Data Science Applications

Classification Assignment

Abdelrahman Ibrahim

Lamis Sayed

Minah Ghanem

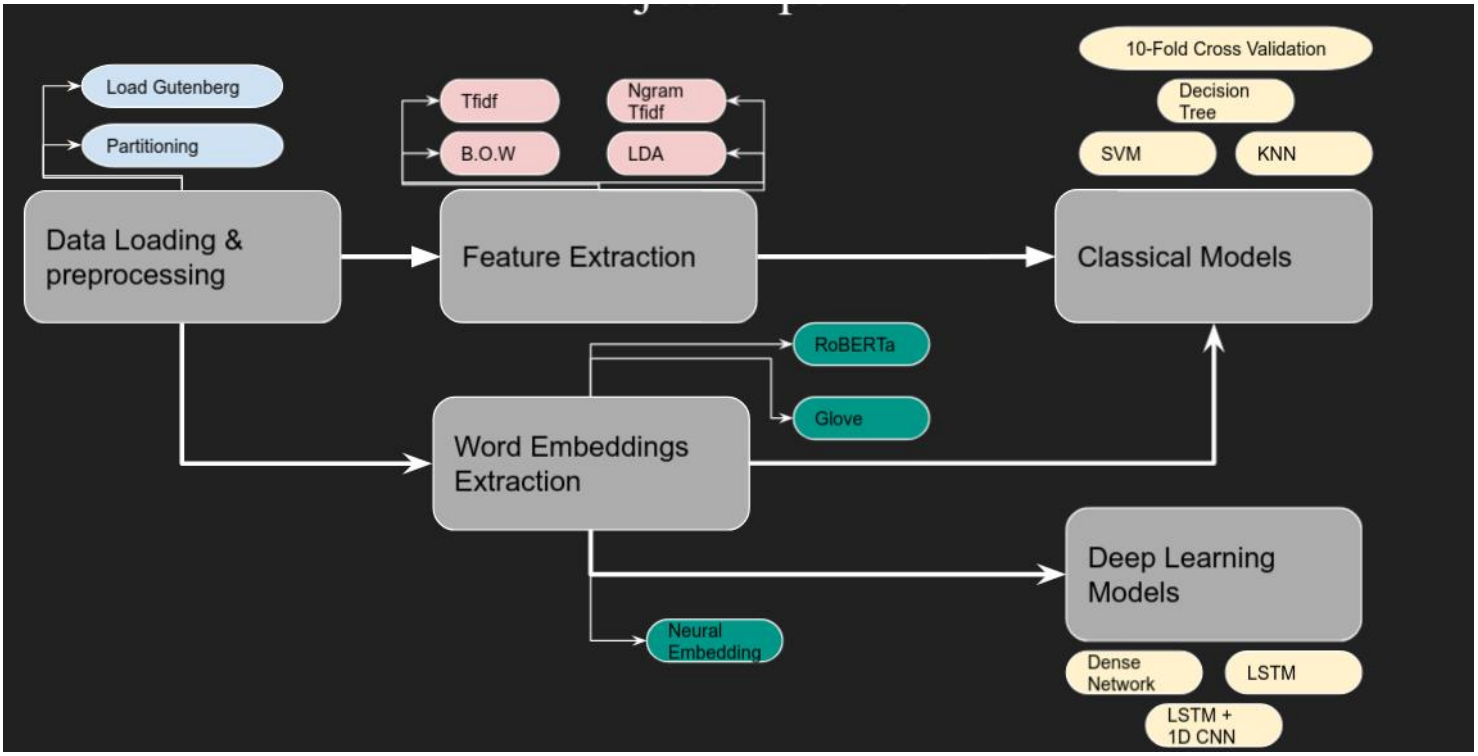
Mohammed El Namory

29th May, 2021

Introduction

In this assignment we have classified different samples of five different books from the Gutenberg digital books. Our problem is focused on classifying text into five different categories or authors: **milton-paradise , shakespeare-caesar, melville-moby_dick, chesterton-brown, whitman-leaves**

Through the whole process of our task, we have passed through multiple steps which will be introduced in the next pages:



Data Preprocessing

Main Functions:

Book_pagination is used to make partitions out of the text. It splits each book into 200 different partition that each one of it consists of 150 word.

```
def book_pagination(book, book_name):
    """
    Partitioning(pagination) of the book to take each 100 word with the label/book_name for each partition.
    """
    tokenized_words=nltk.word_tokenize(book)
    offset = 0
    pages = []
    for i in range(1, min(int(np.floor(len(tokenized_words)/100.0)), 200)):
        limit = i*100
        pages.append({'book_name': book_name,
                      'partition': " ".join(tokenized_words[offset:limit])})
        offset = limit
    return pages
```

