CS-410 Text Information Systems

PROJECT DOCUMENTATION

Sandeep Nanjegowda – sgn3 (Captain)

Sunitha Vijayanarayan - sunitha3

Valentina Mondal – vmondal2

INDEX

1. Overview

Our goal is to classify the given test set data of twitter responses as “SARCASM” or “NOT SARCASM” by using various classification methods. Have classified the test data using Linear Regression, Naïve Bayes, GRU, CNN, LSTM, SVM, Bidirectional and BERT models. While predicting the label of the “response” based on the “context”. Context which is an ordered list of dialogue for which response is a reply to the last dialogue in the context. Test data has unique ids along with tweet “responses” to be classified. We build, train several different models using given trained data and predict the test data using the trained model and generate an "answer.txt” file that has unique ids along with the predicted label. Since there are 1800 ids in test dataset, our generated “answer.txt” file has exactly 1800 rows.

At a high level, we perform the below steps:

- Preprocess both the training and test data by removing the html tags, converting the tweets to lower case, removing punctuations and numbers, removing stop words, converting emojis and emoticons, removing single character and multiple spaces, removing left over special characters.
- Split the test dataset into train and test data.
- Create the embedding matrix using glove or word2vec embeddings.
- Create the model using embedding layer, adding layers such as LSTM, CNN etc.
- Fit the model on train data
- Predict the model on test data and generating predicted labels

Once the answer.txt file is generated and uploaded to git hub, predicted label values are compared with the actual result set to obtain the precision, recall and F-score value.

Precision indicates fraction of relevant instances among the retrieved instances. Precision is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly. Recall indicates fraction of the total amount of relevant instances that were retrieved. Recall is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive. F-score is the harmonic mean of the precision and recall. The highest possible value of an F-score is 1. Our baseline to be achieved is 0.723

We were successfully able to beat the baseline using Logistic Regression and BERT model. We will be walking through further in below points regarding implementation and usage for all the models we tried.

The models that we tried can be used to classify any tweets that has response and context attributes. The models can be leveraged for other applications with appropriate changes such as classification of tags in news headline, sentiment analysis for movie reviews etc.

2. Implementation of Models for Tweet Classification

This section describes implementation of different Tweet Classification Models

2.1 BERT

We have used BERT (Bidirectional Encoder Representations from Transformers) model BertForSequenceClassification for Tweet classification. Model is built using Jupyter Notebook. Jupyter Notebook has following Sections in same order. Chris McCormick Blogs and code about BERT were very helpful for using BERT for classification. We have used certain sections of this code from (***McCormick***) in our Model.

2.1.1 Preprocessing of Data

Two data files were provided – train.jsonl and test.jsonl . Training and test data are read into Pandas data frame. Python functions ***load_training_data_to_pandas*** and ***load_test_data_to_pandas*** load training and test data to panda's data frame.

URLs, @, non-ASCII characters, extra space, &, <, > are removed, space is inserted between punctuation marks. All characters are converted to lower case for both training and test dataset. Labels in training are converted to 1 and 0 (1 for Sarcasm and 0 for Not Sarcasm).

2.1.2 Model

We have used transformers package from Hugging Face which will give us a PyTorch interface for working with BERT. We have installed transformers package.

2.1.3 Tokenizing Input and creating DataLoaders

To feed our text to BERT, it must be split into tokens, we need to add special tokens to start and end, Pad and Truncate all sentences to a single constant length (we have used max length as 256), differentiate real tokens from padding tokens with attention mask. and then these tokens must be mapped to their index in the tokenizer vocabulary. The tokenization must be performed by the tokenizer included with BERT. We have used “uncased” version - “***bert-base-uncased***".

Training dataset is split for training and validation. We will also create an iterator for our training, validation and test dataset using the torch DataLoader class. This helps save on memory during training because, unlike a for loop, with an iterator the entire dataset does not need to be loaded into memory. Note: We have used ***SequentialSampler*** for validation and test datasets.

2.1.4 Pre-Trained BERT Model

After this we have loaded Pre-Trained BERT model - BertForSequenceClassification with a single linear classification layer on top.

2.1.5 Model Parameters and Learning Rate Scheduler

We have used AdamW Optimizer with learning rate of 2e-5 and eps of 1e-8 and

get_linear_schedule_with_warmup is created out of the Optimizer.

