Cloud Computing Project

Malo Ferriol individual group 2

May 2020

Abstract

This project aims to assess the possibility to use the cloud native services from AWS and Azure to create a new dataset. We will test the result of a sentiment analysis on tweets with machine learning model. We will compare three models trained on different dataset. The first model will use the ground truth label from the original dataset. The second will be trained on the dataset created with the label from the cloud services. The third will be an unsupervised model. We compared the result on a given test set and found that the supervised model trained on the new dataset from the cloud services yields result very close to the one trained on the original dataset.

1 Introduction

Our project aims to test if it is possible to use Cloud Native Services in order to label dataset to improve the results of a machine learning model. The task used to test the results is a multi-class classification task based on text data (here tweets). We will develop three classification models. The first one is an unsupervised model which aims to show the result if the dataset has never been labelled. The second one is a model based on the cloud-based label from the service AWS Comprehend and Azure Cognitive. The third one is based on the labelled dataset for the purpose of comparing the result to an optimal solution. In our report we will also compare the result from the cloud-based services to the ground-truth in order to analyse the result from those services. We made some changes from the previous report. We have decided to change the dataset used in order to improve our analysis. Previously, we contemplated the idea to use real-time data from Twitter gathered from their API with the topic "coronavirus". We decided to make an adjustment and use labelled dataset. This will allow us to be able to assert the result from the cloud services. We then proceeded to look for a dataset with multi-class label. We were concerned that the result from a binary label (positive, negative) would be subject to randomness so we selected a dataset with three labels (neutral, positive, negative).

2 Original Solution

2.1 Presentation

We create an example architecture for a specific workflow. The architecture is based upon AWS cloud. The objective is to aggregate tweets from the Twitter API with a specific query. Then use three different cloud services to annotate their sentiment. We use AWS comprehend, Azure Cognitive Text API and Google Natural Language API. Then use the results from all platforms to create a new dataset. Once the dataset is completed, we train our own personal model again.

2.2 Cloud based Architecture

For the purpose of the project we defined a possible cloud-based architecture. It is possible to obtain a similar result using different architectures. Since the objective is to leverage the advantages of cloud native machine learning services, we thought it would be logical to use the power of the cloud as much as possible. Training a model takes time and computational power. The result can be inconsistent. To cope with the issue of time and computational problem we will schedule the training of the model. It could be, for example, once a month. Since the resource required will only be used for this purpose and from time to time, we thought it would be wiser to use per request services. AWS develop AWS Lambda a "serverless computing service". To manage the workflow, we decided to use AWS Step Functions. To store the data, we will use AWS S3 (Simple Storage Service). Then for the computation, we will use an EC2 instance on which we will train our own model. Using AWS Lambda to launch a pre-existing script in the new EC2 instance.

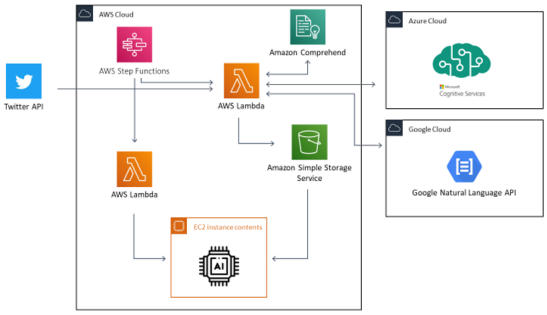

Figure 1: Schema Architecture Cloud

2.3 Model Definition

The objective is to develop a model that could automatically perform training on a new dataset. This has been made possible in the cloud using image docker. We can write our own machine learning model. We can deploy it simply and launch the task it must perform with the image docker and the AWS Lambda script.

3 Experiments implementation

3.1 Local solution

3.2 Model definition

For this model we use the NLTK package. Since we do not have any dataset to train the model, we will simply perform the classification task. First, we start by tagging the word with respect to their class (nouns, verbs, adjectives, and adverbs), then we use the function SentiWordNet to compute the polarity (positive, negative) of each word. From this score we create a rule to generate an output (positive, neutral, negative).

3.3 Our solution

The original solution for which we created the architecture is too big to be implemented for this project. To be able to evaluate this project we decided to implement solely some part of it. Then, we manually executed the task. The use of the cloud services was restricted to the minimum to reduce the cost and the time of implementation

3.3.1 Create a new Dataset

The new dataset is based upon the training set without the ground truth result. We used both AWS Comprehend and Azure Cognitive services to obtain labels. We aimed to use also Google Cloud Platform, but we faced an issue with the billing system and did not want to take any risk. We decided that having already two different label sets will be sufficient for our project. Once those labels were obtained, we proceeded to create a rule to assign a new label for each sample. We applied a very simple rule, for a tweet if both labels were the same then this tweet kept this label. We added another rule, if one label was "neutral" and the other "mixed" then the label was neutral. This addition is in order to keep some sample with the label mix but without making the choice between positive or negative in case of a mixed sample.

