Capstone 2 Milestone Report

Priya Sathish

Content based News Recommendation System

Problem

Online platforms support so many of our daily activities that we have become dependent on them in our personal and professional lives. We rely on them to buy and sell goods and services, to find information online and to keep in touch with each other. These platforms could help consumers by recommending items as per their interest and preference by just analyzing your past interaction or behavior with the system. From Amazon to LinkedIn, Uber eats to Spotify, Netflix to Facebook, Recommender systems are most extensively used to suggest "Similar items", "Relevant jobs", "preferred foods", "Movies of interest" etc. to their users. Recommender system with appropriate item suggestions helps in boosting sales, increasing revenue, retaining customers and also adds competitive advantage. Recommender systems use a number of different technologies and can be classified into two broad groups as Content based recommendation and Collaborative filtering.

On a day to day basis, the internet has a lot of sources that generate immense amount of daily news diversified in subject matter. There is continuous demand for new information to be available immediately and with ease by the consumers. So, it is crucial that the news is classified and targets the needs and requirements of the user effectively and efficiently. News services have attempted to identify articles of interest to readers based on the articles that they have read in the past. The similarity might be based on the similarity of important words in the documents or on the articles that are read by people with similar reading tastes. The same principles apply to recommending blogs from among the millions of blogs available or other sites where content is provided regularly.

This project focuses on content-based recommendation using News category dataset. The goal is to recommend news articles which are similar to the already read article by using attributes like article headline, short description, category, author and publishing date.

Client

News readers, blog readers, news agencies, bloggers, retailers and several online platforms.

Data and approach

https://www.kaggle.com/rmisra/news-category-dataset

This dataset contains around 200k news headlines from the year 2012 to 2018 obtained from HuffPost. News in this dataset belongs to 41 different categories. Each news record consists of a headline with a short description in our analysis. In addition, we will combine attributes 'headline' and 'short description' into a single attribute 'text' as the input for classification and proceed with developing a deep learning model to build the recommender system.

Citation: "https://rishabhmisra.github.io/publications/"

Data Wrangling

Acquired the news category dataset from kaggle as a json file and converted it to a pandas dataframe with a shape of (200853, 6) for analysis. When grouped by category we can see that the dataset contains 41 categories of news articles.
'THE WORLDPOST' and 'WORLDPOST' should be the same category, so we change 'THE WORLDPOST' to 'WORLDPOST' and merge them.

After which we have,
Total number of articles: 200853
Total number of authors: 27993
Total number of unique categories: 40

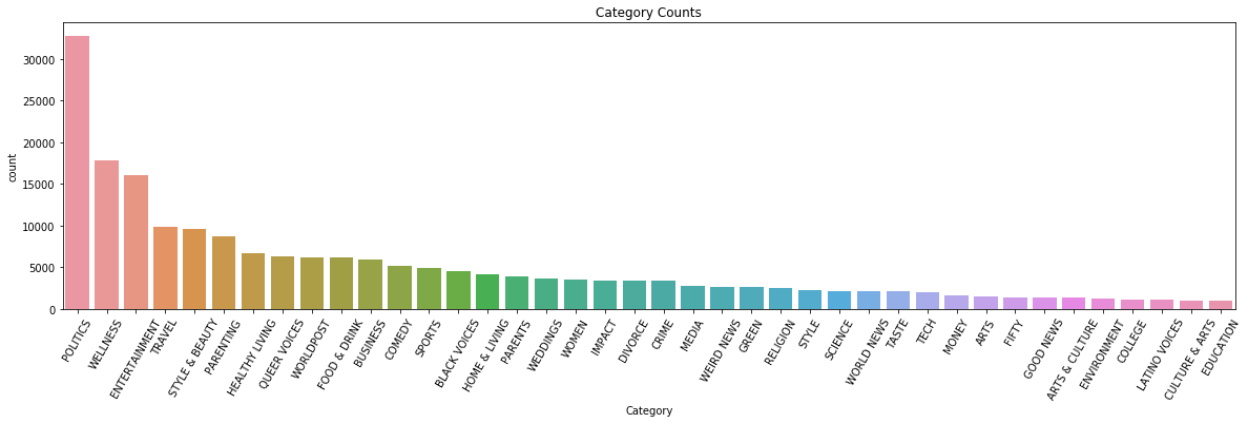
Top 5 categories include:

category	
POLITICS	32739
WELLNESS	17827
ENTERTAINMENT	16058
TRAVEL	9887
STYLE & BEAUTY	9649

Also, we will check for any missing data,

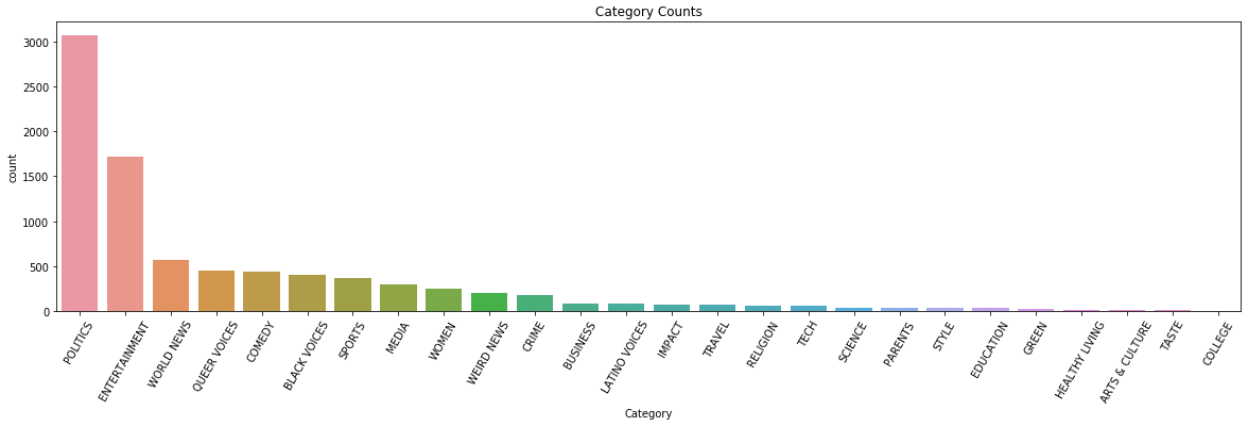
category	0
headline	0
authors	0
link	0
short_description	0

date 0
text 0
words 0
word_length 0



From the plot above we observe that politics, wellness, entertainment, travel and beauty form the top 5 categories of news article headlines during the period of 2014-2018.

We will consider only the latest articles from the year 2018 as the size of the dataset is quite large and processing may consume too much time. So, we filter the dataset to contain only row from the year 2018 and will proceed with text preprocessing.



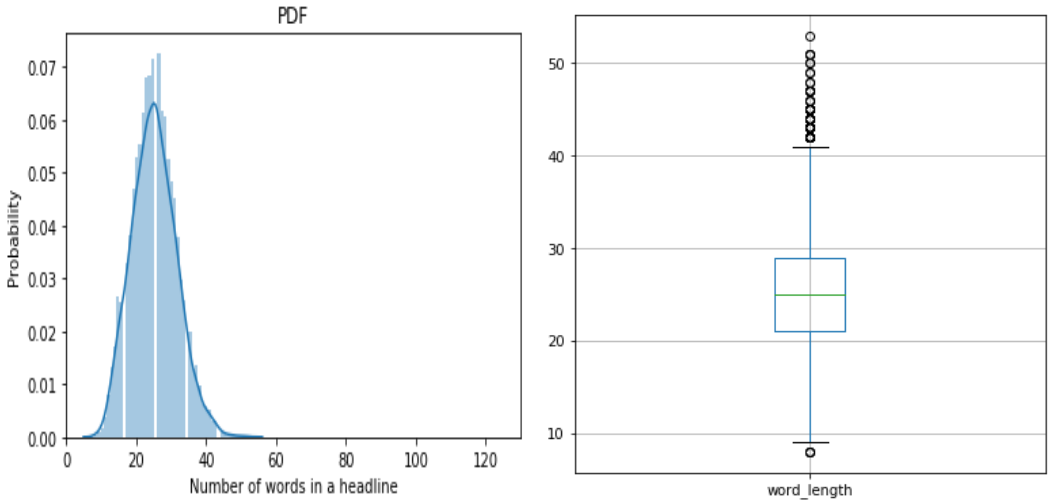
From the plot above we can see that politics, entertainment, world news, queer voices and comedy form the top 5 categories in the year 2018.

We will be using headlines and short description as input X. So, we will combine the short description and headline by concatenation and create a new column.

We tokenize the headlines and then we delete some empty and short data with word length less than 5. We take a look at the word length distribution:

Count 8583.000000
mean 25.176861
std 6.418315
min 8.000000
25% 21.000000
50% 25.000000
75% 29.000000
max 53.000000
Name: word_length, dtype: float64

We create a PDF and box plots to view the word length distribution.



From the plot above we can clearly see that the word length ranges between 8 and 53 with a mean value of 25 and standard deviation of about 6 over 8583 news articles.

Text Preprocessing

We will clean and process the text data so that it is ready for modeling using the Natural Language Toolkit or NLTK Python library.

Tokenize:
We will split the news headline text into tokens based on white space or punctuation. This is considered as a base step for stemming and lemmatization. Once we use word_tokenize() on the headline text we can use the output for stop words removal.

Stop words removal:
Stop words are not much helpful in analysis and also their inclusion consumes much time during processing so let's remove these. We will use the default NLTK corpus to remove the unwanted stop words.

Lemmatize:
Lemmatization is the algorithmic process of finding the lemma of the word depending on the meaning. We will use the WordNetLemmatizer() from the NLTK Python library to remove inflectional endings.

