

DigiSay Challenge

Sites Binary Classification

Mohamed Ahmed

Index

- Scraping sites
 - New Dataframe
- Preprocessing
- Dataset Visualization
- New Features May help?
- CountVectorizer & TfidfVectorizer
- Word tokenizer & padding sequences
- Pipeline and test split
- Imbalance Dataset

Index

- Which model we are going to use?
- Tune Hyperparameters with GridSearchCV
- SGD Classifier
- SVM Classifier
- XGBoost Classifier
- Word embedding's
- Bi-LSTM With Attention
 - RNN
 - LSTM
 - Bi-LSTM
 - Attention Layer

Index

- Bert
- Run Code
- Flask APP
- Notes

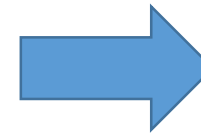
Scraping sites

- Using beautifulsoup to scrape each site and extract data from it
- Seem some sites give 404 error on online mode so we use the `page_source_path` not the domain link for this job
- Data collected from different sites and each one has it's own structure and style but all of this suppose to have same tags so beautifulsoap will extract text info from major tags like {title, h1, h2, h3,}
- But still major tags not give much info so will take the full text content of each site and will clean it and our New Dataframe will be like this

New Dataframe

- Dataframe transferred from one on left to right
 - Filling Nan alt_content_name from content
 - Filling Nan class with unrelated as most freq class (70%)
- As we can see there is many Nan values even in sites
- Major tags too
- But content seem pretty fine
- And content with nan will be repl
 - Tag <NONE>

```
link_id          0
page_source_path 0
link_url         0
link_domain_name 0
content_name     0
trans_content_name 5085
alt_content_names 851
campaign_name    0
number_of_episodes 0
class           10
dtype: int64
```



```
link_id          0
page_source_path 0
link_url         0
link_domain_name 0
content_name     0
trans_content_name 5085
alt_content_names 0
campaign_name    0
number_of_episodes 0
class           0
response_status  0
domain_name_status 0
content_name_freq 0
trans_content_name_freq 0
alt_content_names_0 214
alt_content_names_1 214
alt_content_names_2 214
alt_content_names_3 214
alt_content_names_4 214
campaign_name_freq 214
title            281
h1              1710
h2              2205
h3              4450
h4              5347
h5              6642
h6              6910
content         224
top_5_word      0
```

