5
13176/13176 [==============================] - 7s 562us/step - loss: 0.4740 - accuracy: 0.8078 - val_loss: 0.3643
- val_accuracy: 0.8511
Epoch 5/5
13176/13176 [==============================] - 7s 557us/step - loss: 0.4378 - accuracy: 0.8282 - val_loss: 0.3827
- val_accuracy: 0.8429
```

IBM



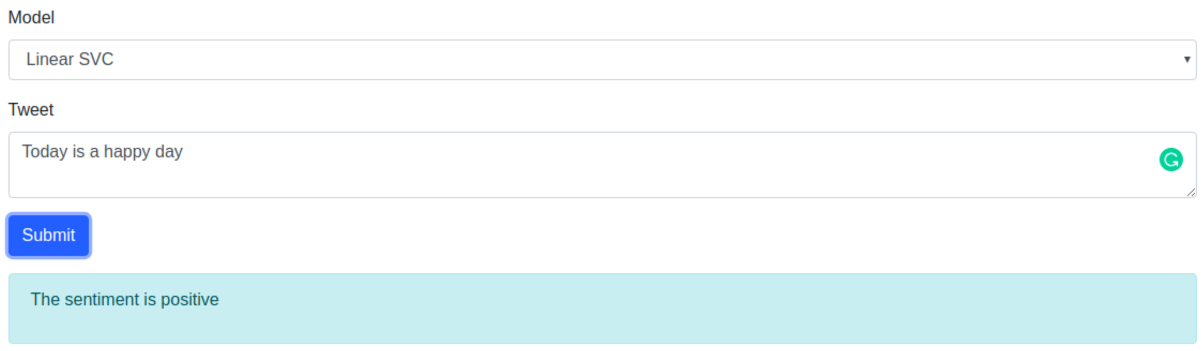
```
13176/13176 [==============================] - 8s 587us/step - loss: 0.8643 - acc: 0.6280 - val_
loss: 0.4687 - val_acc: 0.8347
Epoch 2/5
13176/13176 [==============================] - 8s 574us/step - loss: 0.6162 - acc: 0.7440 - val_
loss: 0.3949 - val_acc: 0.8429
Epoch 3/5
13176/13176 [==============================] - 8s 578us/step - loss: 0.5222 - acc: 0.7870 - val_
loss: 0.3699 - val_acc: 0.8477
Epoch 4/5
13176/13176 [==============================] - 8s 577us/step - loss: 0.4718 - acc: 0.8099 - val_
loss: 0.3713 - val_acc: 0.8449
Epoch 5/5
13176/13176 [==============================] - 8s 580us/step - loss: 0.4348 - acc: 0.8282 - val_
loss: 0.3692 - val_acc: 0.8456
saved model!
```

KaaS & CaaS: Kubernetes (Container) as a service

The cloud ethos do “Product” as a Service was very much present in this project. For instance, we used IBM Cloud to host our Kubernetes cluster – this service is a hybrid of KaaS (Kubernetes as a Service) and CaaS (Container as a Service). We also utilized a CaaS framework to effect performance comparison – we utilizing training containers in AWS and IBM Cloud, leveraging their CaaS capabilities to allow for “apples to apples comparison of virtualized environments with similar parameters/configurations. As evidenced above, the performance between AWS and IBM Cloud was very similar – this conclusion was simplified by the utilization of both services CaaS framework.

Web-User Interface

The user interface provides the user with a dropdown menu to choose between the two models. After entering the text in the Tweet box, the predicted sentiment and confidence (in case of sequential neural network model) would be displayed in a box below.