Clean_text is used for cleaning by (lowering text - removing punctuations - keeping only alphanumeric tokens - removing any tabs - removing stop words)

```
def clean_text(text):
  ## 1. Lowercase the text
  text = text.lower()

  ## 2. Remove Punctuations
  text = text.translate(str.maketrans('', '', string.punctuation))

  ## 3. Tokenize all the words
  words = nltk.word_tokenize(text)

  ## 4. Remove stopwords and word digits
  clean_text = " ".join([ w for w in words if w.isalnum() ])
  clean_text = clean_text.replace("\t", ' ')
  # clean_text = " ".join([ w for w in words if w.isalnum() and (w not in stop_words)  ])
  return clean_text
```

After the cleaning stage is finished our data is transformed into a dataframe to be easily dealt with. And splitted randomly into training and testing datasets with a percentage of 80% to 20% respectively

Feature Extraction

In the feature extraction phase, we have tried multiple methods and compared their results:

CountVectorizer Features:

It is a simple way of building a vocabulary of known words. The feature output of that method is an encoded vector with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

CountVectorizer Using n-grams:

The same idea is implemented using n-grams. Which is simply a sequence of N words.instead of getting the frequency of one word, the same method is implemented on N sequence of words.

TF-IDF Features:

In this method we have calculated the Term Frequency – Inverse Document (TF-IDF)

Which consists of **Term Frequency:** This summarizes how often a given word appears within a document. **And the Inverse Document Frequency:** This downscales words that appear a lot across documents.

It overcomes the issue of the simple count of having large counts for words that do not have meaningful insights, instead it gives more weight for words that matter.

LDA Features:

Latent Dirichlet Allocation (LDA) is an algorithm to extract topic modeling from large documents. We Made a vector out of the LDA model to use it as input for the supervised classifier. The feature vector for each partition consists of a set of probabilities of all the topics. We used the soft prediction of the topic classification model as an input feature for the supervised classifier

---

Classical Models

We used 3 different classifiers algorithms SVM, Decision Tree and K-nearest neighbors.

- **Support Vector Machine(SVM):**

  SVM is a relatively simple Supervised Machine Learning Algorithm used for classification and/or regression. It is more preferred for classification. In text classification, It determines the best decision boundary between vectors that belong to a given category and vectors that do not belong to it. So, the texts have to be transformed into vectors. We applied it on TFiDF, BOW and n-grams.

TDiDF Results:

```
SVM on TFiDF
SVM  Confusion Matrix:  [[76  0  0  0  0]
 [ 1 78  4  4  1]
 [ 1  0 70  2  2]
 [ 0  0  1 67  0]
 [ 0  0  3  0 79]]
SVM  Accuracy :  95.11568123393316
SVM  Report :
                      precision    recall  f1-score   support

    milton-paradise       0.97      1.00      0.99        76
  shakespeare-caesar      1.00      0.89      0.94        88
  shakespeare-hamlet      0.90      0.93      0.92        75
 shakespeare-macbeth      0.92      0.99      0.95        68
      whitman-leaves      0.96      0.96      0.96        82

            accuracy                          0.95       389
           macro avg      0.95      0.95      0.95       389
        weighted avg      0.95      0.95      0.95       389
```

- **Decision Tree:**

A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label.

```
Decision Tree on TFiDF
Decision Tree  Confusion Matrix:  [[69  0  1  0  6]
 [ 2 63  6  8  9]
 [ 4  4 51  5 11]
 [ 0  5 10 48  5]
 [17  2  4  3 56]]
Decision Tree  Accuracy :  73.7789203084833
Decision Tree  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.75      0.91      0.82        76
 shakespeare-caesar       0.85      0.72      0.78        88
 shakespeare-hamlet       0.71      0.68      0.69        75
shakespeare-macbeth       0.75      0.71      0.73        68
     whitman-leaves       0.64      0.68      0.66        82

           accuracy                           0.74       389
          macro avg       0.74      0.74      0.74       389
       weighted avg       0.74      0.74      0.74       389
```

- **K-nearest neigbours(KNN):**

  The closeness or proximity amongst samples of data determines their neighborhood and this is done by calculating the distance between points.

The K in KNN indicates the number of categories that the data will be classified to. We use K=15 because as we increase the number of classes the classification become easier so the accuracy increases.

```
KNN on TFiDF
KNN  Confusion Matrix:  [[75  0  0  0  1]
 [ 0 84  3  0  1]
 [ 0  2 72  1  0]
 [ 0  8  9 51  0]
 [ 1  1  5  0 75]]
KNN  Accuracy :  91.77377892030847
KNN  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.99      0.99      0.99        76
  shakespeare-caesar      0.88      0.95      0.92        88
  shakespeare-hamlet      0.81      0.96      0.88        75
shakespeare-macbeth       0.98      0.75      0.85        68
      whitman-leaves      0.97      0.91      0.94        82

           accuracy                           0.92       389
          macro avg       0.93      0.91      0.92       389
       weighted avg       0.93      0.92      0.92       389
```

