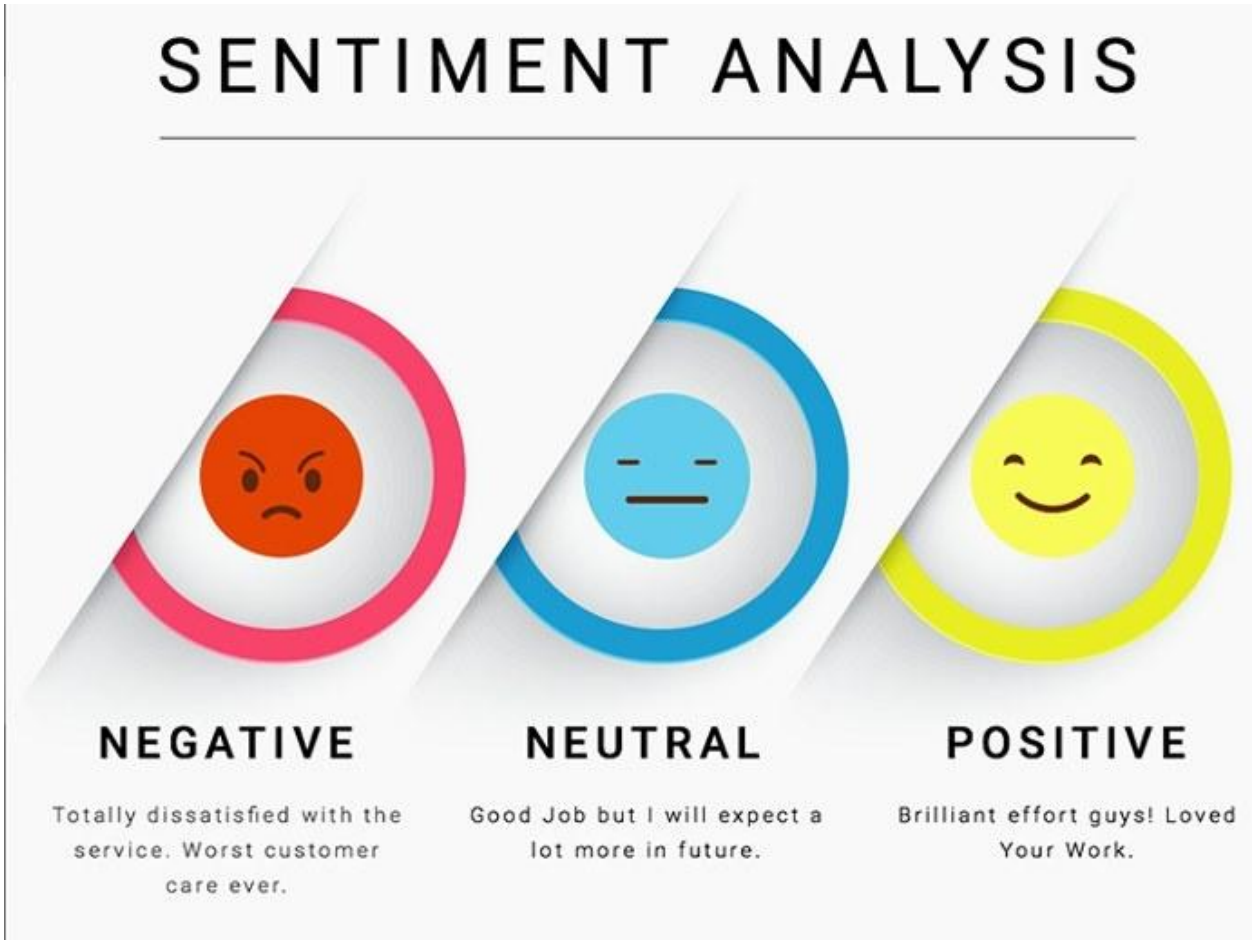


FINAL PROJECT REPORT ON 'SENTIMENT ANALYSIS THROUGH DEEP LEARNING'

APPLIED MACHINE LEARNING AND DATA SCIENCE 2020
COURSE CODE-002

19.07.2020

AKSHUNN TRIVEDI

Overview

Natural Language Processing has been a very important set of techniques which has risen by huge volumes in the recent decade,due to its huge commercial and cultural viability in processing natural languages and abstracting meaningful and insightful inferences which can be used to solve a huge array of problems.

This project report is a summary of my recent journey in ML and especially deep learning and NLP techniques.It also serves as a firm and solid continuation of my interim project report.Even though I was able to achieve a decent level of accuracy in my final iterations,still it's far from state of the art accuracy,just showing the huge potential and depth in this field.

There are various aspects to touch while training models and thus we would be moving step by step in this report,examining each and every aspect of how we reached towards the final model.