2.1.6 Utility Functions

Utility function ***format_time*** is created for formatting time and utility function ***flat_accuracy*** is created for calculating accuracy.

2.1.7 Training Loop

Each pass in our loop we have a training phase and a validation phase

Training**:**

- Unpack our data inputs and labels
- Load data onto the GPU for acceleration
- Clear out the gradients calculated in the previous pass.
    - In PyTorch the gradients accumulate by default (useful for things like RNNs) unless you explicitly clear them out.
- Forward pass (feed input data through the network)
- Backward pass (backpropagation)
- Tell the network to update parameters with optimizer.step()
- Track variables for monitoring progress

Evalution**:**

- Unpack our data inputs and labels
- Load data onto the GPU for acceleration
- Forward pass (feed input data through the network)
- Compute loss on our validation data and track variables for monitoring progress

2.1.8 Evaluating Test Data

Prediction is done on test dataset. Test dataset is read in batches and prediction is appended to list, which is then added to test panda's data frame and answer_BERT.txt is created.

2.2 SVM

We have used Support Vector Machines model for Tweet classification. Code is written in python language. We have used certain sections of code from (***Bronchal***) in our model:

2.2.1 Preprocessing of Data

Two data files were provided – train.jsonl and test.jsonl . Training and test data are read into Pandas data frame. Python functions ***load_training_data_to_pandas*** and ***load_test_data_to_pandas*** load training and test data to panda's data frame.

HTML tags, punctuation, numbers, single character, multiple spaces, stop words, left over special characters are removed. All characters are converted to lower case for both training and test dataset. Labels in training are converted to 1 and 0 (1 for Sarcasm and 0 for Not Sarcasm).

2.2.2 Tokenizing Input

Twitter-aware tokenizer designed to be flexible and easy to adapt to new domains and tasks. Tuple regex_strings define a list of regular expression strings. regex_strings strings are put, in order, into a compiled regular expression object called word_re. The tokenization is done by word_re.findall(s), where s is the user-supplied string, inside the tokenize() method of the class Tokenizer. When instantiating Tokenizer objects, there is a single option: preserve_case. By default, it is set to True. If it is set to False, then the tokenizer will downcase everything except for emoticons.

2.2.3 Vectorize

CountVectorizer converts a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix. The following parameters are used:

Analyzer: feature is made of word n-grams.

Tokenizer: override the string tokenization step while preserving the preprocessing and n-grams generation steps

Lowercase: convert all characters to lowercase before tokenizing.

Ngram_range: The lower and upper boundary of the range of n-values for different word n-grams to be extracted i.e. (1,1)

Stop_words: a built-in stop word list for English is used

2.2.4 Cross validation and grid search

We use cross validation and grid search to find good hyperparameters for our SVM model. We build a pipeline to get features from the validation folds when building each training model. GridSearchCV implements a “fit” and “predict” method. Snapshot of code below for reference. It exhaustively searches over specified parameter values for an estimator. param_grid enables searching over any sequence of parameter settings. cv determines the cross-validation splitting strategy. n_jobs as –1 means to use all processors in parallel. Verbose controls the verbosity of messages. Scoring is to evaluate the predictions on the test set.

```
kfolds = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

np.random.seed(1)

pipeline_svm = make_pipeline(vectorizer,SVC(probability=True, kernel="linear", class_weight="balanced"))

grid_svm = GridSearchCV(pipeline_svm,
                    param_grid = {'svc__C': [0.01, 0.1, 1]},
                    cv = kfolds,
                    scoring="roc_auc",
                    verbose=1,
                    n_jobs=-1)

grid_svm.fit(X_train, y_train)
```

2.2.5 Evaluating Test Data

Model with the best hyperparameters works on test data and basis the prediction, labels generated with value greater than 0.5 as “SARCASM” else “NOT_SARCASM” is appended to test panda's data frame and answer_SVM.txt is created.

2.3 CNN

We have used Convolutional neural network model for Tweet classification. Code is written in python language. We have used certain sections of this code from (***Celeni***) in our model:

2.3.1 Preprocessing of Data

For preprocessing of twitter test and train data, same function that was used in BERT model as described in section 2.2.1 has been also used for CNN model.