New Dataframe

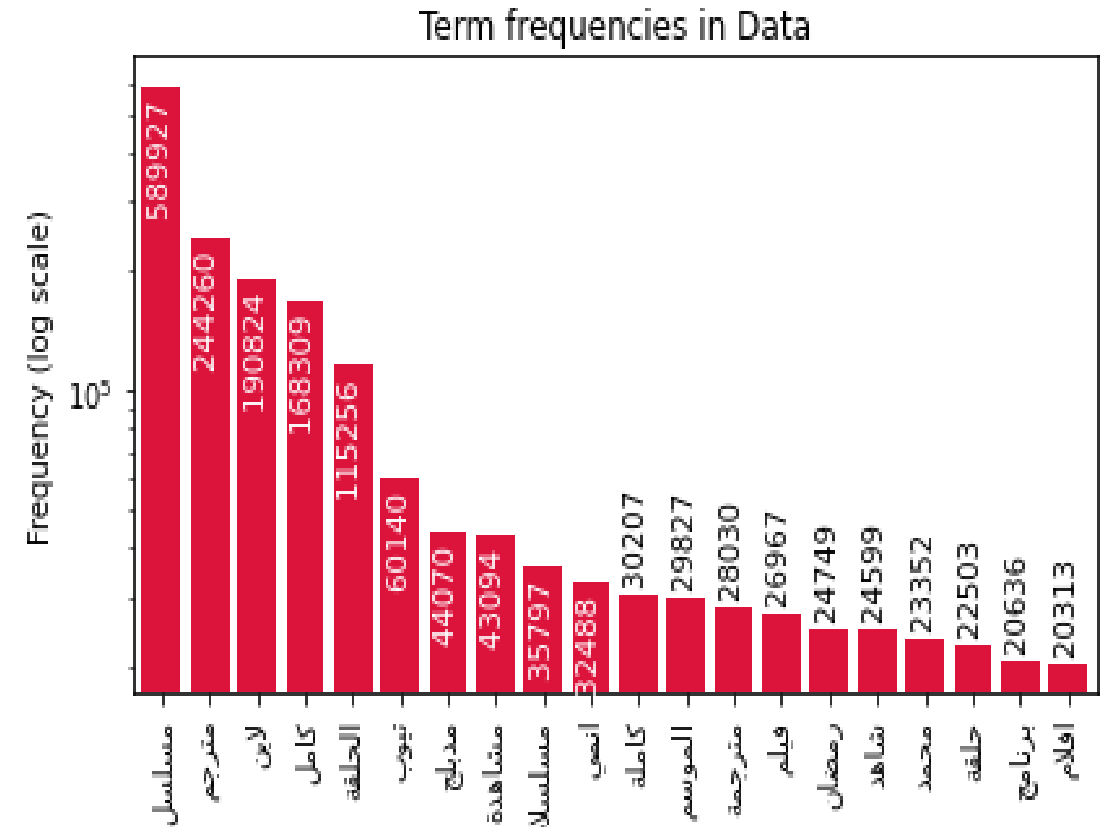
- response_status:-
 - response code on online mode
- domain_name_status:-
 - 0 or 1 if domain exist in page links
- content_name_freq:-
 - Number of occurrence of content_name
- trans_content_name_freq:
 - Number of occurrence of trans_content_name_freq
- alt_content_names_{i}:-
 - Number of occurrence for each alt content name
- campaign_name_freq:-
 - Number of occurrence for each alt content name