Introduction

Developing and training a model for a specific problem is a very iterative process,with several iterations involving different parts of the model.Thus we decided to divide the model in different parts,and would be individually processing each one separately.Our rough rubrics:

- Data extraction and preprocessing
- Training Process

A common observation might be the "versions" type treatment of the project , but that is because that's the way we moved,thus analysing in that aspect only.

Data extraction and preprocessing

Data extraction was not a problem due to already provided datasets and also no dearth of good datasets on Kaggle,UCI Repo etc.Thus it would be safe to move past this otherwise very important and hassle prone aspect of an ML project.

Preprocessing of the data was a very important job to be done as the raw data was far from clean,albeit also in text form,which was not usable for ML algorithms.

Extracting the very first and second statement from the test data would serve our point.

1. Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :)
2. Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday.

Several issues spotted and solved:

1. Lowercase the data,we don't want the algorithms to differentiate from L and l.
2. Removing things like '\',they don't add any meaning to the statement.
3. :) doesn't actually convey the fact that is a "happy or positive statement",able to use this for polarising the statement can do wonders.Thus we changed them to their meaning like ':)' changed to 'smile'.
4. Also things like changing http,https,www, etc to 'URL',since that's what they mean.
5. Avoiding repetitions in characters, and treating special characters like @.These don't add much substance but lead to significantly increased computational costs.

Many words like 'the','an','or' etc are called stopwords and they don't add much meaning to the sentences albeit increase the computational costs.

Thus nltk package's 'stopwords' was used to eliminate these words but also taking care not to remove polarised words like 'don't','not' etc.

The above techniques were used for mostly the first half of my models,however our accuracy was still around 60%,thus more refinement was done in data processing.This was the part we introduced 'Snowball Stemmer' in order to stem in only the core words,removing their derivatives ,reducing our computational costs and possibly increasing the accuracy.The former part was achieved but no significant improvement in accuracy was achieved.

This was the part we introduced word embeddings in our model.Earlier we were tokenizing the text data and then one hot encoding to feed to our model.However after we introduced the w2v to our model and used it to make an embedding matrix to be used for embedding layer in our DL models,we finally achieved a significant increase in accuracy(around 2%).

Thus we achieved the final preprocessing pipelines and data cleaning used in our final model.

Tweet_text>>NLTK[Stopwords]+NLTK[Snowball Stemmer]+Custom Cleaning+Tokenize+W2V word embeddings

Sentiment>>Changed labels to '0','1','2' from 'positive','negative','neutral' and categorized through label encoder.

Training Process

Now this is the part which took the maximum time and fiddling.

We would move into a version wise style so that the journey of reaching the final model can be understood better.

Version 1:

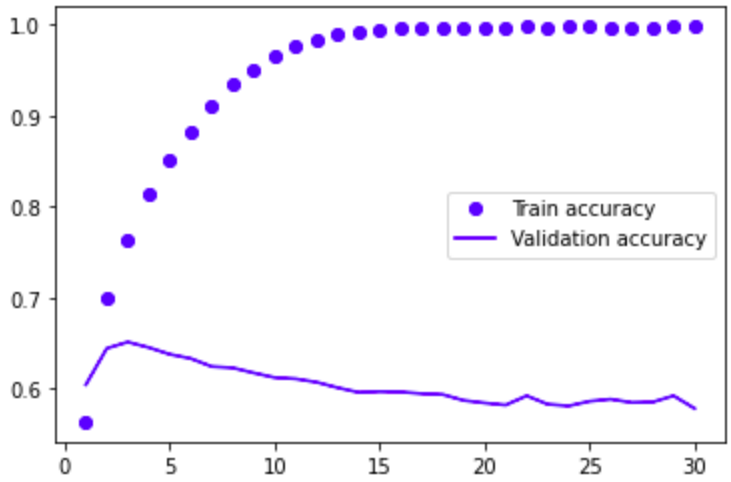
We first started with simple models of ML like Logistic Regression(Only for positive and negative),SGD Classifier and Naive Bayes model.These models were simple to set up offering me accuracies me from 40% to 55%(respective models).Fiddling with these models led to not a significant increase in accuracies,leading our way to Deep Learning.