To perform all of the above-mentioned operations on our headline text we will define a function to process:

```
def process_headlines(main_text):
    headlines_without_numbers = re.sub('[^a-zA-Z]', ' ', main_text)
    words = word_tokenize(headlines_without_numbers.lower())
    stop_words_english = set(stopwords.words('english'))
    final_words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words_english]
    return(' '.join(final_words))
```

Bag Of Words – Count Vectorize

To extract features from the text documents and create a vocabulary of all the unique words in all of the news headlines we will use CountVectorizer() from the NLTK Python library. The BOW model only considers if a known word occurs in a document or not. It does not care about meaning, context and order in which they appear. This gives the insight that similar documents will have word counts similar to each other. In other words, the more similar the words in two documents, the more similar the documents can be. However, there are some limitations using this method such as it doesn't take the semantic meaning or context into account and also if the vector size is huge it might result in lot of computation and time.

Preliminary model evaluation using default parameters

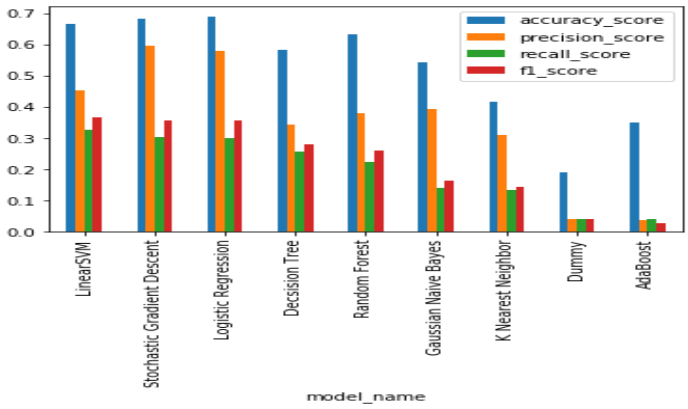
Now that we have preprocessed our text, we will try several kinds of classifiers and compare them with their default parameters. The problem we have here is that the algorithm may not perform well right away but might perform really well with the right set of hyperparameters. To get a preliminary understanding of which types of classifiers will inherently work better we will compare them.

We will take a look at 8 different classifiers along with sklearn's dummy classifier which is just a random baseline.
The classifiers we will compare include:
Dummy Classifier, Stochastic Gradient Descent, RandomForestClassifier, DecisionTreeClassifier, AdaBoostClassifier, Gaussian Naive Bayes, LogisticRegression, LinearSVM and K Nearest Neighbor.

The metrics we will use to evaluate the different classifiers are:
Accuracy – the fraction of samples predicted correctly
Precision – the ratio of true positives to false positives (ability of the classifier not to label a positive as a negative sample)
Recall – the ratio of true positives to false negatives (ability of the classifier to find all the positive samples)
F1 Score – the harmonic average of precision and recall (we will use macro averaging which will compute the average of the F1 scores)

Comparison Results:

model_name	accuracy_score	precision_score	recall_score	f1_score
LinearSVM	0.666078	0.452625	0.328205	0.36622
Stochastic Gradient Descent	0.682316	0.59672	0.305721	0.358697
Logistic Regression	0.689375	0.581914	0.301341	0.356729
Decsision Tree	0.584539	0.345634	0.257183	0.28229
Random Forest	0.633251	0.379824	0.225787	0.259581
Gaussian Naive Bayes	0.545358	0.395019	0.141648	0.164515
K Nearest Neighbor	0.417226	0.311188	0.133848	0.144641
Dummy	0.190964	0.0400646	0.0404579	0.040161



From the above plot and the F1 scores we can see that Linear SVM (0.37), Stochastic Gradient (0.36) Descent and Logistic Regression (0.36) performed better than all other classifiers. The F1 scores for the top three classifiers are also not that great though the real test is how they perform on unseen articles.

The scores were calculated only the news articles from the year 2018. The classifiers might have performed well if we had used the news articles from the years 2014-2018.

Deep Learning Models and comparison

As our next step, we will implement several deep learning models, tune them and compare them with evaluation metrics. We considered only the data from the year 2018 for our preliminary preprocessing and models. For deep learning models we will use the full dataset from years 2014 to 2018 for better performance and evaluation metrics.

A word embedding is an approach to provide a dense vector representation of words that capture something about their meaning. Word embeddings are an improvement over simpler bag-of-word model word encoding schemes like word counts and frequencies that result in large and sparse vectors (mostly 0 values) that describe documents but not the meaning of the words. Word embeddings work by using an algorithm to train a set of fixed-length dense and continuous-valued vectors based on a large corpus of text. Each word is represented by a point in the embedding space and these points are learned and moved around based on the words that surround the target word. It is defining a word by the company that it keeps that allows the word embedding to learn something about the meaning of words. The vector space representation of the words provides a projection where words with similar meanings are locally clustered within the space. The use of word embeddings over other text representations is one of the key methods that has led to breakthrough performance with deep neural networks on problems like machine translation.