BOW Results

```
SVM on BOW
SVM  Confusion Matrix:  [[75  0  0  1  0]
 [ 2 72  8  4  2]
 [ 2  0 65  3  5]
 [ 0  2  8 58  0]
 [ 1  1  2  0 78]]
SVM  Accuracy :  89.46015424164524
SVM  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.94      0.99      0.96        76
 shakespeare-caesar       0.96      0.82      0.88        88
 shakespeare-hamlet       0.78      0.87      0.82        75
shakespeare-macbeth       0.88      0.85      0.87        68
     whitman-leaves       0.92      0.95      0.93        82

           accuracy                           0.89       389
          macro avg       0.90      0.90      0.89       389
       weighted avg       0.90      0.89      0.89       389
```

```
Decision Tree on BOW
Decision Tree  Confusion Matrix:  [[68  0  3  0  5]
 [ 1 60 13  6  8]
 [ 3  3 54  3 12]
 [ 3  5  9 46  5]
 [12  0  8  2 60]]
Decision Tree  Accuracy :  74.03598971722364
Decision Tree  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.78      0.89      0.83        76
 shakespeare-caesar       0.88      0.68      0.77        88
 shakespeare-hamlet       0.62      0.72      0.67        75
shakespeare-macbeth       0.81      0.68      0.74        68
     whitman-leaves       0.67      0.73      0.70        82

           accuracy                           0.74       389
          macro avg       0.75      0.74      0.74       389
       weighted avg       0.75      0.74      0.74       389
```

```
KNN on BOW
KNN  Confusion Matrix:  [[76  0  0  0  0]
 [ 7 50 13 17  1]
 [10  1 53 10  1]
 [ 5  0 14 49  0]
 [12  3  6  4 57]]
KNN  Accuracy :  73.26478149100257
KNN  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.69      1.00      0.82        76
 shakespeare-caesar       0.93      0.57      0.70        88
 shakespeare-hamlet       0.62      0.71      0.66        75
shakespeare-macbeth       0.61      0.72      0.66        68
      whitman-leaves       0.97      0.70      0.81        82

            accuracy                           0.73       389
           macro avg       0.76      0.74      0.73       389
        weighted avg       0.77      0.73      0.73       389
```

Bi-grams Results:

In Bi-grams we used TFiDf feature extraction. It produces accuracy better than the BOW

```
SVM on n-grams
SVM  Confusion Matrix:  [[76  0  0  0  0]
 [33  0 55  0  0]
 [15  0 59  0  1]
 [35  0 33  0  0]
 [43  0  5  0 34]]
SVM  Accuracy :  43.44473007712082
SVM  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.38      1.00      0.55        76
  shakespeare-caesar       0.00      0.00      0.00        88
  shakespeare-hamlet       0.39      0.79      0.52        75
 shakespeare-macbeth       0.00      0.00      0.00        68
      whitman-leaves       0.97      0.41      0.58        82

            accuracy                           0.43       389
           macro avg       0.35      0.44      0.33       389
        weighted avg       0.35      0.43      0.33       389

Decision Tree  Confusion Matrix:  [[42  3  4 23  4]
 [ 3 37 16 20 12]
 [ 3  9 38 19  6]
 [ 2  6 18 39  3]
 [ 8  4  1 22 47]]
Decision Tree  Accuracy :  52.185089974293064
Decision Tree  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.72      0.55      0.63        76
  shakespeare-caesar       0.63      0.42      0.50        88
  shakespeare-hamlet       0.49      0.51      0.50        75
 shakespeare-macbeth       0.32      0.57      0.41        68
      whitman-leaves       0.65      0.57      0.61        82

            accuracy                           0.52       389
           macro avg       0.56      0.53      0.53       389
        weighted avg       0.57      0.52      0.53       389
```

```
KNN on n-grams
KNN  Confusion Matrix:  [[70  2  2  0  2]
 [ 0 69 14  0  5]
 [ 2 11 57  2  3]
 [ 0 13 26 27  2]
 [ 0  3  2  1 76]]
KNN  Accuracy :  76.86375321336762
KNN  Report :
                     precision    recall  f1-score   support

    milton-paradise       0.97      0.92      0.95        76
  shakespeare-caesar       0.70      0.78      0.74        88
  shakespeare-hamlet       0.56      0.76      0.65        75
shakespeare-macbeth       0.90      0.40      0.55        68
     whitman-leaves       0.86      0.93      0.89        82

           accuracy                           0.77       389
          macro avg       0.80      0.76      0.76       389
       weighted avg       0.80      0.77      0.76       389
```

Cross-Validation k-fold:

We split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time. We 10 folds

TFiDF Results:

```
Cross Validation SVM on TFiDF
Accuracy:  0.9671049863244268
```

```
Cross Validation Decision Tree on TFiDF
Accuracy:  0.7509993688196929
```

```
Cross Validation KNN on TFiDF
Accuracy:  0.9104881127708817
```