2.3.2 Word embedding

Word embedding is vector representation of a particular word. Weight matrix is created from word2vec gensim model. And then embedding vectors are obtained from word2vec and using it as weights of non-trainable keras embedding layer. Corpus for twitter data for word2vec named as “3000tweets_notbinary” was referred from (***Celeni***)

We also build the embeddings using Global Vectors for Word Representation. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. Corpus can be downloaded from the link (***Paletto***) that we used as embedding file in our code to build the weight matrix.

2.3.3 Train Embedding Model

Depending upon whichever embedding – word2vec or glove, we are using we train the model by adding non trainable embedding layer followed by the addition of CNN layer that creates a convolution kernel that is convoluted with the layer input over a single spatial dimension to produce a tensor of outputs. The action ReLU is applied to outputs as well. We set the number of filters to the dimensionality of the output space. Kernel_size is to specify the length of the 1D convolution window. Further we maxpool that summarize the most activated presence of a feature. Pooling is required to down sample the detection of features in feature maps. We also use global pooling that that down sample the entire feature map to a single value. This is same as setting the pool_size to the size of the input feature map.

We then use regularization method “dropout” that approximates training many neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. Also, we then add dense layer to perform a linear operation on the layer’s input vector.

Code snippet below:

```
model.add(Conv1D(num_filters, 7, activation='relu', padding='same'))
model.add(MaxPooling1D(2))
model.add(Dropout(0.5))
model.add(Conv1D(num_filters, 7, activation='relu', padding='same'))
model.add(GlobalMaxPooling1D())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

2.3.4 Early stopping

Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold-out validation dataset.

2.3.5 Evaluating Test Data

Once the model is created, model.predict() is run to predict for test data where prediction with value greater than 0.5 is labelled as “SARCASM” else “NOT_SARCASM”. Label is appended to the test panda's data frame and answer_CNN.txt is created.

2.4 LSTM

We have used Long Short-Term Memory network model for Tweet classification. Code is written in python language. We have used certain sections of this code in our model from (*nana*).

2.4.1 Preprocessing of Data

For preprocessing of twitter test and train data, same function that was used in BERT model as described in section 2.2.1 has been also used for LSTM model.

2.4.2 Word embedding

Same functional code as discussed in Section 2.3.2 is used for LSTM model.

2.4.3 Train Embedding Model

Depending upon whichever embedding – word2vec or glove, we are using we train the model by adding non trainable embedding layer followed by the addition of LSTM layers with units specifying the dimensionality of the output space.

We use regularization method “dropout” that approximates training many neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. Also, we then add dense layer to perform a linear operation on the layer’s input vector. We compile the model using the binary cross-entropy loss function since it predicts a binary value and opt optimizer. The hyperparameters were tuned experimentally over several runs.

Code snippet below:

```
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)
model.add(LSTM(128,return_sequences=True))
model.add(Dropout(0.6))
model.add(LSTM(128))
model.add(Dropout(0.7))
model.add(Dense(1, activation='sigmoid'))
opt = SGD(lr=0.01)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['acc'])
```

2.4.4 Early stopping

Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold-out validation dataset.

2.4.5 Evaluating Test Data

Once the model is created, model.predict() is done to do the prediction for test data where prediction with value greater than 0.5 is labelled as “SARCASM” else “NOT_SARCASM”. Label is appended to the test panda's data frame and answer_LSTM.txt is created.

2.5 GRU

We have tried GRU (Gated Recurrent Unit) model for Tweet classification. Code is written in python language. GRU is a special type of Recurrent Neural network. This type of sequence model can retain information from long ago, without washing it through time or remove information which is irrelevant to the prediction. Some of the code & hyper-parameter values specifically those related to ReduceLROnPlateau were decided based on the blog by (***Kohli***)

2.5.1 Preprocessing of Data

For preprocessing of twitter test and train data, same function that was used in BERT model as described in section 2.2.1 has been also used for GRU model.

2.5.2 Word embedding

Same functional code as discussed in Section 2.3.2 is used for GRU model.

2.5.3 Train Embedding Model

Depending upon whichever embedding – word2vec or glove, we are using we train the model by adding a trainable embedding layer followed by the addition of GRU layers with units specifying the dimensionality of the output space.

We use regularization method “dropout” that approximates training many neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. Also, we then add dense layer to perform

a linear operation on the layer's input vector. We compile the model using the binary cross-entropy loss function since it predicts a binary value and opt optimizer. The hyperparameters were tuned experimentally over several runs.

Code snippet below:

