Sentiment Analysis Project

Project Scope and Documentation

PREPARED BY

Nishant Parmar

Economist Intelligence Unit

Contents

---

1. Project Overview

The Italian governing coalition is fragmenting and there is a risk of an early/snap election being called – this is the risk we want to measure. The broad-based (largely static) political stability index using EIU content that can be transferred across countries, can be viewed *here*. To specify the index's results towards particular events and to add more real-time volatility to the score, we could give heavier weighting to the long-term 'election schedule' indicator in this example, as well as relying on sentiment analysis in the later stages of the process.

Sentiment Analysis

There are two machine learning methodologies that can be used to implement sentiment analysis on a textual dataset - supervised learning models, and unsupervised learning models. In our case, the datasets can be anything from the domain of political data (specifically related to our requirements) - like articles and tweets.

The supervised learning method involves a huge collection of **labeled data** being passed to a machine learning model to train it to learn about the features of this dataset. The labels denote sentiment (or classifies) whether this piece of text is either ***positive*** (occurrence of a snap election, public approval for a candidate etc) or ***negative*** (snap election didn't happen, public disapproval for a candidate etc).

An unsupervised learning model helps find previously unknown patterns in data set without these pre-existing labels.

Based on this learning, the model would then make a "guess" to predict the "best possible" outcome.

2. Challenges and Obstacles

Unavailability of training data (Supervised learning)

Inferring from research[1][2] on natural language processing using supervised deep learning models for sentiment analysis, the networks really need to learn from high volume domain-specific datasets.

For instance, generalized sentiment analysis on text would use IMDB reviews dataset as the source of training data. A phrase such as "*it was like Tarantino*" would be associated with positive sentiment but in the political domain, it is quite the opposite.

Also, owing to lack of availability resources (analysts), I have been personally annotating the articles and tweets that would be fed back to the model to improve its efficiency.

Model selection and deployment

After analyzing various deep-learning models to effectively classify textual data according to the requirements of the project, Google AI's BERT was selected, considering its ability to perform effective transfer learning.

Deploying the fine-tuned BERT model on a cloud service - for the ease of accessibility and management - posts some challenges, from selection of the deployment tools to the actual deployment.

Although TensorFlow provides a hosting service, the documentation is quite lengthy and not very helpful in terms of troubleshooting, and seems to suffer from a tunnel vision. The same goes for AWS Sagemaker.

For a breezy deployment, I have been following Cortex Lab's Cortex to deploy the model as a web API on AWS.

API Restrictions

Apart from the aforementioned challenges, there are certain limitations to the usage of the Twitter Search API and Google Cloud Translate API.

For instance, the Sandbox (free) subscription of the Twitter API allows only 25K calls/requests in a month, whereas the Enterprise version allows upto 5M requests.

3. Research & Development

Choosing the right model to solve a problem requires a lot of research and experimenting at each level of the machine learning workflows.

Datasets

The following is a list of the sources of datasets and important web links for the best practices for implementing Natural Language Processing:

- Google Dataset Search - a search tool to find data sets stored across the web by way of a simple keyword search. The tool surfaces information about data sets hosted in thousands of repositories across the web, making these data sets universally accessible and useful.
- VADER, IBM Watson or TextBlob: Which is Better for Unsupervised Sentiment Analysis - a comparative analysis of Python libraries to leverage unsupervised learning for sentiment analysis
- Dataset for political datasets - a comprehensive list of datasets relevant to politics
- We have also made use of our search automation tool (innovation day), ***Firebolt***, to save relevant local language Google Search results in Google Drive
- A Million News Headlines Dataset
- Australian Elections Dataset

- Understanding Convolutional Neural Networks for Text Classification
- Neural Networks as Explicit Word-Based Rules

Source Code

The following is a list of links to Python scripts written in **Google Colaboratory** notebooks, while exploring the best approach for implementation of the proof of concept:

- TF_NLP_HandleOverfitting – code to train a neural network on an input dataset (Sentiment140) with overfitting reduced
- TF_NLP_embedding_visualizer – code to visualize word embeddings in a 3-dimensional space at https://projector.tensorflow.org/
- VADER_Mark1 – code to calculate sentiment scores of individual sentiments from sentences, recorded in a tabular form
    - VADER (Valence Aware Dictionary for sEntiment Reasoning) is a library in Python which uses a combination of qualitative and quantitative methods to produce, and then empirically validate a gold-standard **sentiment lexicon** that is especially attuned to microblog-like contexts
    - VADER is used here as an unsupervised learning approach to heuristically produce sentiment scores (positive, negative, neutral, compound) which can be analyzed and meanings can be inferred from this collection of scores
- NewsAPI – code to extract news headlines (top headlines/everything) from multiple news sources using News API and stored as CSV in Google Drive (folder)

- NLP using BERT (WIP) - working on binary classification over BERT, on two sentiment datasets namely - IMDB Large Movie Review Dataset and Sentiment140 dataset with 1.6 million tweets, in order to test supervised learning in the absence of trainable datasets (specific to elections)
    - The clean version of the Sentiment140 dataset can be downloaded from *here*
- Twitter-data extraction (WIP) - extracting relevant tweets using Twitter Search API

4. Deploying the model

Once the model is ready (accepted after continuous and progressive analysis), we can deploy it using TensorFlow Serving.

For ease of deployment, Cortex is a helpful solution for deploying models without going through tons of documentation of each underlying tech-stack.

The steps to deploy a classifier model are described here in detail, a summary of which is:

- Make sure that Docker is installed in your system
    - If you are working with a Windows machine, you need to **Switch to Linux Containers** from the Settings menu
- The notebook to train and deploy the model is in this Google Colab notebook. It uploads the saved model into a folder of our S3 bucket (which needs to be present there before executing this notebook)
    - The model artifacts will be saved inside an integer-labeled folder for tracking the versions of the saved model
- AWS Credentials and Cluster configurations need to be done before installation

    - This will create resources in your AWS account which aren't included in the free tier, e.g. an EKS cluster, two Elastic Load Balancers, and EC2 instances (quantity and type)
- From Cortex's top-level directory, create a YAML file (e.g. cortex.yaml) for deployment and a request handler file (e.g. handler.py)
    - Request handlers are python files that can contain a pre_inference function and a post_inference function.
- Execute (using CLI):

```
$ ./cortex.sh install
```

- Once Cortex is ready, we can check the operator and API endpoints can be viewed by executing:

```
$ ./cortex.sh info
```

- Make sure that all the required packages are present in the instance.
    - Cortex looks for a requirements.txt file in the top-level cortex directory (in the same level as cortex.yaml)
- Once everything is in place, execute this command to start deployment:

```
$ cortex deploy
```

- We can monitor the status of the deployment by executing:

```
$ cortex get --watch
```

  The status looks something like *this*.

- The API-name is **`classifier`** here and if the deployment was successful, the API endpoint to query the model can be retrieved by executing:

```
$ cortex get classifier
```

- To predict values on runtime, we can simply raise a POST request on the endpoint URL generated above:
    - Note: the `"polarity"` parameter should be the same as mentioned in the handler.py file while saving the model

```
$ curl <API-endpoint-url> -X POST -H "Content-Type: application/json" -d '{"polarity": "<Sentence or phrase to test the polarity of>"}'

"positive"
```