BOW Results:

```
Cross Validation SVM on BOW
Accuracy:  0.8992005049442457


Cross Validation Decision Tree on BOW
Accuracy:  0.7624026930359772


Cross Validation KNN on BOW
Accuracy:  0.7458762886597937
```

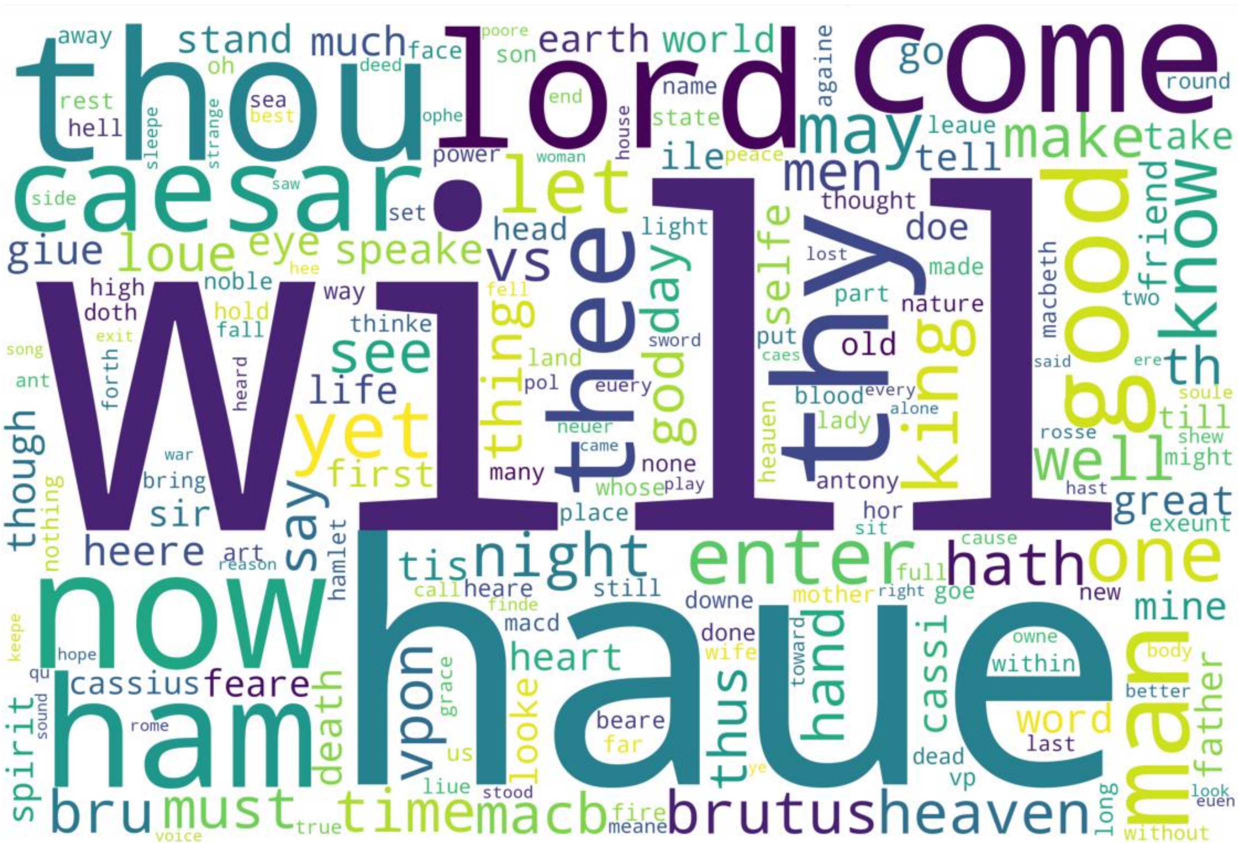
N-grams Results:

```
SVM on n-grams
Accuracy:  0.6594782242794024


Decision Tree on n-grams
Accuracy:  0.575131495897328


KNN on n-grams
Accuracy:  0.7912160740584894
```

Word Cloud Representation:

We have made a word cloud representation of our corpus to visualize our input.

```
# text is the input to the generate() method
wordcloud = WordCloud(width = 3000, height = 2000, random_state=1, background_color='white', collocations=False).generate(''.join(books_df.partition.values.tolist()))
#draw the figure
#Set figure size
plt.figure(figsize=(20, 20))
# Display image
plt.imshow(wordcloud)
# No axis
plt.axis("off")
plt.show()
```