```
#GRU
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=True)
model.add(embedding_layer)
model.add(GRU(32,dropout=0.5, recurrent_dropout=0.2,return_sequences=True))
model.add(GlobalMaxPooling1D())
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
rlrp = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2)
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience =2)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
opt = Adam(learning_rate=0.01)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['acc'])
print(model.summary())
#model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['acc'])
print(model.summary())
```

2.5.4 ReduceLROnPlateau

This option was used so that the learning rate to be reduced when training is not progressing.

2.5.5 Early stopping

Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold-out validation dataset.

2.5.5 Model Checkpointing

Model checkpointing was used so that the best training model based on hold-out set validation would be saved and the saved model used to evaluate test data.

2.5.6 Evaluating Test Data

Once the model is created, model.predict() is done to do the prediction for test data where prediction with value greater than 0.5 is labelled as "SARCASM" else "NOT_SARCASM". Label is appended to the test panda's data frame and answer_GRU.txt is created.

2.6 Naïve Bayes and Linear Regression

Naïve Bayes and Logistic Regression Models are built using Jupyter Notebook for Tweet Classification. Jupyter Notebook has following Sections in same order. Bert Carremans Blog and code about Naïve Bayes and Linear Regression were very helpful. We have used certain sections of code in our Model from the reference (*Carremans*)

2.6.1 Preprocessing of Data

Two data files were provided – train.jsonl and test.jsonl . Training and test data are read into Pandas data frame. Python functions load_training_data_to_pandas and load_test_data_to_pandas load training and test data to panda's data frame.

We have Python class ***TextCounts*** , it extends sklearn.base.BaseEstimator and sklearn.base.TransformerMixin , this class extracts additional features like word counts, hash tags, mentions, capital words, question marks, urls and emojis.

We have Python class ***CleanText*** it extends sklearn.base.BaseEstimator and sklearn.base.TransformerMixin , this class removes mentions, urls, oneword, punctuations, digits, Stopwords , performs stemming and converts to lower case.

Test and Training data is passed through this classes fit methods. Extra features extracted for training data is combined with Cleaned data for both Training and Test data. Steps for Test data is done after training.

We have Python class ***ColumnExtractor*** it extends sklearn.base.BaseEstimator and sklearn.base.TransformerMixin , this class is used for selecting columns in test and training dataset.

2.6.2 Hyperparameter tuning and cross-validation

We first declare parameters for grid search. We have set parameters_vect , parameters_mnb and parameters_logreg parameters. We have parameters for TF-IDF like max_df , min_df , ngram_range , parameters for Linear Regression like clf_c and clf_penality , parameters for Naiive bayes like alpha. we have function grid_vect1 which does grid search. This function uses skilearn pipeline and it is based on below code from the reference (***Scikit Learn***) .

2.6.3 Model Training and Prediction

Naiive Bayes and Logistic Regression models are created and passed to grid_vect1 which vectorizes the data using TF-IDF in ski learn pipeline and does grid search for best Hyper parameters.

Once we find the best hyperparameters, we fit both Naïve bayes and Logistic regression models with best parameters and perform prediction on test data. Predicted data is added to pandas data frame as new column and answer.txt is created for both models.

2.7 Bi-directional Models (GRU & LSTM)

We have tried bi-directional versions of both GRU (Gated Recurrent Unit) model & LSTM model for Tweet classification. Code is written in python language. Bidirectional models are an extension of traditional LSTM & GRU models that can improve model performance on sequence classification problems. In problems where all timesteps of the input sequence are available, Bidirectional models train two instead of one LSTMs on the input sequence. Some of the code was referenced from (***LillySimeonova***)

2.7.1 Preprocessing of Data

For preprocessing of twitter test and train data, same function that was used in BERT model as described in section 2.2.1 has been also used for these models as well.

In addition, we also combined the additional text counts features calculated in the LogisticRegression model and combined them with the text features in a bid to improve model performance on the test set.

2.7.2 Word embedding

Same functional code as discussed in Section 2.3.2 is used for Bidirectional LSTM model & Bidirectional GRU.

2.7.3 Train Embedding Model

Depending upon whichever embedding – word2vec or glove, we are using we train the model by adding a trainable embedding layer followed by the addition of GRU/LSTM layers with units specifying the dimensionality of the output space. We are then combining text features with count features using multiple input models. All the other layers are added exactly like GRU/LSTM models already discussed in section 2.5 & 2.4. Combining of text features with non-text was coded by referring (***Freischlag***)

We compile the model using the binary cross-entropy loss function since it predicts a binary value and Adam optimizer. The hyperparameters were tuned experimentally over several runs.

Code snippet below:

```
nlp_input1 = Input(shape=(maxlen,))
meta_input1 = Input(shape=(7,))
#emb = Embedding(output_dim=embedding_size, input_dim=100, input_length=seq_length)
emb1 = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)(nlp_input1)
#lstm = LSTM(300, dropout=0.3, recurrent_dropout=0.3)(embed)
#nlp_out2 = Bidirectional(LSTM(128))(meta_input1)
x = Dense(8, activation="relu")(meta_input1)


nlp_out1 = Bidirectional(GRU(128))(emb1)
concat1 = concatenate([nlp_out1, x])
drop1 = Dropout(0.6)(concat1)
#dens = Dense(1)(drop)

#classifier = Dense(32, activation='relu')(drop)
output1 = Dense(1, activation='sigmoid')(drop1)
model_bgru = Model(inputs=[nlp_input1 , meta_input1], outputs=[output1])
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
model_bgru.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(model_bgru.summary())
```

2.7.4 Early stopping

Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold-out validation dataset.

2.7.5 Model Checkpointing

Model checkpointing was used so that the best training model based on hold-out set validation would be saved and the saved model used to evaluate test data.

2.7.6 Evaluating Test Data

Once the model is created, model.predict() is done to do the prediction for test data where prediction with value greater than 0.5 is labelled as “SARCASM” else “NOT_SARCASM”. Label is appended to the test panda's data frame and answer_Bidirectional.txt is created.

3. Tweet Classification Models Usage

This Section describes steps for running different Tweet Classification Models which are described in Section 2.

3.1 BERT

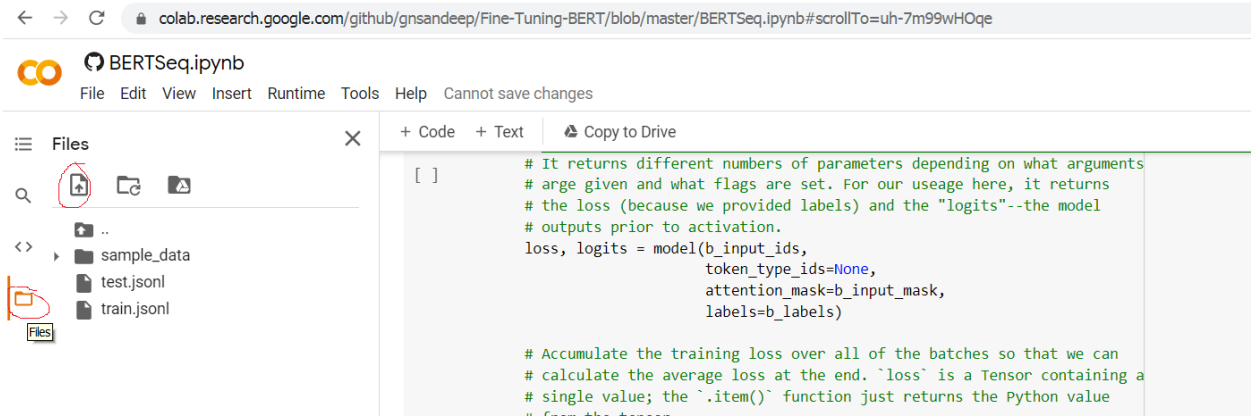
Google Colab is preferred for running BERT model. We have used Colab for training and evaluating test dataset.

3.1.1 Google Colab

Open Colab in browser (Mozilla, Chrome) https://colab.research.google.com , choose GitHub and use our Project GitHub URL and chose BERTSeq.ipynb

3.1.2 Load Training and Test data to google Colab

Click on Files and Click on upload button and upload both test and train jsonl files.



3.1.3 Run

Click on Runtime and click on Run All to Run all the Cells.

3.1.4 Result

Predicted values are stored in answer_BERTSeq.txt, we can see the file in Files section Colab.

3.2 SVM

SVM model can be run locally. We have used python language.

3.2.1 Load Training and Test data

First place the test.json and train.json data files as provided in github link https://github.com/CS410Fall2020/ClassificationCompetition/tree/main/data to your local path under data folder.

3.2.2 SVM code

Load the SVM.py code in your local path

3.2.3 Prerequisites

Ensure to download and install the below libraries and modules to run the code to not throw any errors