3.3.2 Local Testing

The task we chose to test our solution is a multiclass classification Sentiment analysis based on tweets. To simplify the task, we limited the scope of the research to one machine learning classifier Logistic Regression. The package used is scikit-learn. For hyperparameter tuning we used GridSearch. We tuned several steps of the machine learning pipeline. We used an encoder to perform TF-IDF.

3.4 Optimal solution

3.4.1 Model definition

We used the same model characteristics as the one we trained on the cloud labelled dataset. However, this model is trained based upon the true label found in the original dataset.

4 Results Analysis

4.1 Study of the Dataset

4.1.1 Analysis of ground-truth compared to cloud-based results

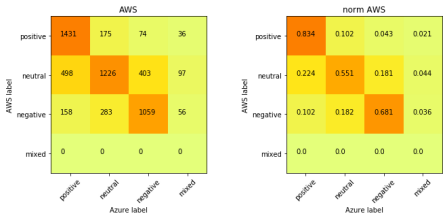

Figure 2: AWS vs Ground Truth

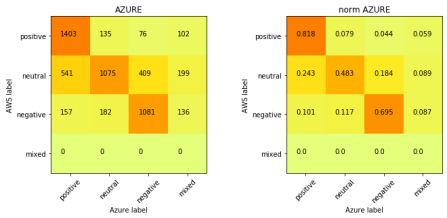

Figure 3: Azure vs Ground Truth

From the comparison of the original dataset to the result we obtained from AWS see Figure 2 we see that the category for which it has most errors is the neutral category. Almost half of the neutrals are misclassified.

The result from the Azure services see Figure 3 is pretty similar to the one obtained from AWS. There are more results with the "mixed" label.

Figure 4: AWS vs AZURE

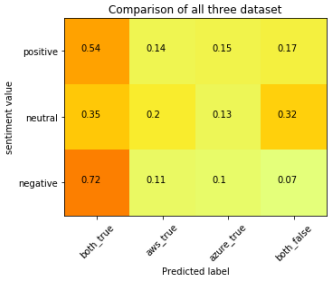

Figure 5: All three dataset

Now, we compare the result from both Cloud Services see Figure 4. We averaged the result over all the data. Personally, we were expecting the results to be closer than they were. They rarely assigned opposite label (negative and positive) which represented only 5% of the result. They also did not so frequently assign the label mixed at the same time (only 8,5% of the mixed result). The result was either mixed or neutral from one or the other for 23,5% of the mixed result.

Now we will compare all three dataset at the same time see Figure 5. For this purpose, I created a table inspired by the confusion matrix. The row represents the ground truth label. The column represents the label from both Cloud Services. The first column is when both Services have the same label and the same as the ground truth. The second is when AWS is accurate but not Azure. The third is when Azure is accurate but not AWS. The last one is when both services chose a wrong label.

We see that the negative label is well categorized. The neutral label is often wrongly assigned. Regarding the positive label, they are both right half of the time.

5 Model Result

5.1 Optimal Model

	precision	recall	f1-score	support
positive	0.77	0.67	0.72	1103
neutral	0.58	0.72	0.65	1430
negative	0.65	0.53	0.58	1001
accuracy			0.65	3534
macro avg	0.67	0.64	0.65	3534
weighted avg	0.66	0.65	0.65	3534

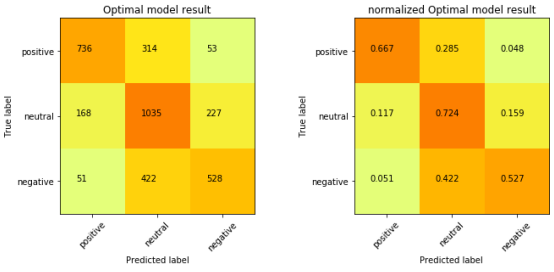

Figure 6: Result from optimal model

The results are based upon a dataset of 3,534 tweets. The precision for the class positive and negative is good. For the class neutral we see that the recall is good but not the precision. The recall is good for positive. The class neutral and negative are more likely to be misinterpreted as one another.