will
haue
thou
lord
come
caesar
now
ham
thy
good
thee
man
let
may
make
know
king
yet
enter
night
one
hath
brutus
heaven
time
macb
must
bru
hand
vpon
see
well
thing
god
day
loue
speake
eye
vs
men
tell
take
ile
go
earth
world
stand
much
th
selfe
great
first
say
tis
heere
heart
death
looke
thus
cassi
cassius
feare
word
mine
father
friend
spirit
though
sir
life
giue
doe
head
old
till
away
face
son
poore
oh
deed
end
rest
sea
best
ophe
hell
sleepe
strange
power
woman
house
peace
name
state
againe
leaue
round
side
saw
set
light
thought
lost
made
macbeth
two
noble
high
doth
hold
fall
way
thinke
land
pol
euery
blood
every
alone
lady
nature
part
put
sword
caes
said
ere
soule
might
shew
rosse
song
exit
ant
forth
heard
war
bring
neuer
came
none
many
whose
play
place
antony
heauen
hast
hor
sit
exeunt
cause
nothing
hamlet
art
reason
call
heare
still
downe
mother
right
full
goe
new
finde
macd
done
keepe
hope
qu
wife
toward
owne
body
within
grace
better
sound
rome
us
beare
far
last
dead
vp
ye
liue
stood
fire
meane
long
look
euen
voice
true
without

Word Embeddings Extraction:

We used the function load_embeddings to load pertained models to embed and vectorize sentences

```
def load_embeddings(embeddings_path , sentences):
  """
     Load pre-trained embeddings models to embed and vectorize sentences
  """
  ## Use word embeddings to extract the average sentence embeddings
  model = SentenceTransformer(embeddings_path)
  sentence_embeddings = model.encode(sentences)
  print("Shape of sentences after embeddings ::")
  print(sentence_embeddings.shape)

  ## Splitting data into train/test for modelling
  return sentence_embeddings
```

GLoVe

GLoVe: global vector for word representation 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.

We used a pretrained model from glove “ average_word_embeddings_glove.6B.300d” which is trained on 6 billion tokens and has a feature vector of a length of 300.

```
Embeddings
Train Accuracy :  96.98492462311557
Test Accuracy :  83.41708542713567

                 TEST DATA METRICS
[[40  5  1  0  1]
 [ 4 33  2  0  4]
 [ 0  1 23  0  1]
 [ 0  0  3 34  3]
 [ 1  5  2  0 36]]
              precision    recall  f1-score   support

           0       0.89      0.85      0.87        47
           1       0.75      0.77      0.76        43
           2       0.74      0.92      0.82        25
           3       1.00      0.85      0.92        40
           4       0.80      0.82      0.81        44

    accuracy                           0.83       199
   macro avg       0.84      0.84      0.84       199
weighted avg       0.84      0.83      0.84       199
```

RoBERTa

RoBERTa is a language based model based on BERT. It improves on Bidirectional Encoder Representations from Transformers, or BERT, the self-supervised method released by Google in 2018.

```
Embeddings
Train Accuracy :  99.2462311557789
Test Accuracy :  84.92462311557789

                 TEST DATA METRICS
[[45  1  0  0  1]
 [ 4 31  4  1  3]
 [ 0  0 24  0  1]
 [ 1  0  4 33  2]
 [ 2  2  2  2 36]]
              precision    recall  f1-score   support

           0       0.87      0.96      0.91        47
           1       0.91      0.72      0.81        43
           2       0.71      0.96      0.81        25
           3       0.92      0.82      0.87        40
           4       0.84      0.82      0.83        44

    accuracy                           0.85       199
   macro avg       0.85      0.86      0.84       199
weighted avg       0.86      0.85      0.85       199
```

Deep Learning Models

Embedding Layer

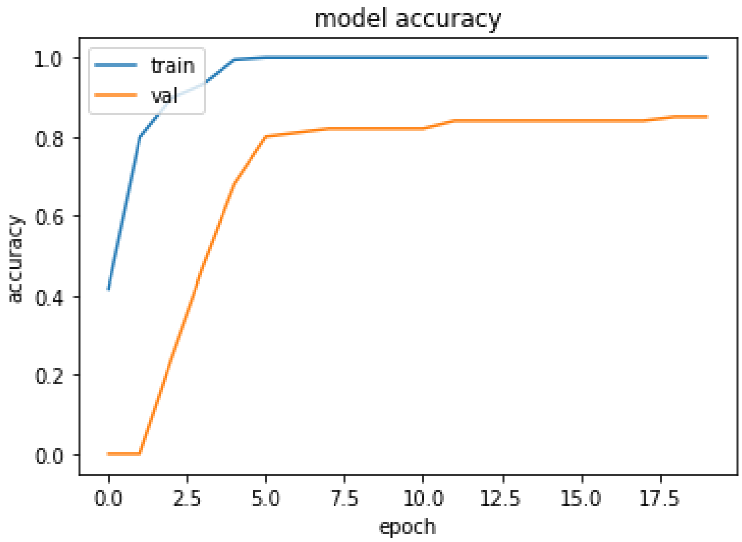
- In each DL model we used; we added an embedding layer for our architecture before passing it to the deep neural network.
- The aim of using a neural embedding layer is to build the context vector and learn more about the semantic and syntactic meaning of each partition sequence passed to neural network.