Preprocessing

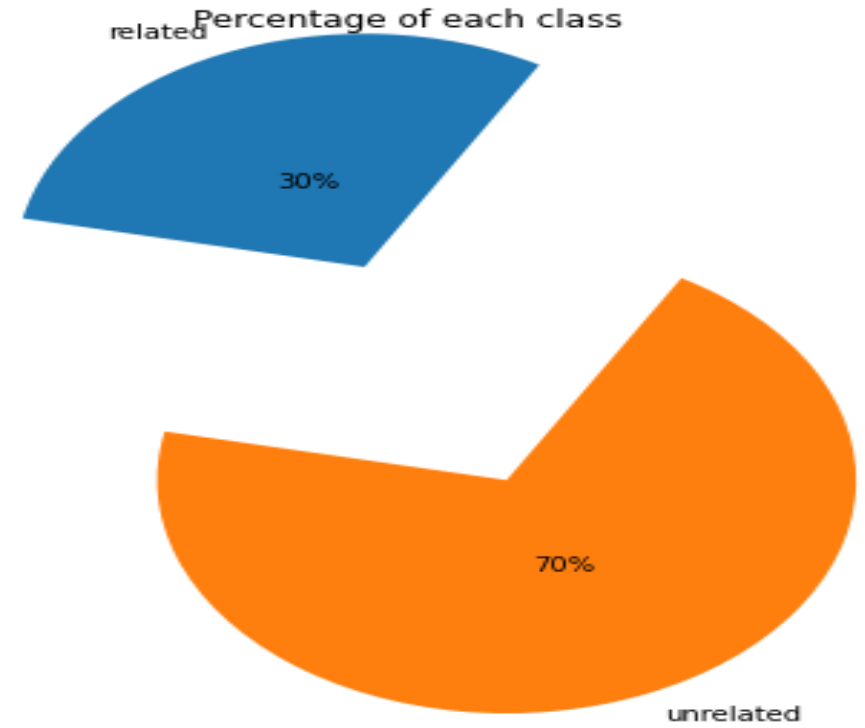
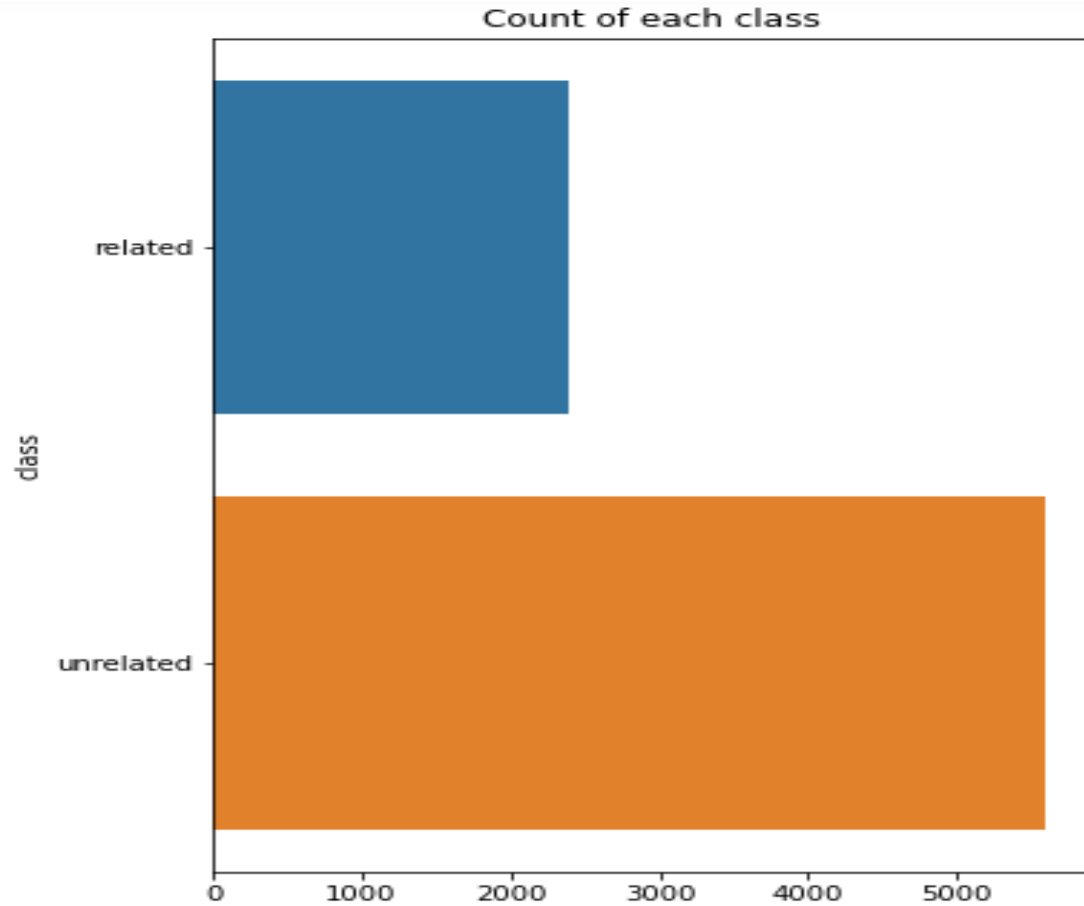
- As we are trying to find some patterns in content that represent each class we don't need to use lemmatization and stemming process some additional character or some words can represent useful info
- So preprocessing will follow these steps
 - Lower case the words
 - Remove emoji
 - Remove email
 - Remove accounts tag
 - Remove hashtag
 - Remove stop_words
 - Remove links
 - Remove HTML tags
 - Remove punctuation
 - Remove words contain numbers
 - Remove less 3 characters
 - Remove Numbers
 - Remove repeated spaces