Final accuracy- 40-55 %

Version 2:

We first tried a very basic network with 2 layers of neurons with 64 neurons each of 'relu activation' and final layer of 'softmax' activation.The model clearly presented us with overfitting problem.Even though training reached around 98% accuracy, cross_validation score reached in early 60's and 59's.However,this was a significant improvement from simple ML algorithms but still far from good.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                640064
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 195
=================================================================
Total params: 644,419
Trainable params: 644,419
Non-trainable params: 0
_________________________________________________________________
```



Accuracy test of basic model on train data

Final accuracy on test data - 60-61 %

Version 3:

Overfitting was the major problem here.

This was our approach which we used to solve it.We had 3 options for it.

1. Decrease the layers and their number of neurons.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 32)                320032
_________________________________________________________________
dense_19 (Dense)             (None, 3)                 99
=================================================================
Total params: 320,131
Trainable params: 320,131
Non-trainable params: 0
_________________________________________________________________
```

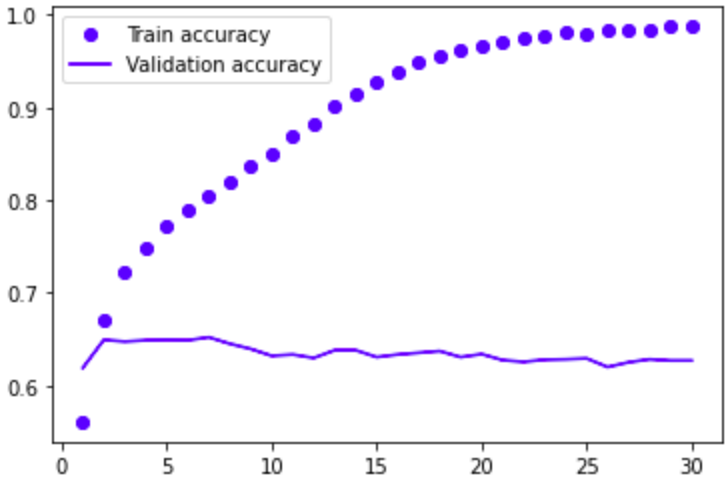
2. Include dropout layers in the model

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 64)                640064
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_21 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_22 (Dense)             (None, 3)                 195
=================================================================
Total params: 644,419
Trainable params: 644,419
Non-trainable params: 0
_________________________________________________________________
```

3. Include regularizers in the model.(l1 regularizer used)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 64)                640064
_________________________________________________________________
dense_24 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_25 (Dense)             (None, 3)                 195
=================================================================
Total params: 644,419
Trainable params: 644,419
Non-trainable params: 0
```

After testing all the three models we found the highest accuracy was found with the regularizers model,helping us significantly reduce overfitting.



Accuracy test of regularized model on train data

We finally tested the regularized model and found our final accuracy to reach 63.6 % on final test data.

Final accuracy on test data - 63.6 %

Version 4:

We had got reasonable accuracy by now,but had still to try many things.This is when we included w2v embedding and started working with LSTM and Convolutional Neural Network. We also added callbacks to stop from overfitting during training.Basically we tried everything we knew.

1. Our first basic LSTM Model-

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 300, 300)          8487000
_________________________________________________________________
lstm (LSTM)                  (None, 100)               160400
_________________________________________________________________
dense (Dense)                (None, 3)                 303
=================================================================
Total params: 8,647,703
Trainable params: 160,703
Non-trainable params: 8,487,000
_________________________________________________________________
```

We were excited,but got no significant improvement in accuracy as it kept on fluctuating between 60-63%.It would be not be wrong to say that we were confused now as what to do,and the only thing had in mind was to play with more models,change the hyperparameters(# of epochs,# of LSTM neurons,adding more dense layers,changing the w2v vocabulary size etc)

2. On consulting web articles we got to know that including 1-D CNN's might help us in the model as they "may be able to pick out invariant features for good and bad sentiment. These learned spatial features may then be learned as sequences by an LSTM layer".Thus we included them also in our model.Also we fiddled up more(changed # of epochs,shifted layers,increased # of neurons etc)and added more dense layers.Finally we also added callbacks so that we don't overtrain the model.

   FINAL MODEL:

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_14 (Embedding)     (None, 300, 300)          8487000
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 300, 32)           28832
_________________________________________________________________
max_pooling1d_10 (MaxPooling (None, 150, 32)           0
_________________________________________________________________
dropout_11 (Dropout)         (None, 150, 32)           0
_________________________________________________________________
lstm_12 (LSTM)               (None, 200)               186400
_________________________________________________________________
dense_17 (Dense)             (None, 150)               30150
_________________________________________________________________
dropout_12 (Dropout)         (None, 150)               0
_________________________________________________________________
dense_18 (Dense)             (None, 3)                 453
=================================================================
Total params: 8,732,835
Trainable params: 245,835
Non-trainable params: 8,487,000
_________________________________________________________________
None
```