```
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
import json

#nltk.download('stopwords')
from nltk.corpus import stopwords

from nltk.stem import SnowballStemmer,PorterStemmer
from nltk.tokenize import TweetTokenizer

from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score, f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix, roc_auc_score, recall_score, precision_score

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

3.2.4 Run

Run SVM.py code in your local.

3.2.5 Result

Predicted values are stored in your local path as ‘answer_SVM.txt’

3.3 CNN

CNN model can be run locally. We have python language.

3.3.1 Load Training and Test data

First place the test.json and train.json data files as provided in github link https://github.com/gnsandeep/CourseProject/tree/main/data to your local path under data folder.

3.3.2 CNN code

Load the CNN.py code in your local path

3.3.3 Prerequisites

Ensure to download and install the below libraries and modules to run the code without throwing any errors

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
import json

from numpy import array
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Masking, Input, Dense, Embedding, Conv2D, MaxPool2D, Conv1D, MaxPooling1D, GlobalMaxPooling1D
from keras import regularizers
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.optimizers import SGD
import seaborn as sns
from numpy import array
from numpy import asarray
from numpy import zeros

#nltk.download()
from nltk.stem.porter import PorterStemmer
#from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer

import gensim
import string
```

3.3.4 Run

Run CNN.py code in your local.

3.3.5 Result

Predicted values are stored in your local path as ‘answer_CNN.txt’

3.4 LSTM

LSTM model can be run locally. We have used python language.

3.4.1 Load Training and Test data

First place the test.json and train.json data files as provided in github link https://github.com/gnsandeep/CourseProject/tree/main/data to your local path under data folder.

3.4.2 LSTM code

Jupyter notebook LSTM.ipynb .

3.4.3 Prerequisites

Please download Glove embedding file glove.6B.100d.txt (it is available under glove.6B.zip) and place it in data folder. We can download Glove embedding file from

https://nlp.stanford.edu/projects/glove/

Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip

3.4.4 Run

Open the Jupyter notebook LSTM.ipynb and click Kernel Restart and Run All.

3.4.5 Result

Predicted values are stored in your local path as 'answer_LSTM.txt'

3.5 GRU

GRU model can be run locally. We have used python language.

3.5.1 Load Training and Test data

First place the test.json and train.json data files as provided in github link https://github.com/gnsandeep/CourseProject/tree/main/data to your local path under data folder.

3.5.2 GRU code

Load the GRU.py code in your local path

3.5.3 Prerequisites

Ensure to download and install the below libraries and modules to run the code without throwing any errors

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
import json
import sklearn
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, LSTM,Dropout , Input, GRU
from keras.layers import GlobalMaxPooling1D
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.optimizers import SGD
import seaborn as sns
from numpy import array
from numpy import asarray
from numpy import zeros
from nltk.stem import PorterStemmer
from sklearn.metrics import confusion_matrix
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
```

3.5.4 Run

Run GRU.py code in your local.