Dataset Visualization

- Most Frequency word for all classes



Classes Distribution



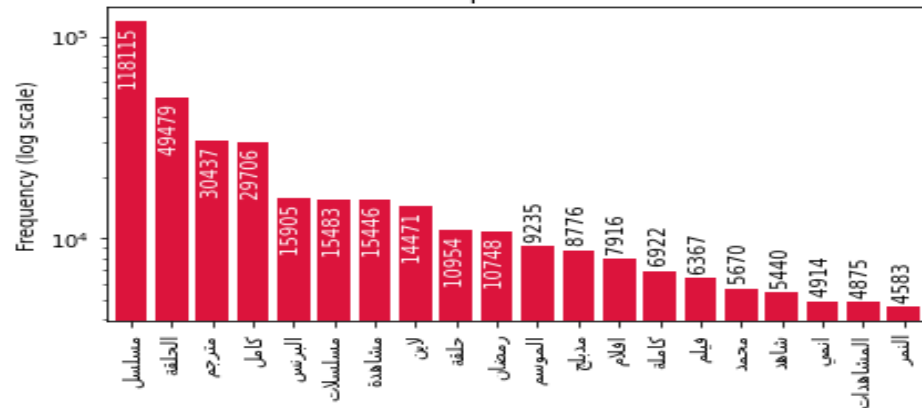
For each class Class

Related

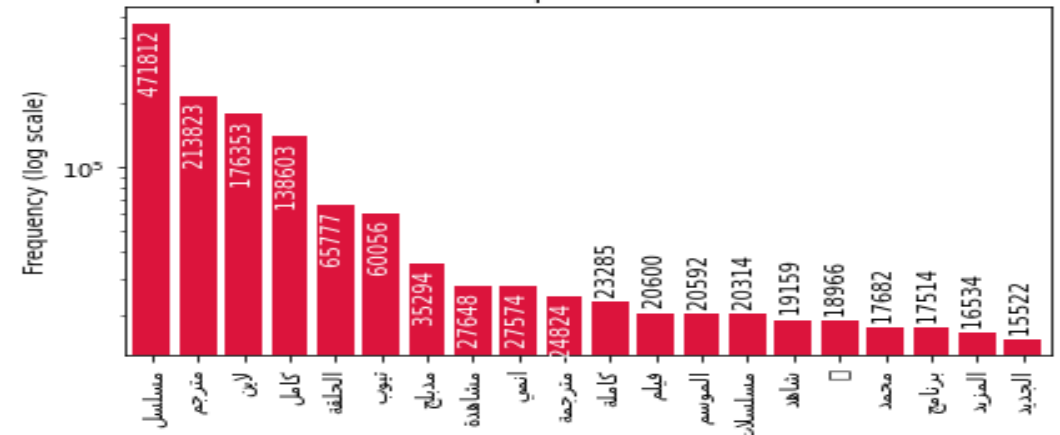