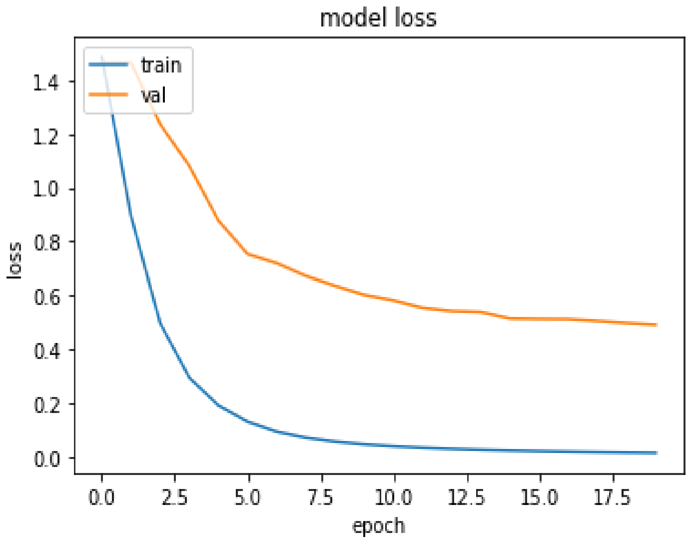
Model-1

- Our first deep learning model is not so deep, just 10 units of dense neural network with a softmax layer.

```
[ ]  # define the model
     model = Sequential()
     # model.add(Embedding(vocab_size, 8, input_length=max_length))
     model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
     model.add(Flatten())
     model.add(Dense(10, activation='tanh'))
     model.add(Dense(5, activation='softmax'))
     # compile the model
     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
     # summarize the model
     print(model.summary())
```

Error Analysis:

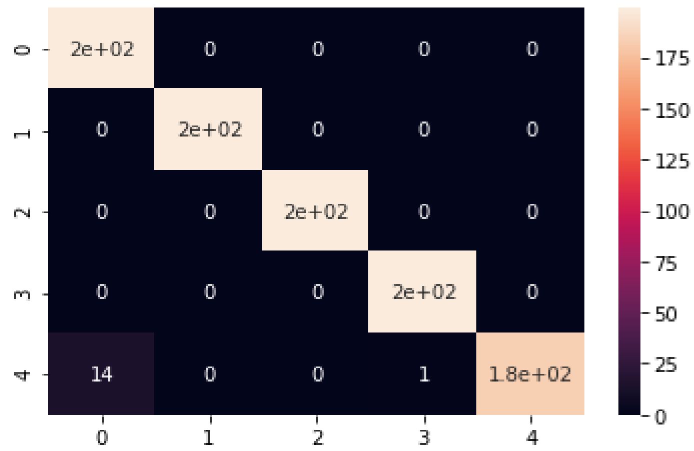




Classification Report

	precision	recall	f1-score	support
0	0.93	1.00	0.97	199
1	1.00	1.00	1.00	199
2	1.00	1.00	1.00	199
3	0.99	1.00	1.00	199
4	1.00	0.92	0.96	199
accuracy			0.98	995
macro avg	0.99	0.98	0.98	995
weighted avg	0.99	0.98	0.98	995