3.5.5 Result

Predicted values are stored in your local path as ‘answer_GRU.txt’

3.6 Naïve Bayes and Logistic Regression

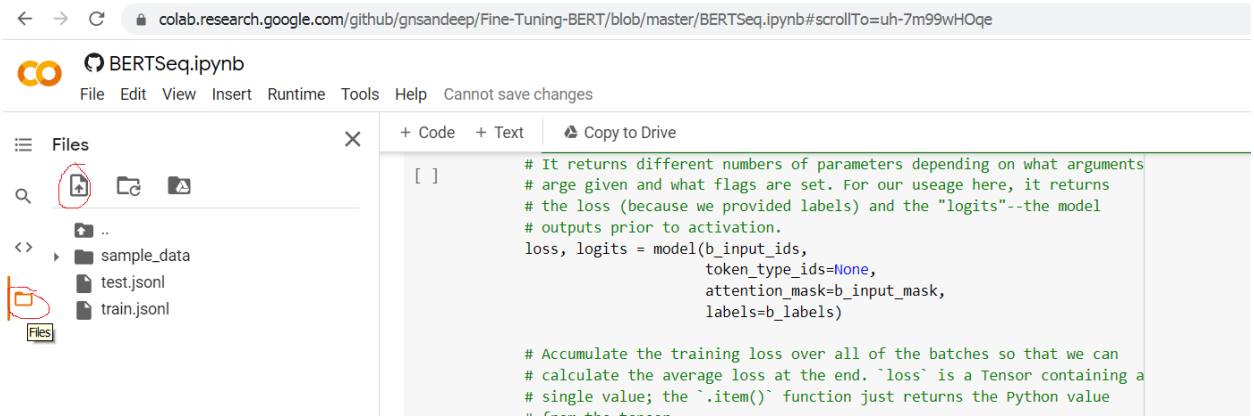
Google Colab is preferred for running Naïve Bayes and Logistic Regression model. We have used Colab for training and evaluating test dataset.

3.6.1 Google Colab

Open Colab in browser (Mozilla, Chrome) https://colab.research.google.com , choose GitHub and use our Project GitHub URL and chose LGNB_CV.ipynb

3.6.2 Load Training and Test data to google Colab

Click on Files and Click on upload button and upload both test and train jsonl files from https://github.com/gnsandeep/CourseProject/tree/main/data.



3.6.3 Run

Click on Runtime and click on Run All to Run all the Cells.

3.6.4 Result

Predicted values are stored in answerNB.txt and answerCVLG.txt, we can see the file in Files section Colab.

3.7 Bi-directional Models (GRU & LSTM)

Bi-directional models can be run locally. We have used python language.

3.7.1 Load Training and Test data

First place the test.json and train.json data files as provided in github link https://github.com/gnsandeep/CourseProject/tree/main/data to your local path under data folder.