Unrelated



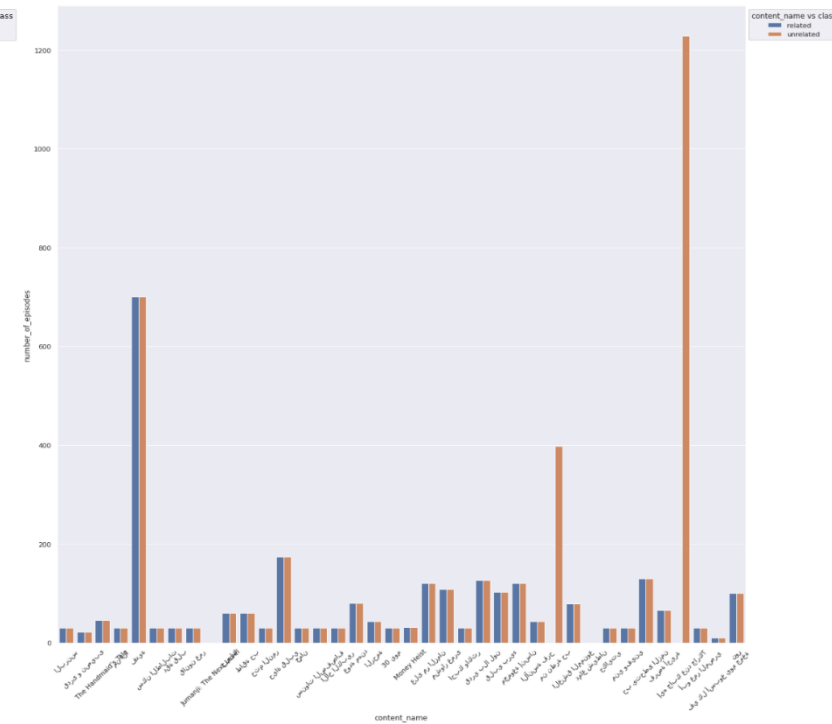
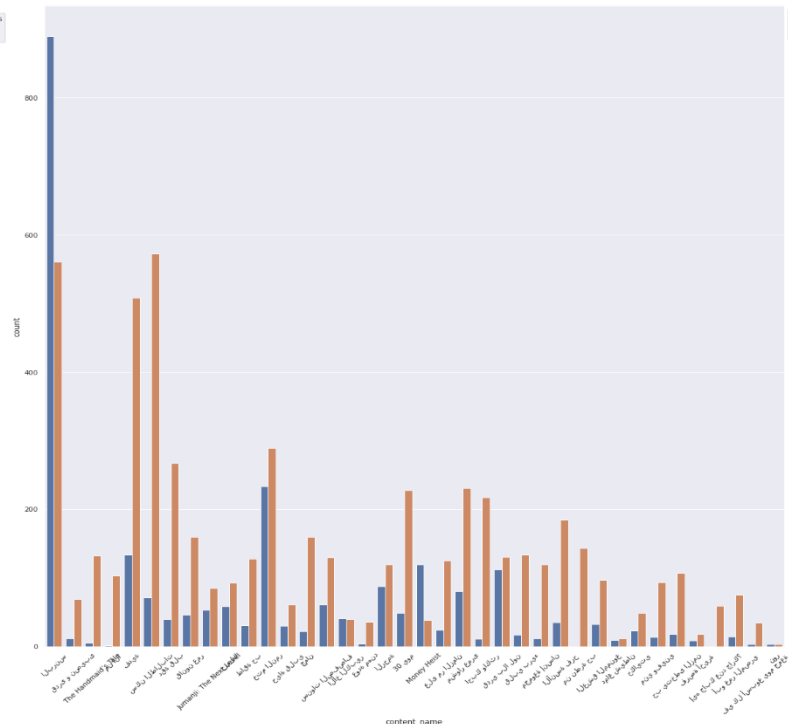
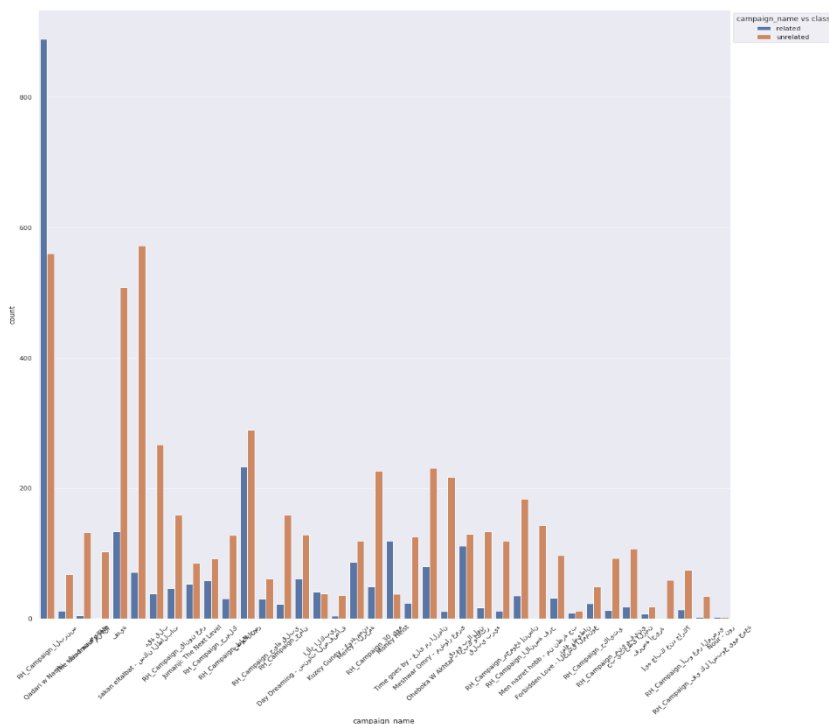
Term frequencies in Data