Confusion Matrix

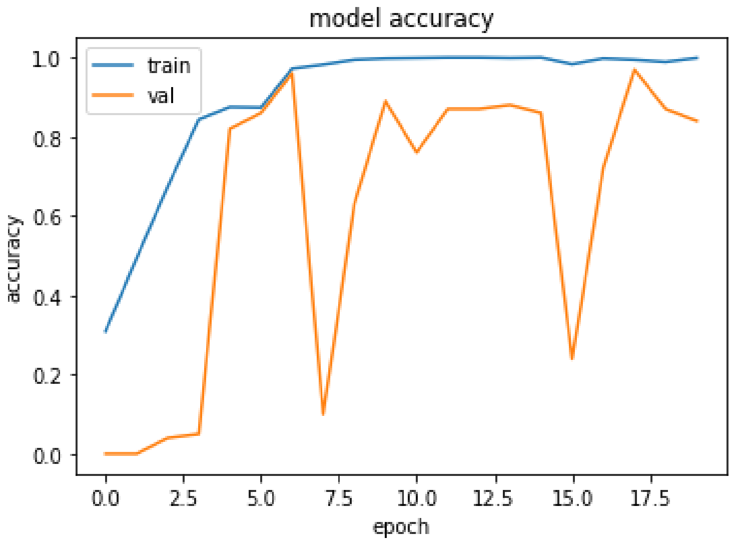


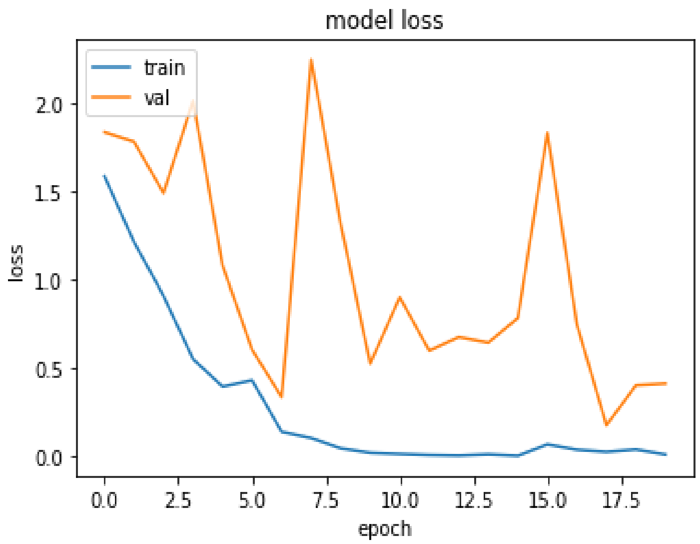
- **Model-2**

    - Here we used a new RNN-based model. We used the LSTM unit to deal with the sequence data better than the dense network which is widely used in the NLP work flow.

```
[ ]  model = Sequential()
     model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
     model.add(SpatialDropout1D(0.2))
     model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
     model.add(Dense(5, activation='softmax'))
     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

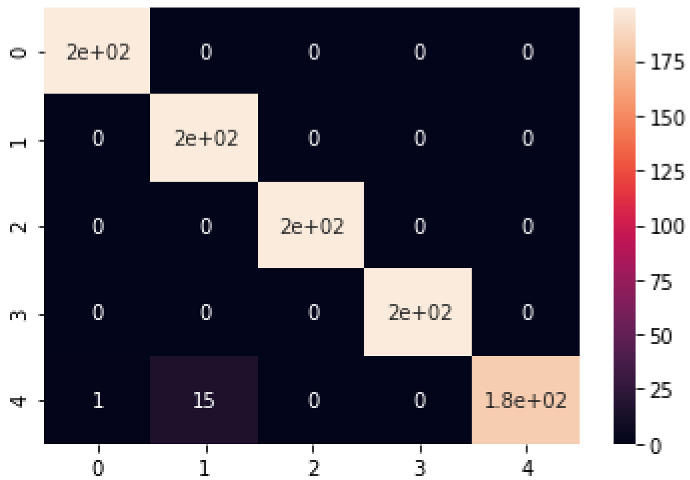
Error Analysis:





```
Classification Report

              precision    recall  f1-score   support

           0       0.99      1.00      1.00       199
           1       0.93      1.00      0.96       199
           2       1.00      1.00      1.00       199
           3       1.00      1.00      1.00       199
           4       1.00      0.92      0.96       199

    accuracy                           0.98       995
   macro avg       0.98      0.98      0.98       995
weighted avg       0.98      0.98      0.98       995
```

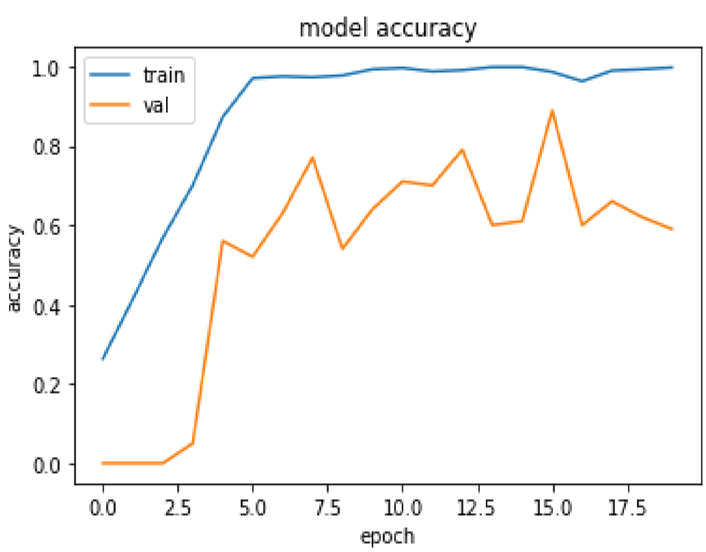
Confusion Matrix

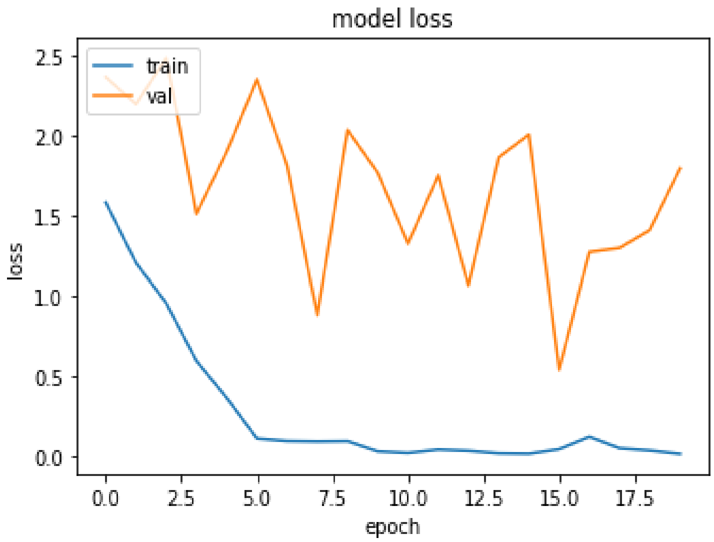


- Model-3
  - Here we used a more complex model leveraging two types of neural networks LSTM + CNN. We used 1D CNN to pass on the sequence and treat it as an image and pass a filter on the text sequence to highlight the important parts in the text.