GloVe Embedding

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. The smallest GloVe pre-trained model is from the GloVe website. It an 822 Megabyte zip file with 4 different models (50, 100, 200 and 300-dimensional vectors) trained on Wikipedia data with 6 billion tokens and a 400,000-word vocabulary.

Convolution Neural Network (CNN) - GloVe embedding

Traditionally CNN is popular is for identifying objects inside images. It can also be extended for text classification with the help of word embeddings. CNN has been found effective for text in search query retrieval, sentence modelling and other traditional NLP (Natural Language Processing) tasks. Once an image is converted to vectorized representation or text is converted to embedding, it looks similar to machine as shown in picture below. In case of image each cell in the represents raw intensity of specific channel whereas in case of text, each row of table represents a word. Just like in traditional CNN, lower level layers help in identifying edges, parts of bigger objects and successive layers identifies objects, in case of text classification, lower layer tried to find association between words whereas higher layer tries to find association between group of words. These groups can be sentences, paragraphs or smaller subgroups.

A typical convolution layer network architecture has multiple layers. CNN is supervised ML algorithm. The training set is first converted to word embeddings using glove embeddings. We first pass it through a series of convolution and pooling layer to extract lower levels features first and then learn higher level features from lower level features.

In our training first we split input dataset into 80% training and 20% validation set. We feed input dataset for training to CNN. we are going to apply 64 such filters on training dataset. Each filter will be applied to 2,3,4 words at a time. After convolution the output is passed through RELU activation layer to remove negative samples and keep only positive samples. Output of RELU is passed through max pooling layer to retain most important information.

We then pass the output through a dropout layer to prevent overfitting. We then pass it through another set of 1D convolution, RELU and max pooling.

Finally, the last layer in CNN is typically feed forward neural network that learns to map the pooling function output to output categories in terms of softmax probabilities.

Now that our network architecture is up, we train the model for 20 epochs and measure its performance over validation set.

Model Summary:

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 50)	0	
embedding_1 (Embedding)	(None, 50, 100)	11661800	input_1[0][0]
conv1d_1 (Conv1D)	(None, 50, 64)	12864	embedding_1[0][0]
conv1d_2 (Conv1D)	(None, 50, 64)	19264	embedding_1[0][0]
conv1d_3 (Conv1D)	(None, 50, 64)	25664	embedding_1[0][0]

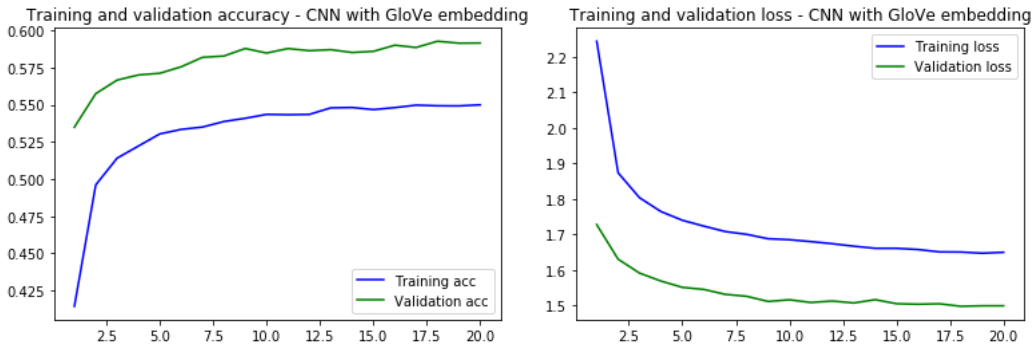
max_pooling1d_1 (MaxPooling1D)	(None, 16, 64)	0	conv1d_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 16, 64)	0	conv1d_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 16, 64)	0	conv1d_3[0][0]
dropout_1 (Dropout)	(None, 16, 64)	0	max_pooling1d_1[0][0]
dropout_2 (Dropout)	(None, 16, 64)	0	max_pooling1d_2[0][0]
dropout_3 (Dropout)	(None, 16, 64)	0	max_pooling1d_3[0][0]
concatenate_1 (Concatenate)	(None, 16, 192)	0	dropout_1[0][0] dropout_2[0][0] dropout_3[0][0]
flatten_1 (Flatten)	(None, 3072)	0	concatenate_1[0][0]
dropout_4 (Dropout)	(None, 3072)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 40)	122920	dropout_4[0][0]

Total params: 11,842,512
Trainable params: 180,712
Non-trainable params: 11,661,800

Model Score:

Validation loss: 1.4993746934662053
Validation accuracy: 0.5915763974189758

Model Metrics Plots:



Recurrent Neural Network (RNN) - Long Short Term Memory (LSTM) using Keras - without Embedding

Vectorize news headlines, by turning each text into either a sequence of integers or into a vector. Limit the data set to the top 1000 words. SpatialDropout1D performs variational dropout in NLP models. The next layer is the LSTM layer with 512 memory units. The output layer must create 40 output values, one for each class. Activation function is softmax for multi-class classification. Because it is a multi-class classification problem, categorical_crossentropy is used as the loss function.