This model gave us better and improved results,albeit with a small margin.

```
Train on 20391 samples, validate on 1074 samples
Epoch 1/20
20391/20391 [==============================] - 12s 596us/step - loss: 0.6079 - accuracy: 0.6566 - val_loss: 0.5475 - val_accuracy: 0.7216
Epoch 2/20
20391/20391 [==============================] - 11s 560us/step - loss: 0.5508 - accuracy: 0.7144 - val_loss: 0.5144 - val_accuracy: 0.7415
Epoch 3/20
20391/20391 [==============================] - 11s 560us/step - loss: 0.5325 - accuracy: 0.7303 - val_loss: 0.5036 - val_accuracy: 0.7486
Epoch 4/20
20391/20391 [==============================] - 11s 554us/step - loss: 0.5234 - accuracy: 0.7342 - val_loss: 0.5031 - val_accuracy: 0.7467
Epoch 5/20
20391/20391 [==============================] - 11s 551us/step - loss: 0.5178 - accuracy: 0.7390 - val_loss: 0.5040 - val_accuracy: 0.7439
Epoch 6/20
20391/20391 [==============================] - 11s 558us/step - loss: 0.5114 - accuracy: 0.7440 - val_loss: 0.5011 - val_accuracy: 0.7461
Epoch 7/20
20391/20391 [==============================] - 11s 555us/step - loss: 0.5069 - accuracy: 0.7476 - val_loss: 0.4983 - val_accuracy: 0.7561
Epoch 8/20
20391/20391 [==============================] - 12s 568us/step - loss: 0.5047 - accuracy: 0.7491 - val_loss: 0.4969 - val_accuracy: 0.7526
Epoch 9/20
20391/20391 [==============================] - 11s 561us/step - loss: 0.5006 - accuracy: 0.7520 - val_loss: 0.4959 - val_accuracy: 0.7536
Epoch 10/20
20391/20391 [==============================] - 11s 557us/step - loss: 0.4978 - accuracy: 0.7537 - val_loss: 0.4969 - val_accuracy: 0.7489
Epoch 11/20
20391/20391 [==============================] - 11s 552us/step - loss: 0.4956 - accuracy: 0.7540 - val_loss: 0.4969 - val_accuracy: 0.7523
Epoch 12/20
20391/20391 [==============================] - 12s 565us/step - loss: 0.4929 - accuracy: 0.7569 - val_loss: 0.4964 - val_accuracy: 0.7554
CPU times: user 3min 4s, sys: 25.5 s, total: 3min 30s
Wall time: 2min 18s
```

It achieved around 75% accuracy on validation sets.

Final accuracy on test data - 65.238 %[Current high score for us]

Summary

So finally we were able to achieve ~65% accuracy.

We worked good on some areas like helping not overfit our model,or making our model clean.Still there were plenty of areas where we expected our model to perform really well,but it didn't went up to the expectations.Some notes-

1. w2v didn't work up to expectations. One good example is as follows.

```
[ ] w2v_model.most_similar("love")
```

```
/usr/local/lib/python3.6/dist-packages/ipy
  """Entry point for launching an IPython
/usr/local/lib/python3.6/dist-packages/ger
  if np.issubdtype(vec.dtype, np.int):
[('x', 0.43743306398391724),
 ('appreciate', 0.43567147850990295),
 ('hate', 0.3938419222831726),
 ('mind', 0.38253432512283325),
 ('sa', 0.3786958158016205),
 ('poetic', 0.37081852555274963),
 ('enjoyed', 0.3705761134624481),
 ('malik', 0.36925673484802246),
 ('xx', 0.3685652017593384),
 ('sweet', 0.3647758960723877)]
```

Note how 'hate' is closely embedded to 'love' in the embeddings.This is very strange.

We tried different word embeddings also,but still got similar results,thus stuck to w2v finally.Still,word embedding was a letdown for us.

2. One more thing we experienced was how 'experimentative' training process is.After adding LSTM and CNN and still not getting very great results,all what we did majorly was change and tune our hyperparameters,which was purely experimentative and seemed like based on luck(which is false).It seems this is a process of experience,which the author doesn't have much.Anyways even though we tried to understand LSTM and CNN's as much as we can,it was pretty much like operating from a black-box.We worked our way for days and managed to only improve by 2-3%.
3. Also the data was little non-regularised (with negative sentiment less) and even though we tried to do something for it, we were not able to implement it in the final model.
4. One strange thing experienced was how the number of outputs exceeded the number of inputs of test data when training the models through TPU's in Kaggle.The problem was resolved when trained through GPU's.The issue was also not discussed much in detail on the internet.
5. Finally we still feel that more experience with these models and possibly more techniques in Deep Learning can still help us to improve our accuracy much.

References and sources

1. AI and ML IITK Course
2. https://keras.io/
3. https://www.tensorflow.org/ [Articles on Deep Learning/LSTM/CNN and hyperparameter tuning]
4. https://machinelearningmastery.com/ [Understanding and improving NLP techniques and implementing w2v]
5. https://www.kaggle.com/ [Helping implement w2v]
6. Picture Credit-kdnuggets