```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(Dropout(0.2))
model.add(Conv1D(8, 5, activation='relu'))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(100))
model.add(Dense(5, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',    metrics=['accuracy'])
```

Error Analysis:

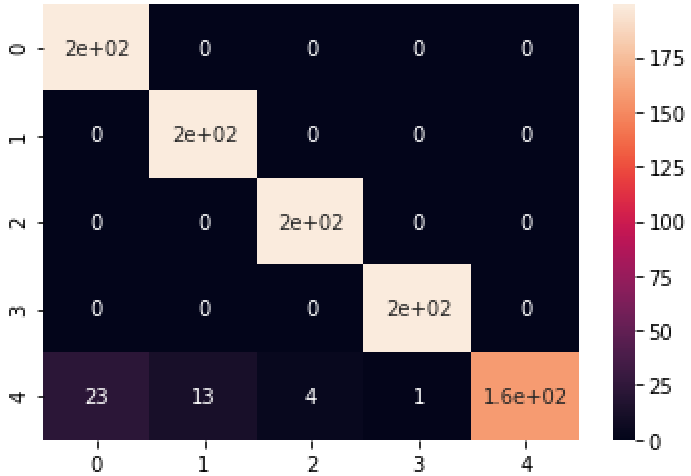




Classification Report

	precision	recall	f1-score	support
0	0.90	1.00	0.95	199
1	0.94	1.00	0.97	199
2	0.98	1.00	0.99	199
3	0.99	1.00	1.00	199
4	1.00	0.79	0.89	199
accuracy			0.96	995
macro avg	0.96	0.96	0.96	995
weighted avg	0.96	0.96	0.96	995

Confusion Matrix



Comparing Results

Deep Learning Models

	Confusion matrix				
Model	Avg. Precision	Avg. Recall	Avg. F1-score	Train accuracy	Validation Accuracy
Dense	0.99	0.98	0.98	100.00%	85.00%
LSTM	0.98	0.98	0.98	99.80%	84.00%
LSTM+CNN	0.96	0.96	0.96	99.57%	59.00%

Cross Validation metrics

Model	Features	Avg. Accuracy
SVM	TFIDF	95.68%
	BOW	88.94%
	Bigram TFIDF	65.94%
	LDA	74.58%
Decision Tree	TFIDF	76.02%
	BOW	76.85%
	Bigram TFIDF	57.51%
	LDA	68.10%
KNN	TFIDF	91.55%
	BOW	73.17%
	Bigram TFIDF	79.12%
	LDA	67.58%

		Test data metrics				
Model	Features	Avg. Precision	Avg. Recall	Avg. F1-score	Train-accuracy	Test-accuracy
SVM	TFIDF	0.954	0.958	0.954	100.00%	95.48%
	BOW	0.86	0.87	0.86	99.49%	85.42%
	Bigram TFIDF	0.35	0.44	0.33	100.00%	43.44%
	Glove	0.84	0.84	0.84	96.98%	83.42%
	RoBERTa	0.85	0.86	0.84	99.25%	84.93%
Decision Tree	TFIDF	0.646	0.64	0.64	100.00%	62.31%
	BOW	0.61	0.61	0.61	100.00%	58.79%
	Bigram TFIDF	0.55	0.52	0.53	100.00%	51.92%
KNN	TFIDF	0.922	0.916	0.916	93.71%	91.45%
	BOW	0.78	0.73	0.73	81.78%	71.85%
	Bigram TFIDF	0.8	0.76	0.76	100	76.86%

References:

- Results Sheet: https://docs.google.com/spreadsheets/d/13BLKEPEv0fWEbEFswhlal5dQV6FNDevyl27sSdOih8g/edit?usp=sharing
- https://towardsdatascience.com/unsupervised-nlp-topic-models-as-a-supervised-learning-input-cf8ee9e5cf28
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
- https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/
- https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
- https://realpython.com/python-keras-text-classification/#convolutional-neural-networks-cnn
- https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
- Lecture notes from Dr. Arya Rahgozar and Notebook samples for visualization and loading the pre-trained models