Model Summary:

Model: "sequential_11"

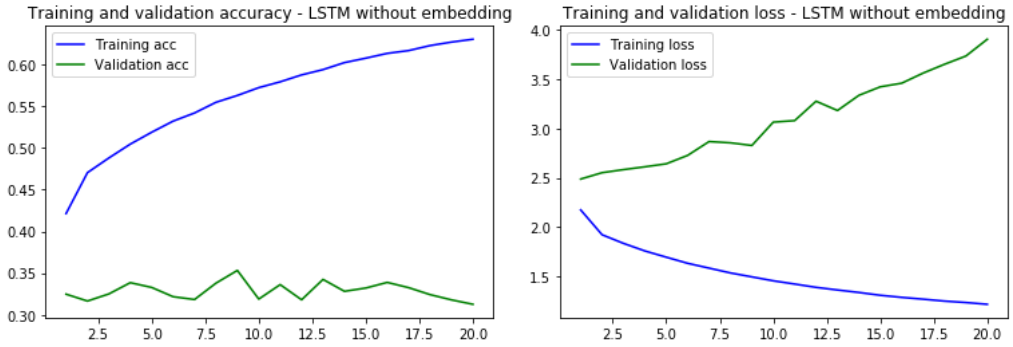
Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 512)	512512
activation_7 (Activation)	(None, 512)	0
dropout_10 (Dropout)	(None, 512)	0
dense_15 (Dense)	(None, 40)	20520
activation_8 (Activation)	(None, 40)	0

Total params: 533,032
Trainable params: 533,032
Non-trainable params: 0

Model Score:

Validation Loss: 3.9045065682964952
Validation accuracy: 0.31256356835365295

Model Metrics Plots:



Recurrent Neural Network (RNN) - Long Short Term Memory (LSTM) architecture - using Keras Embedding

Vectorize news headlines, by turning each text into either a sequence of integers or into a vector. Limit the data set to the top 1000 words. The first layer is the embedded layer that uses 100 length vectors to represent each word. SpatialDropout1D performs variational dropout in NLP models. The next layer is the LSTM layer with 100 memory units. The output layer must create 40 output values, one for each class. Activation function is softmax for multi-class classification. Because it is a multi-class classification problem, categorical_crossentropy is used as the loss function. Keras inbuilt embedding is used here.

Model Summary:

Model: "sequential_7"

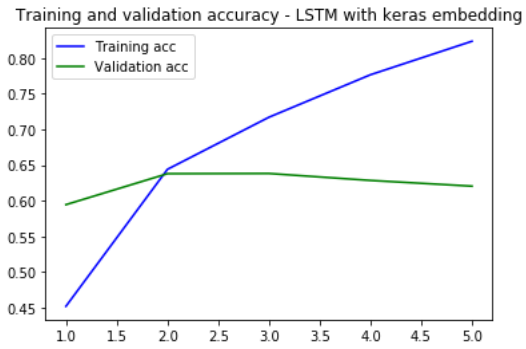
Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 50, 100)	11661800
spatial_dropout1d_3 (Spatial	(None, 50, 100)	0
lstm_3 (LSTM)	(None, 100)	80400
dense_10 (Dense)	(None, 40)	4040

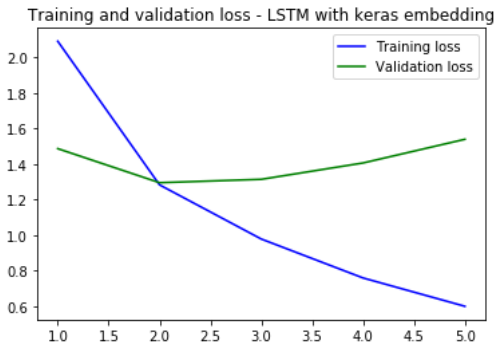
Total params: 11,746,240
Trainable params: 11,746,240
Non-trainable params: 0

Model Score:

Validation Loss: 1.5391094215074308
Validation Accuracy: 0.620488703250885

Model Metrics Plots:





Recurrent Neural Network (RNN) - Long Short Term Memory

LSTM architecture - with gensim Word2Vec

Word2vec, like doc2vec, belongs to the text preprocessing phase. Specifically, to the part that transforms a text into a row of numbers. Word2vec is a type of mapping that allows words with similar meaning to have similar vector representation. The idea behind Word2vec is rather simple: we want to use the surrounding words to represent the target words with a Neural Network whose hidden layer encodes the word representation. First, we load a word2vec model. It has been pre-trained by Google on a 100 billion-word Google News corpus. Google's pre-trained model(1.5GB!) includes word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a Google News dataset. The vector length is 300 features. Gensim allocates a big matrix to hold all of the word vectors.

