DELIVERING MULTI - LABEL, MULTI-CLASS CLASSIFICATION NATURAL LANGUAGE PROGRAMMING MODELS

Project Goal:

After concluding a project that enabled me to complete a neural translation machine and also implement a word2Vec neural network, I decided to deepen my Natural Language programming skills as part of my Springboard Capstone Project.

I decided to do a relatively complicated task - a multi-label, multi-class classification - which I would implement using a series of NLP models.

Three models were selected:

1. TF-IDF model
2. LSTM (Long Short Term Memory) model
3. BERT - (Bidirectional Encoder Representations from Transformers) model

Because I was focused on gaining knowledge in deploying a variety of NLP models, I decided to get a relatively easy to manipulate but moderately complicated dataset. The dataset I chose was the Kaggle/Jigsaw/Conversational AI Toxic Comments Challenge dataset which is a collection of wikipedia comments labelled toxic by human raters.

After successfully concluding the development of the models to an average accuracy of 99% accuracy, the next step was the deployment of a flask based api.

Next Steps: After this the development of a web portal to enable users key in fresh text and confirm if it is toxic or not would be a great next step

TF IDF Model:

The following steps were concluded as part of the development of the TF IDF model:

1. Data Exploration:
    a. The distribution of the classes was reviewed in order to confirm if there was any imbalance in the classes.
    b. The correlation between the classes was also reviewed. While there was some correlation (to be expected in a multi-label problem), it was not above 0.7 and mostly under 0.4.
    c. The distribution of word length per class
2. Data Cleaning
    a. Removal of HTML tags, punctuations and non-alphabets
    b. Removal of stop word
3. Data Preparation
    a. Stemming and Tokenization of the word
4. Modeling
    a. TF IDF Vectorization
    b. One Vs Rest Classification using a logistic regression classifier

The beauty of the TF IDF model was that it was relatively accurate whilst simple to use and fast.

The model was implemented using the Scikit learn library and the accuracy was 99% on the average

LSTM Model:

The following steps were taken as part of this model development

1. Library selection: The choice was between Pytorch and Tensorflow. I chose Tensorflow since I had previously implemented some deep learning models in Pytorch and I wanted to use Tensorflow to get an understanding of the library and its relationship with keras.
2. Data Preparation: This involved mostly removal of stop words and tokenization
3. Model Development: I selected a single LSTM layer at first and thereafter implemented a 2 layer LSTM model. The model also had a dropout layer to reduce overfitting
4. Model Training: Model was trained on a subset of the data
5. The accuracy was less than that of the TFIDF model at 98.6%

BERT MODEL:

The following steps were taken as part of this model development

1. Research to understand transformer models as a whole and BERT in particular
2. Coding to alter the existing BERT material to suit a multi-class, multi-label classification problem
3. Data Preparation: This was mostly tokenization which is relatively specialized for the BERT model
4. Model Training: This was relatively straight forward since we used an uncased pre-trained model
5. Model Evaluation. Model accuracy was 98.9%
6. Model was then saved to an HDF5 model to enable it to be ported to a web portal or deployed via Tensorflow lite to a model device if required.

