CS 410 Text Information System Final Project

Code Documentation

Zhenchen Yu (zy23) Dec 11th, 2020

Contents

Project Description and Summary

The goal of this project is to use natural language processing and machine learning techniques to detect twitter sarcasm. In this project, I am going to try both traditional statistical learning classification models such as Support Vector Machine, Random Forest etc., and the state-of-the-art model BERT.
The results show that all the statistical learning models and simple neural networks have similar performance, but none of them beat the baseline. Both BERT base and BERT large **beat the baseline**. Due to the limit of time, I didn't further fine tune the model to improve the results. My best testing F1 score is 73.28%.

Leaderboard ID: 5f83d14b872c465d24df8b08

Rank	Username	Submission Number	precision	recall	f1	completed
30	zy23	41	0.6191570881226054	0.8977777777777778	0.7328798185941043	1

Part 1: Statistical Learning Models

Required Libraries

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
```

Preprocess Data

For labels, I replaced "SARCASM" with 1 and "NOT_SARCASM" with 0.
For response text, I used a CountVectorizer to vectorize the text, and removed the stop words. The stop words list was from MP2. Further I think @User are meaningless in the text so I removed those as well.
Finally, I split the dataset into 80% training dataset and 20% testing dataset.

Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a classifier that is defined by a separating hyperplane. The SVM algorithm generates an optimal hyperplane which categorizes new examples given labeled training data.
SVM is very powerful model that has been widely applied in the industry. Therefore, it is intuitive for me to try support vector machine to make the classification prediction.

SVM can use different kernels. In this project, I tried both linear kernel and gaussian (RBF) kernel. For hyperparameter tuning, I used a grid search method to find the best parameters.
For linear kernel, I tried value [0.01, 0.1, 1, 10, 100] for parameter C.
For RBF kernel, I tried value [0.01, 0.1, 1, 10, 100] for parameter C and [0.01, 0.1, 1, 10, 100] for gamma.

Random Forest

Random forest is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random forests randomly select m number of features at each tree node split, and therefore all the features get chances to be selected and the model prediction won't be dominated by some strong features. One of the great features of Random forests is that it corrects for decision trees' habit of overfitting to their training set.
For model parameters tuning, I set up the max_depth to 3, and searched for the best n_estimator from 10 to 100 step 2.

XGBoost

The full name of XGBoost is called eXtreme Gradient Boosting. XGBoost is a decision tree - based ensemble Machine Learning algorithm that uses a gradient boosting framework. XGBoost performs very well on processing unstructured data such as text. Compare to the general boost machine, XGBoost runs faster and has a high prediction accuracy in general.

I didn't tune parameters for this model. Instead, I just used the default setting.

Logistic Regression

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable, which is exactly the scenario we are facing.
For parameter tuning, I chose the 'liblinear' solver and tried the value [0.01, 0.1, 1, 10, 100] for parameter C.

Testing Results

Model	Training Validation F1 Score	Test Dataset Precision	Test Dataset Recall	Test Dataset F1
SVM Linear	76.02%	58.00%	65.22%	61.40%
SVM RBF	75.86%	50.03%	100%	66.69%
Random Forest	73.21%	56.73%	77.78%	65.60%
XGBoost	74.41%	58.38%	77.78%	66.70%
Logistic Regression	75.66%	58.96%	66.89%	62.68%

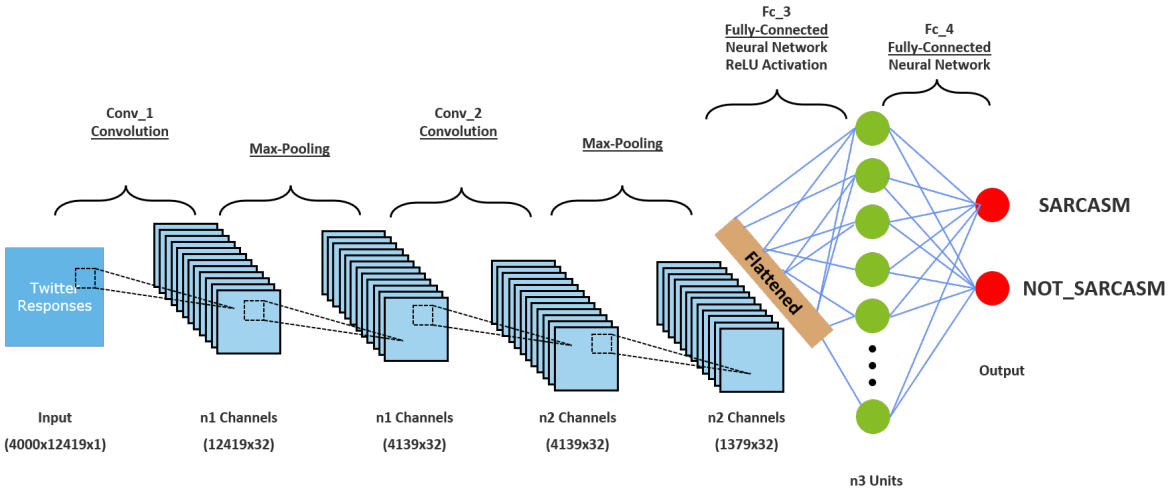
Part 2: Simple Neural Networks

Required Libraries