5.2 Our Model

	precision	recall	f1-score	support
positive	0.66	0.72	0.69	1103
neutral	0.59	0.69	0.64	1430
negative	0.68	0.46	0.55	1001
accuracy			0.63	3534
macro avg	0.65	0.62	0.63	3534
weighted avg	0.64	0.63	0.63	3534

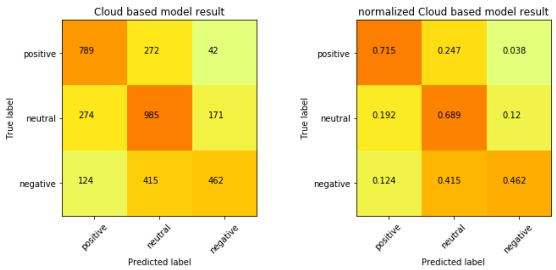

Figure 7: Result from our model

The precision for the class positive and negative is good. For the class neutral we see that the recall is good but not the precision. The recall is good for positive. The recall for negative is terrible. Less than half of the label "negative" are actually negative.

5.3 Unsupervised Model

	precision	recall	f1-score	support
positive	0.63	0.36	0.46	1103
neutral	0.45	0.84	0.58	1430
negative	0.68	0.14	0.24	1001
accuracy			0.49	3534
macro avg	0.59	0.45	0.43	3534
weighted avg	0.57	0.49	0.45	3534

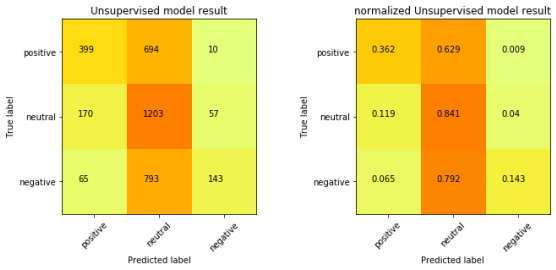

Figure 8: Result from Unsupervised model

The result are too inconsistent to be useful. Most of the samples are labelled neutral. Only 14

5.4 Result comparison

Overall, we see that the accuracy for the supervised model is very close between optimal model and the one based on the cloud based dataset. The model from unsupervised dataset is very far from being useful since it not trustworthy. The model trained on our new dataset is very close to the one trained on the ground truth dataset. Even the precision and the recall are not far from it.

6 Conclusion

This is a very specific problem. The idea behind it is that it allows the project to build a faster and cheaper dataset than to use a service such as Amazon Mechanical Turk. The result obtained from the analysis of the model result demonstrated that it is actually worth it to use this method. The results are not as good but not so far either. However, this was possible for a classification that was not specialized. The result from the Cloud Native Services could degrade greatly in case of a specified task. There are plenty of possible improvements. It would be worth trying another cloud native system such as IBM and Google. The size of the dataset could be increased. The machine learning classifier could be improved (testing more classifier, use neural network).

A Artifact Appendix

A.1 Artifact check-list (meta-information)

- **Algorithm: Tweet sentiment analysis**
- **Program: Scikit-Learn, AWS Comprehend, Azure Cognitive service, NLTK**
- **Model: SentiWordNet, Logistic Regression**
- **Data set: Included (train (5500 tweets) tweets and test (4000 tweets) set)**
- **Run-time environment: Mac OS**
- **Hardware: Intel core i7 CPU**
- **Metrics: Accuracy, F1 score, Confusion Matrix**
- **Output: Classification Report**
- **Experiments: Compare classification result from three models**
- **How much disk space required (approximately)?: 1GB**
- **How much time is needed to prepare workflow (approximately)?: A few minutes**
- **How much time is needed to complete experiments (approximately)?: A few minutes**
- **Publicly available?: Yes**

A.2 Description

A.2.1 How to access

Clone repository from GitHub : https://github.com/malof/cloud-computing-project

A.2.2 Data sets

The dataset is included in the repository. The datasets from our experiments are also included for reproduction purpose.

A.2.3 Models

We use very simple machine learning models. SentiWordNet from NLTK and Logistic Regression from Scikit-Learn.

A.3 Installation

Follow the installation section in the ReadMe file in the repository.

A.4 Evaluation and expected result

The results are confusion matrix and classification report for each of the model used in the experiment.

A.5 Experiment customization

If you wish to custom the experiment you will need to put your own Azure Cognitive Service Key in the file 'script_azure.py' and add your own AWS credentials to your terminal.

Once your files are updated you simply need to change the 'train.csv' file in the original dataset section.

If you wish to use more than 20% of you dataset for you experiment you can simply modify the file script_extract_smaller_dataset.py. Either change the number in the scikit learn function or simply rename you dataset 'dataset.csv' in the same directory as 'train.csv'