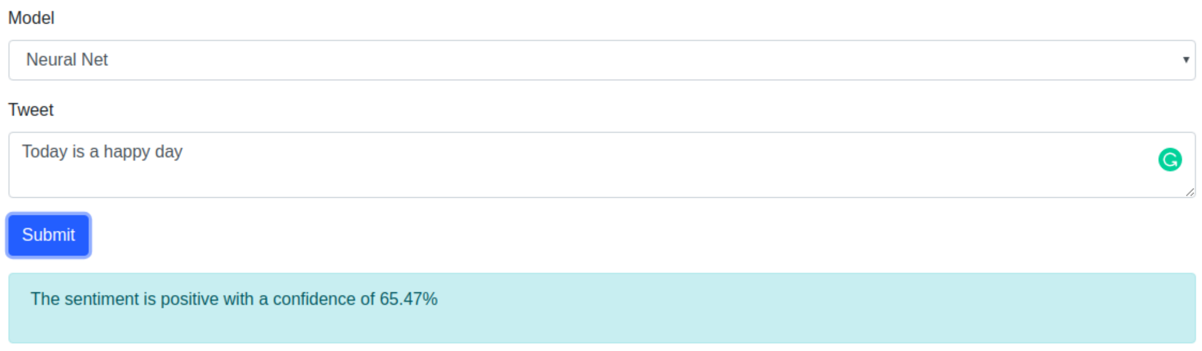

Model

Linear SVC

Tweet

Today is a happy day

The sentiment is positive


Model

Neural Net

Tweet

Today is a happy day

The sentiment is positive with a confidence of 65.47%

Personal Discussion

Challenges & Lessons Learned

Map Reduce Container

Our initial idea was entirely different. We had intended to create a map/reduce cluster that would perform large scale SQL-type selections of tweets with certain keywords and locations. We had intended to train these tweets for clustering / sentiment analysis so as to be able to stratify how different segmentations of the population react to various topics. For instance, we wanted to identify the sentiment towards hotels from how individuals tweeting from Mumbai than from individuals in NY (using #covid and #hotels hashtag)

Clustering

We quickly found out that the clustering algorithm was not giving us meaningful results and we pivot the project swiftly to sentiment analysis. We progressed with the plan of a map/reduce framework for a few days and then realized that it would disallow us from being able to have pre-trained models, as a dynamic query from an user would need to be processed in a map/reduce container and then be trained, taking a considerable amount of time. The use and validity of the hybrid map/reduce + ML architecture is still high, but not something we would want to demonstrate in a live setting at a project presentation.

Data considerations

We had intended to perform text classification (for example identifying topic) instead of sentiment identification. We were thwarted in that effort by the very limited amount of publicly available twitter datasets with topic labels. We struggled to find good datasets even for tweets with sentiments that included neutral, beyond positive and negative. It became obvious pretty quickly how important it is to have “ready-to-use data” to be able to focus on substantive ML/Cloud aspects of the project. We realized this as we were dedicating a lot of time preparing raw data from the API.

Model Selection

We evaluated 4-5 training models, settling with the final two (SVM & SNN) for accuracy considerations. In hindsight, we could have not made an ex-ante selection of these two models, but instead kept all of them as independent containers the user could select based on whatever data is to be evaluated. These are the models we evaluated:

1. Recurrent Neural Networks
2. Sequential Neural Networks
3. Naive Bayes

4. Linear SVM
5. Random Forest

Project Workstream Report

Both project partners split the work dynamically on time blocks. We agreed on a predetermined working schedule (for example from 7pm-1am, or 8pm-2am) over a period of 3 weeks, doing most of the work in that time frame, with obvious exceptions of late night / early morning debugging. Daniel took care of the inference service and deployment of the project to IBM Kubernetes. Srishti focussed on training the Machine Learning models and their fine tuning, along with the web user interface. Having said this, there was a lot of overlap in our responsibilities and work. The partners would sync up multiple times a night over calls to divide and conquer the work dynamically. This ensured a very balanced contribution from both partners.

Conclusion and Future Work

Kubernetes was a game changer
Throughout this project, we were able to experience the benefits of scalability of the Kubernetes infrastructure. It allowed us to make a job performed by source code into a dynamic living environment that is scalable, elastic and can be accessed everywhere.

These features are particularly relevant as, while the skeleton of the model is completed, we need to continue to improve the data environment and fine tune the models appropriately – this can be uniquely done in the Kubernetes cluster framework, which allows us to create new clusters as we experiment with new models, while old modes can continue to run for comparison.

Project as a living organism
The precision of this model can be enhanced with more labeled twitter data. Now this project is done, we are exploring alternatives to scale this data up so the tool can be more effective and useful. We can continue to fine tune the balance of the dataset so that there is no particular major class by undersampling or oversampling each class present or try more model configurations.

We intend to use our Google Cloud credits to maintain the UI & cluster running as we continue to refine the tool and collaborate on this idea. This will allow us to have a continued demo as well as a repository page for source code.

Potential practical applications
One immediate commercial application would be on the design of systematic trading strategies to capitalize on heterogeneous sentiment clustering across different markets. Immediate applications would be global long short equity and fixed income relative value.

Bibliography & Reference

Relevant websites:

1. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
2. http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/
3. https://towardsdatascience.com/covid-19-with-a-flair-2802a9f4c90f
4. https://towardsdatascience.com/kaggle-released-cord-19-an-ai-challenge-on-the-covid-19-50d657378ff4
5. https://towardsdatascience.com/how-did-twitter-react-to-the-coronavirus-pandemic-2857592b449a
6. https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge/kernels
7. https://www.kaggle.com/smid80/coronavirus-covid19-tweets#2020-03-16%20Coronavirus%20Tweets.CSV

Relevant code:

8. https://gist.github.com/vgpena/b1c088f3c8b8c2c65dd8edbe0eae7023
9. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
10. https://github.com/prakashpandey9/Text-Classification-Pytorch/tree/master/models