```
import random
import keras
from keras.layers import Dense, Dropout, LSTM,  BatchNormalization
from keras.layers import Activation, Flatten, Conv1D, MaxPooling1D
from keras.models import Sequential
```

Convolutional Neural Network (CNN)

Convolutional Neural Network is a kind of deep learning network, which is mostly used to classify images, but it can be used to solve the general classification problems.
In this project, I have designed a simple CNN, with only 2 layers of convolution, to make the prediction and set as a benchmark for deep learning models.
Below is an illustration of the model.

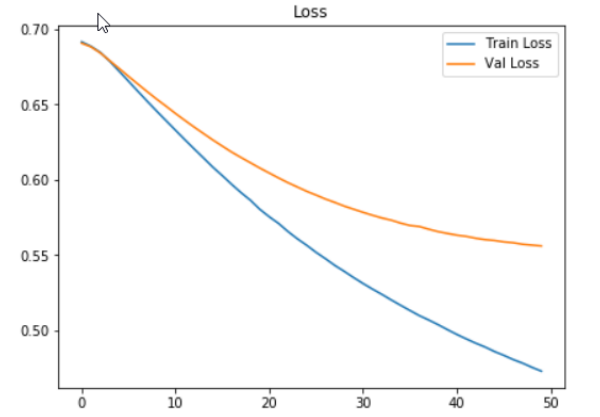
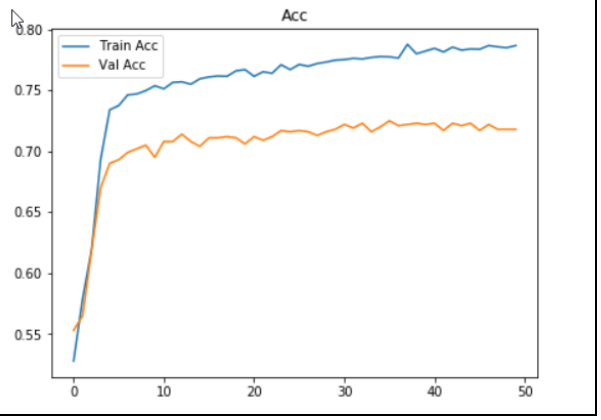


Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 12419, 32)	64
activation_1 (Activation)	(None, 12419, 32)	0
max_pooling1d_1 (MaxPooling1	(None, 4139, 32)	0
conv1d_2 (Conv1D)	(None, 4139, 32)	1056
activation_2 (Activation)	(None, 4139, 32)	0
max_pooling1d_2 (MaxPooling1	(None, 1379, 32)	0
flatten_1 (Flatten)	(None, 44128)	0
dense_1 (Dense)	(None, 32)	1412128
activation_3 (Activation)	(None, 32)	0
dense_2 (Dense)	(None, 1)	33

Total params: 1,413,281
Trainable params: 1,413,281
Non-trainable params: 0

Training

The input training dataset is a matrix of size 4000x12419x1. 4000 is the number of total documents, 12419 is the size of vocabulary and 1 is the number of channels.
For parameter tuning, I set the batch size to 64 and did 50 epochs of training. The lost function is set to be “binary_crossentropy”, and I used Adam optimizer with a learning rate of 1e-5, and epsilon to be 1e-6.

Training Loss	Training Accuracy
	

Testing Results

Model	Training Validation F1 Score	Test Dataset Precision	Test Dataset Recall	Test Dataset F1
CNN	73.55%	60.03%	75.78%	66.99%

Recurrent Neural Network (RNN) + LSTM

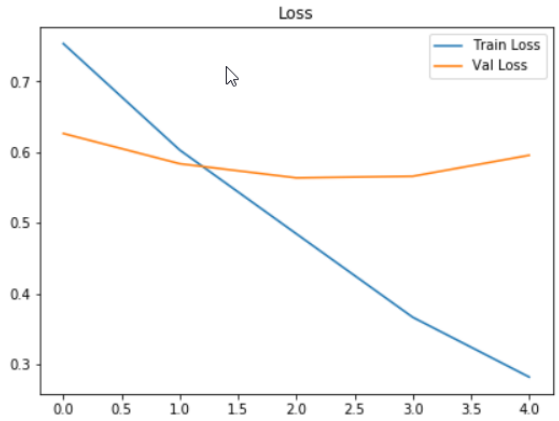
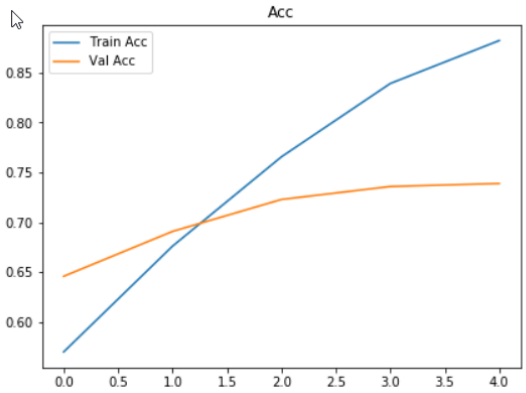
Recurrent Neural Network is another deep learning network. It is often used to work on tasks such as speech recognition, translation etc. Hence, it is an effective model to do text classification.
In this project, I also designed a simple RNN, with only 3 layers of LSTM. Below is an illustration of the model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 1, 256)            12980224
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 256)            0
_________________________________________________________________
batch_normalization_1 (Batch (None, 1, 256)            1024
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 256)            525312
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 256)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1, 256)            1024
_________________________________________________________________
lstm_3 (LSTM)                (None, 256)               525312
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_3 (Dense)              (None, 32)                8224
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 14,042,177
Trainable params: 14,040,641
Non-trainable params: 1,536
_________________________________________________________________
```