Model Summary:

Model: "sequential_8"

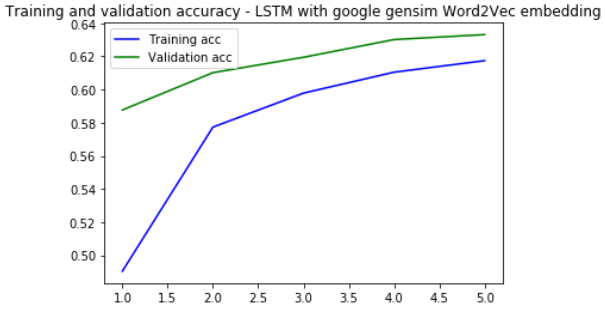
Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 50, 300)	34985400
spatial_dropout1d_4 (Spatial	(None, 50, 300)	0
lstm_4 (LSTM)	(None, 100)	160400
dense_11 (Dense)	(None, 40)	4040

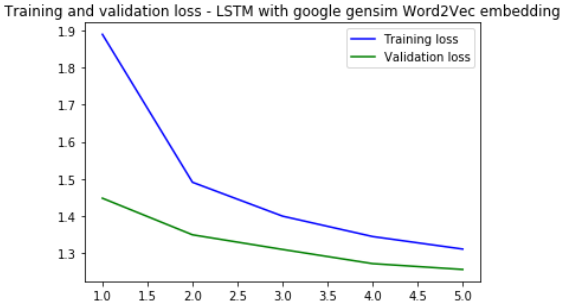
Total params: 35,149,840
Trainable params: 164,440
Non-trainable params: 34,985,400

Model Score:

Validation loss: 1.2570988957271256
Validation accuracy: 0.6331190466880798

Model Metrics Plots:





Build neural network with LSTM + CNN - with gensim Word2Vec embedding

The LSTM model worked well. However, it takes forever to train five epochs. One way to speed up the training time is to improve the network adding “Convolutional” layer. Convolutional Neural Networks (CNN) come from image processing. They pass a “filter” over the data and calculate a higher-level representation. They have been shown to work surprisingly well for text, even though they have none of the sequence processing ability of LSTMs.

Model Summary:

Model: "sequential_10"

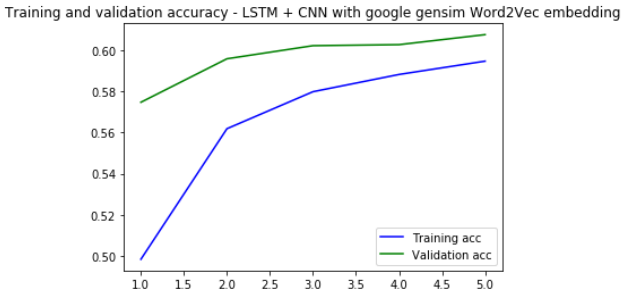
Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 50, 300)	34985400
dropout_9 (Dropout)	(None, 50, 300)	0
conv1d_5 (Conv1D)	(None, 46, 64)	96064
max_pooling1d_5 (MaxPooling1	(None, 11, 64)	0
lstm_6 (LSTM)	(None, 100)	66000
dense_13 (Dense)	(None, 40)	4040

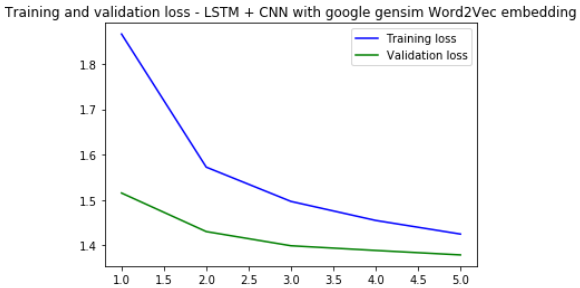
Total params: 35,151,504
Trainable params: 166,104
Non-trainable params: 34,985,400

Model Score:

Validation loss: 1.3792227489106357
Validation accuracy: 0.6077082753181458

Model Metrics Plots:

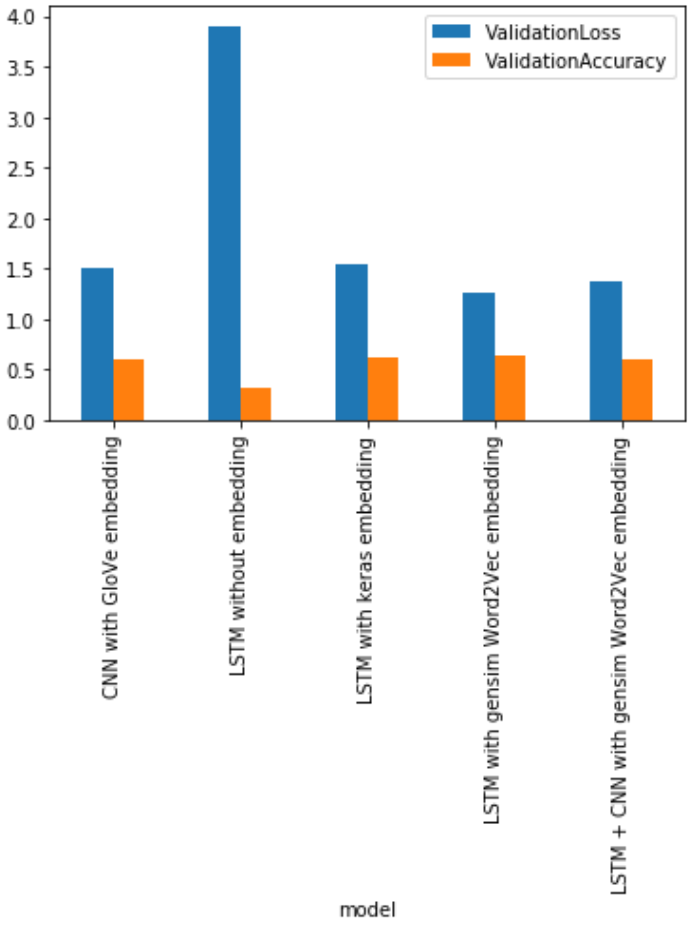




Display models and metrics

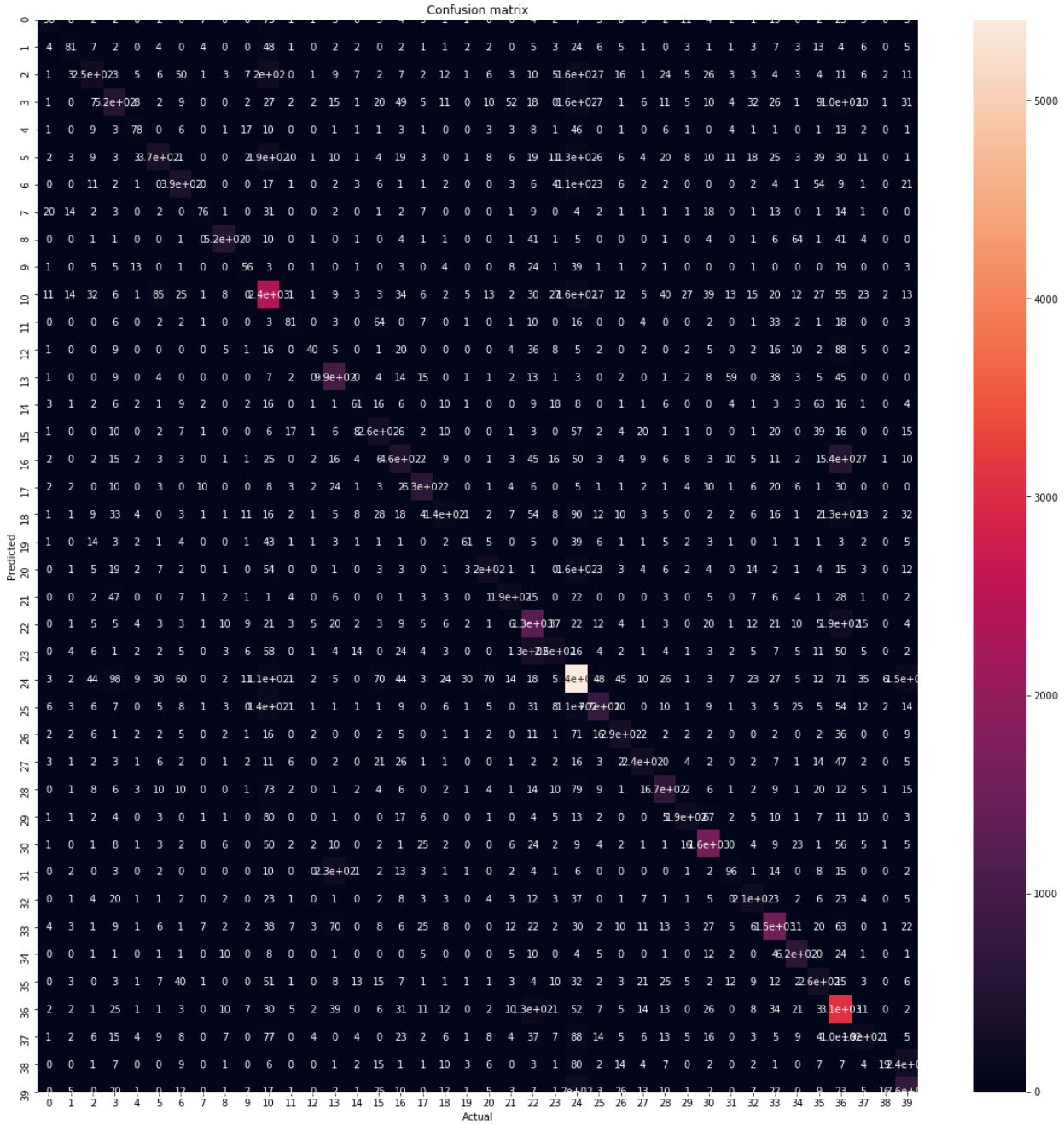
model	ValidationLoss	ValidationAccuracy	
0	CNN with GloVe embedding	1.499375	0.591576
1	LSTM without embedding	3.904507	0.312564
2	LSTM with keras embedding	1.539109	0.620489
3	LSTM with gensim Word2Vec embedding	1.257099	0.633119
4	LSTM + CNN with gensim Word2Vec embedding	1.379223	0.607708