3.7.2 Bi-directional model code

Load the BidirectionalModels.py code in your local path

3.7.3 Prerequisites

Ensure to download and install the below libraries and modules to run the code without throwing any errors

```
import pandas as pd
import numpy as np
import tensorflow as tf
import re
import nltk
from nltk.corpus import stopwords
import string
import os
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from keras.models import Model
from sklearn.metrics import precision_recall_fscore_support
import json

from sklearn.metrics import confusion_matrix
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten, LSTM,Dropout , Input , Bidirectional , concatenate, GRU
from keras.layers import GlobalMaxPooling1D
from keras.layers.embeddings import Embedding
#from tensorflow.keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.optimizers import SGD, Adam
import seaborn as sns
from numpy import array
from numpy import asarray
from numpy import zeros
from sklearn.base import BaseEstimator, TransformerMixin
import emoji
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

3.7.4 Run

Run BidirectionalModels.py code in your local.

3.7.5 Result

Predicted values are stored in your local path as ‘answer_blstm.txt’ & answer_bgru.txt

4. Results

Below are some of the results that were captured in leaderboard live data lab for individual model:

Model	Precision	Recall	F1
Base Line	0.723	0.723	0.723
BERT	0.7181913774973712	0.7588888888888888	0.7379794705564559
Logistic Regression	0.6593406593406593	0.8	0.7228915662650602
Naïve Bayes	0.6177215189873417	0.8133333333333334	0.702158273381295
SVM	0.6225716928769658	0.7477777777777778	0.6794548207975769
CNN	0.6376811594202898	0.6844444444444444	0.660235798499464
LSTM	0.620242214532872	0.7966666666666666	0.6974708171206226
GRU	0.5968109339407744	0.8733333333333333	0.7090663058186738
Bidirectional- GRU	0.6535819430814525	0.74	0.69411151641479
Bidirectional -LSTM	0.639927073837739	0.78	0.703054581872809

Screen shot for BERT

33	bzhao	38	0.6452145214521452	0.8688888888888889	0.740530303030303	1
34	shr23	49	0.6879770992366412	0.8011111111111111	0.7402464065708418	1
35	gnsandeep	34	0.7181913774973712	0.7588888888888888	0.7379794705564559	1
36	yeowlong	15	0.6241352805534205	0.9022222222222223	0.7378464334393458	1

5. Collaboration

Brief description of contribution of each team member in case of a multi-person team.

We had frequent meetings, we discussed about the models we had learnt, built, tested and results. We also did code reviews, incorporated suggestions.

Sandeep:

1. Build and test of LSTM on word embeddings using glove.
2. Build and test of Naïve Bayes and Logistic Regression.
3. Build and test of BERT.
4. Documentation for BERT, Naïve Bayes and Logistic Regression models.
5. Bidirectional LSTM along with combining Text & Non-Text features.
6. Voice over project presentation and demo.
7. Investigation & Test of Bi-directional LSTM model with multiple input features.

Sunitha:

1. Build and test of GRU on word embeddings using word2vec and glove.
2. Investigation & Test of Bi-directional GRU model with multiple input features.
3. Implementation for additional preprocessing steps to convert emojis and emoticons to text.
4. Implementation of ModelCheckpoint ,ReduceLROnPlateau.

5. Test effects of additional emoji pre-processing, EarlyStopping, ModelCheckpointing & ReduceLROnPlateau on GRU, LSTM & CNN models.
6. Documentation for GRU & Bi-directional Models.

Valentina:

1. Creating word vectors by word2vec method, create weight matrix from word2vec gensim model, getting embedding vectors from word2vec and using it as weights of non-trainable keras embedding layer.
2. Build and test of SVM model using cross validation and grid search to find good hyperparameters by building a pipeline.
3. Implementation of early stopping - Build network and train it until validation loss reduces.
4. Build and test of CNN on word embeddings using word2vec and glove.
5. Convert test and train features to InputFeatures that BERT understands, create model using pooled output, layer for tuning, dropout, labels conversion to one hot encoding; and get predictions.
6. Investigation & Test of LSTM model with word2vec, glove and different parameters.
7. Documentation for project report, documentation overview, SVM, CNN and LSTM models.

6. Presentation

Video Presentation : https://web.microsoftstream.com/video/4436fc73-53d1-4dc4-a386-29e7c8e20eca

Leaderboard Results : https://web.microsoftstream.com/video/7ae237ae-9c20-4a19-b4b7-b1b43508023f

References

[Online] / auth. Scikit Learn. - http://scikit-learn.org/stable/auto_examples/model_selection/grid_search_text_feature_extraction.html.

https://mccormickml.com/2019/07/22/BERT-fine-tuning/ [Online] / auth. McCormick.

Combining numerical and text features in deep neural networks [Online] / auth. Freischlag Christian. - https://towardsdatascience.com/combining-numerical-and-text-features-in-deep-neural-networks-e91f0237eea4.

https://github.com/ibrahimcelenli/cnn-word2vec-tweets-classification / auth. Celeni Ibrahim.

https://www.kaggle.com/jdpaletto/glove-global-vectors-for-word-representation [Online] / auth. Paletto J.D..

https://www.kaggle.com/lbronchal/sentiment-analysis-with-svm [Online] / auth. Bronchal Luis.

Sarcasm Detection using LSTM, GRU, (85% Accuracy) [Online] / auth. Kohli Nikhil. - https://www.kaggle.com/nikhilkohli/sarcasm-detection-using-lstm-gru-85-accuracy.

Sentiment Analysis in Python with keras and LSTM [Online] / auth. nana roblex. - https://www.kaggle.com/roblexnana/sentiment-analysis-with-keras-and-lstm.

Sentiment Analysis with Bidirectional LSTM [Online] / auth. LillySimeonova. - https://www.kaggle.com/liliasimeonova/sentiment-analysis-with-bidirectional-lstm.

Sentiment Analysis with Text Mining [Online] / auth. Carremans Bert. - https://towardsdatascience.com/sentiment-analysis-with-text-mining-13dd2b33de27.