Training

The input training dataset is a matrix of size 4000x1x12419. 4000 is the number of total documents and 12419 is the size of vocabulary.

For parameter tuning, I set the batch size to 64 and did 5 epochs of training. The lost function is set to be "binary_crossentropy", and I used Adam optimizer with a learning rate of 1e-4, and decay to be 1e-6.

Training Loss	Training Accuracy
	

Testing Results

Model	Training Validation F1 Score	Test Dataset Precision	Test Dataset Recall	Test Dataset F1
RNN	75.02%	59.68%	78.44%	67.78%

Part 3: BERT

BERT (Bidirectional Encoder Representations from Transformers) is a deep learning language model that was created and introduced by researchers from Google in 2018. When released, it caused an earthquake in the NLP research areas because it beat eleven NLP tasks including GLUE (General Language Understanding Evaluation), SQuAD (Stanford Question Answering Dataset), SWAG (Situations With Adversarial Generations), and others.[1]

BERT is not out of nowhere. It absorbs the advantage of multiple previous state-of-the-art models, such as ELMo, OpenAI GPT and Transformer etc. BERT is powerful in that unlike the previous language models that are trained either from left to right or from right to left, BERT is trained bidirectional, which enables the model to learn better about the relationship between the words. In addition, BERT uses Transformer instead of LSTM or RNN, which are much slower, to train the large dataset that includes the 800M words BooksCorpus and 2,500M words English Wikipedia. BERT is also flexible in dealing with all kinds of tasks, thanks to its fine-tuning component.