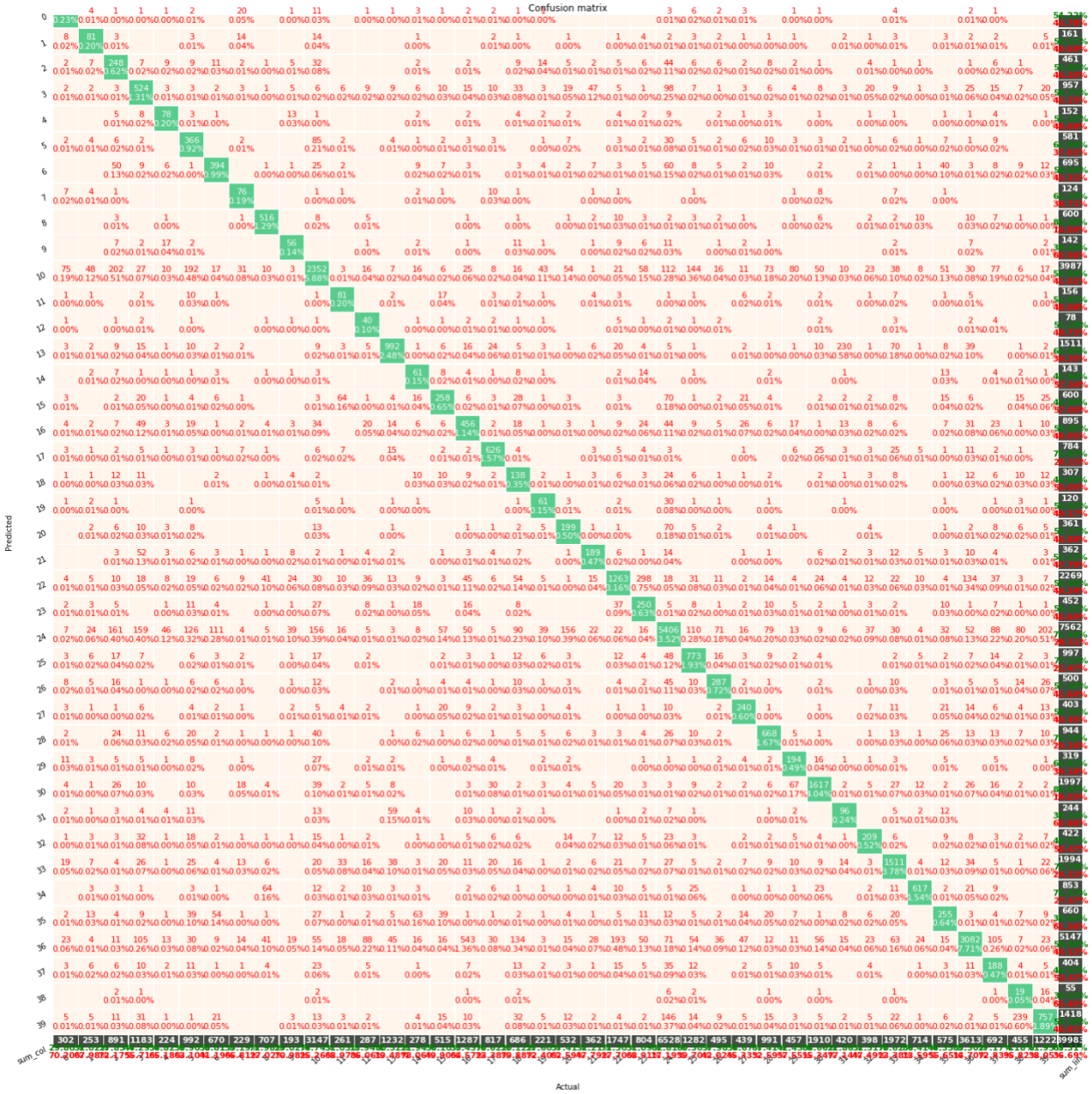
Plot models and metrics



Confusion Matrix - LSTM with gensim Word2Vec embedding

From the above data and plot we clearly see that LSTM model with gensim Word2Vec embedding has given the maximum accuracy (63.3%) and lowest loss (1.26%) compared to other implemented models. Let's take a look at the confusion matrix for the same.





Future Work

Build the recommendation system with combinations of different features including news headline, category, author and publishing date using the genism Word2Vec embedding.