Next steps:
Try to deploy BERT model to mobile android device

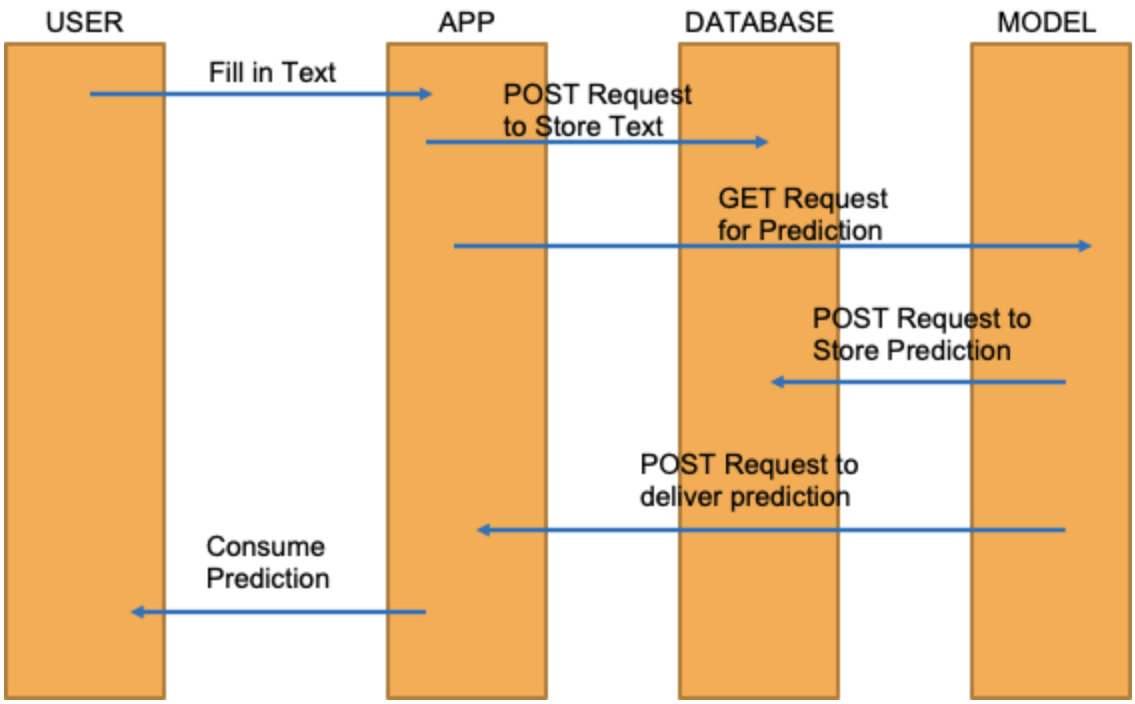
Project Considerations

- Major Components of the System
  The major components of my system are the models and the api. In a perfect world, I would also put up a web interface.

- System Input and Output
  The input is the text input from the user. The out is the prediction

- Data Storage
  In a perfect world where there is a web interface or app and customers are using the interface or app daily, then the data would be stored in a database. In this case, there is no need for data storage

- How will data get from one component of the system to another? Data would move from database to model and from model to app via api calls implemented using flask



- Model LifCycle
    - How frequently do you need to retrain your model? Is it at fixed intervals when you collect a certain amount of new data or when some other conditions are met?

      If the model was to be implemented in real life, one of the key aspects would be the ability of users to confirm/ identify whether a comment is toxic or not. This would provide new data for training but more importantly, data that can be used to monitor the model and determine drift or loss of accuracy.

      The model would be retrained with the new data weekly or monthly depending on the amount of data collected and the number of predictions required

    - What kind of data do you need for retraining? How will you store and manage it?

      As detailed adobe, labelled data obtained from user confirming the validity of predictions would be used for retraining

      This data would be stored in a database and retrieved from the database when needed.

    - How do you know if the retrained model is good enough to deploy?

      In a idea word, after retraining and off line evaluations of the model, there should be an A/B testing of the new retrained model to ascertain that it is better than the existing model

- How will the retrained model be deployed? ○ How will the retrained model be stored as an artifact?

  The retrained model would be stored the same way the current model is stored - as a joblib model, stored in a docker container which would be run within a virtual environment

- Model Monitoring
  There would be a feedback loop whereby users confirm the validity of the predictions. This validity would be stored in the database. Every day, the accuracy and F1 scores would be calculated to confirm that the model is still maintaining its accuracy.

  Also it should be monitored for bugs and other performance issues.

- Response to Outages

  The response to both GET and POST requests would have a code for out of service that would enable users to know when the system is out. It is a simple system and its points of potential failure are not many.

- System Tools?

  Tools and technology used include - python script, environment yml, dockerfile, flask app

- What is the estimated implementation cost in terms of resources, time, and money as applicable?

  Implementation cost. It took me
- 40 hours for initial code and test
- 20 hours for cleaning up the code
- 40 hours in the future for setting up the web portal, flask and docker file