There are two major components in BERT: Pre-training and Fine-Tuning. BERT uses feature-based approach for pre-training and use self-attention mechanism during the fine-tuning. For this project, we only need to do the fine-tuning part and directly use the pre-training BERT model.

Required Libraries

```
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from pytorch_pretrained_bert import BertTokenizer, BertConfig
```

[1] https://en.wikipedia.org/wiki/BERT_(language_model)

```
from pytorch_pretrained_bert import BertAdam, BertForSequenceClassification
from tqdm import tqdm, trange
import torch.optim as optim
```

Preprocess Data

For labels, I replaced “SARCASM” with 1 and “NOT_SARCASM” with 0.
For response text, I added special tokens [CLS] at the beginning and [SEP] at the end of each sentence for BERT to work properly. Then I used a Vectorizer to vectorize the text and removed the stop words. The stop words list was the same as the one used in the previous part.
There are some additional steps required to prepare the data for BERT. One is to set up the max length for the sequence. In BERT, the length of the sequence is fixed. If the text is shorter than the length, then we need to pad the rest of spaces. On the other hand, if it is longer than the length, we need to truncate the text. I checked the maximum length of all my responses and found the max sequence is 74. So I just set up the max_len to be 128.
What’s more, I need to create a mask for BERT to randomly mask some features in order to regularize the model.
Finally, I split the dataset into 80% training dataset and 20% testing dataset.

Training

For BERT training, I have tried both BertAdam and the normal Adam optimizer, and tried a few different learning rate and number of epochs. See details below.

Results

Model	optimizer	Learning Rate	Epoch	Threshold	Precision	Recall	F1
Bert Base	BertAdam	2.00E-05	4	0.5	69.93%	63.56%	66.59%
Bert Base	BertAdam	2.00E-05	4	0.7	67.71%	69.44%	68.57%
Bert Base	BertAdam	2.00E-05	4	0.8	66.11%	74.33%	69.98%
Bert Base	optim.Adam	1.00E-05	10	0.5	67.37%	64.00%	65.64%
Bert Base	optim.Adam	1.00E-05	10	0.8	64.61%	76.67%	70.12%
Bert Base	optim.Adam	5.00E-06	10	0.5	66.33%	66.56%	66.44%
Bert Base	optim.Adam	5.00E-06	10	0.8	63.98%	79.33%	70.83%
Bert Base	optim.Adam	5.00E-06	15	0.9	61.92%	89.78%	73.29%
Bert Base	optim.Adam	5.00E-06	15	0.5	65.30%	76.11%	70.29%
Bert Base	optim.Adam	1.00E-06	40	0.1	73.30%	45.44%	56.10%
Bert Base	optim.Adam	1.00E-06	40	0.99	51.05%	100.00%	67.59%
Bert Base	optim.Adam	1.00E-06	40	0.96	63.25%	84.89%	72.49%
Bert Large	optim.Adam	5.00E-06	50	0.8	66.35%	77.78%	71.61%
Bert Large	optim.Adam	5.00E-06	50	0.96	62.37%	89.89%	73.65%

Conclusion

In this project I have implements 5 different statistical learning models, 2 simple artificial neural network models and 2 state-of-the art BERT models. Using the test dataset, the statistical learning models can achieve F1 scores range from 61% to 66%, the artificial neural networks can achieve F1 score around 66% to 67% and BERT model can achieve an F1 score of 73% with the right parameter tuning, which beat the baseline F1 score.