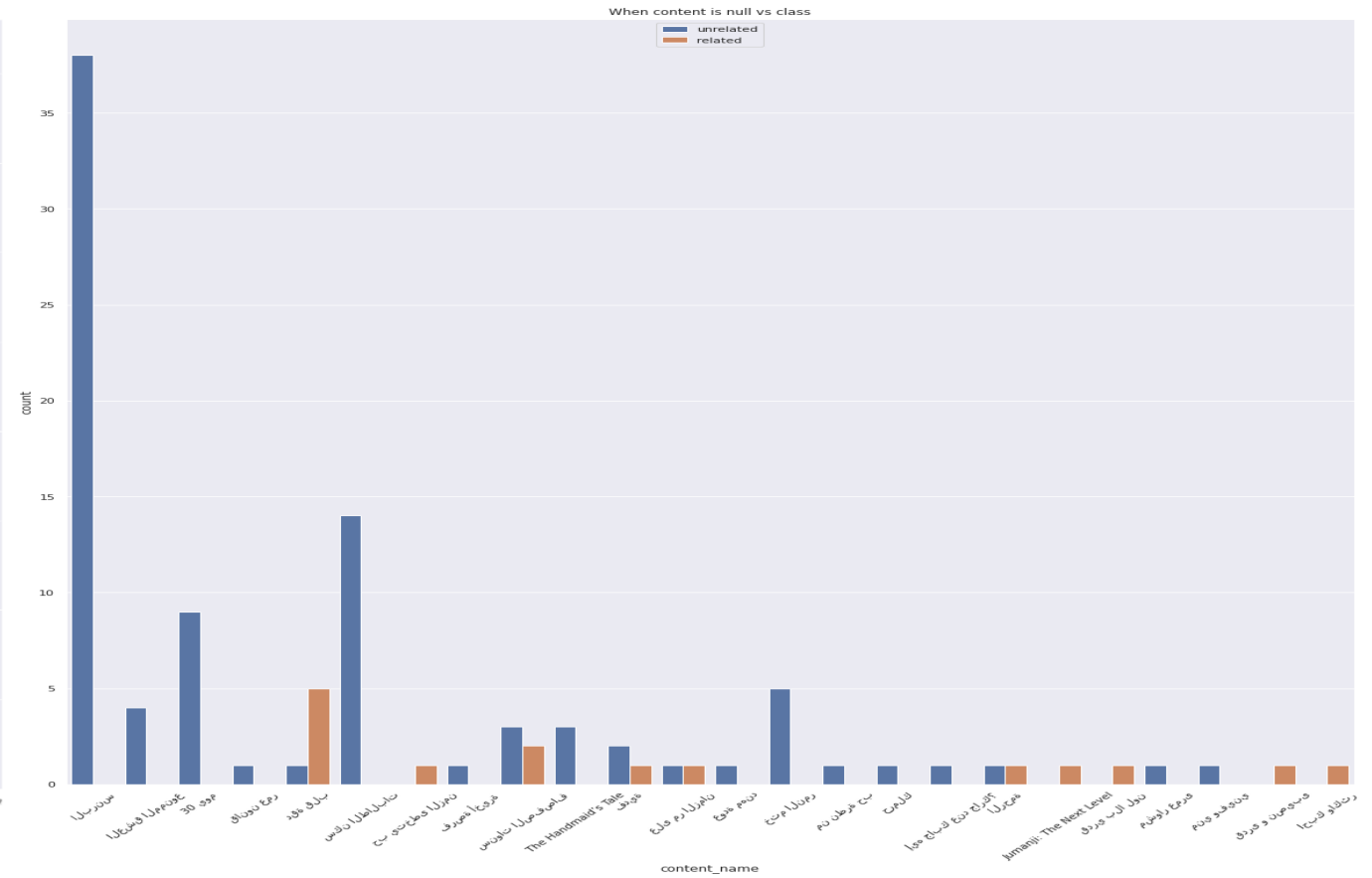
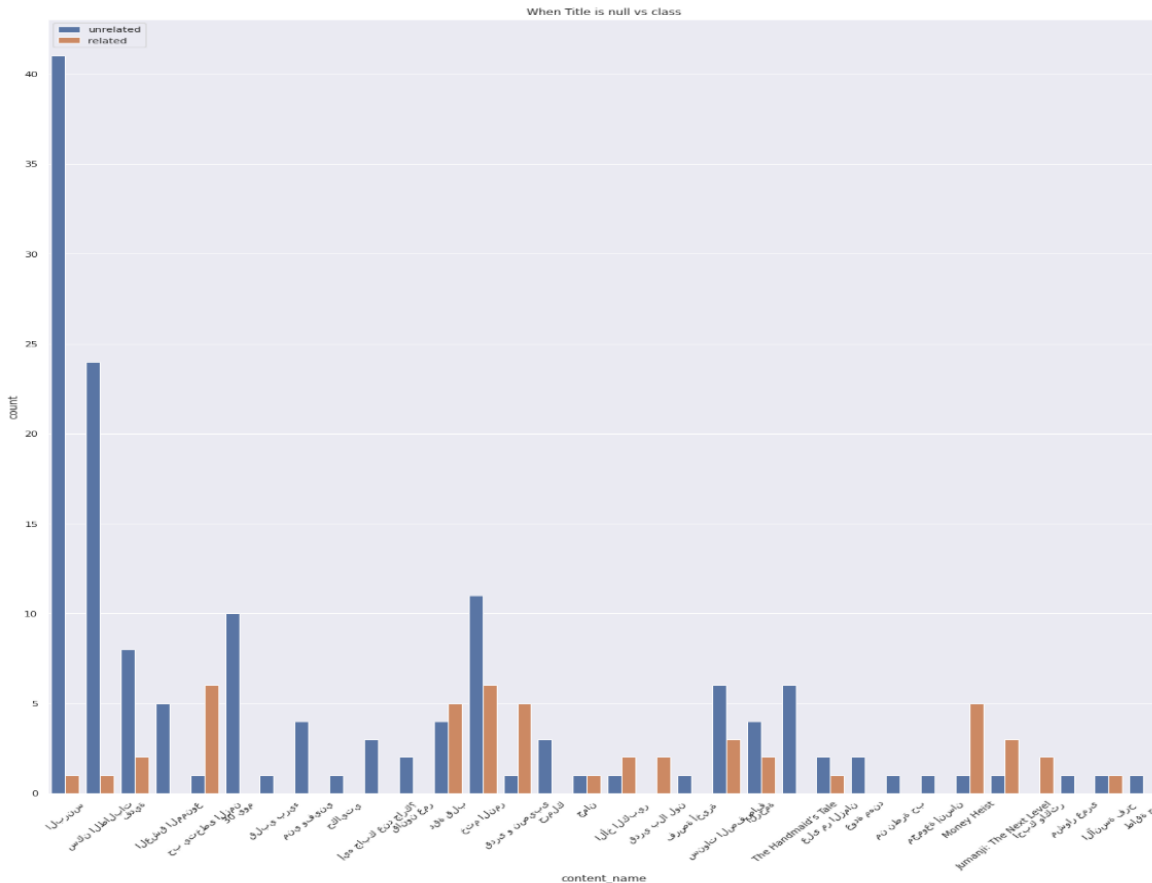
Term frequencies in Data



- We are not going to depend only on the content alone may be [content_name, compagin_name, number_eposides] may help lets find out.



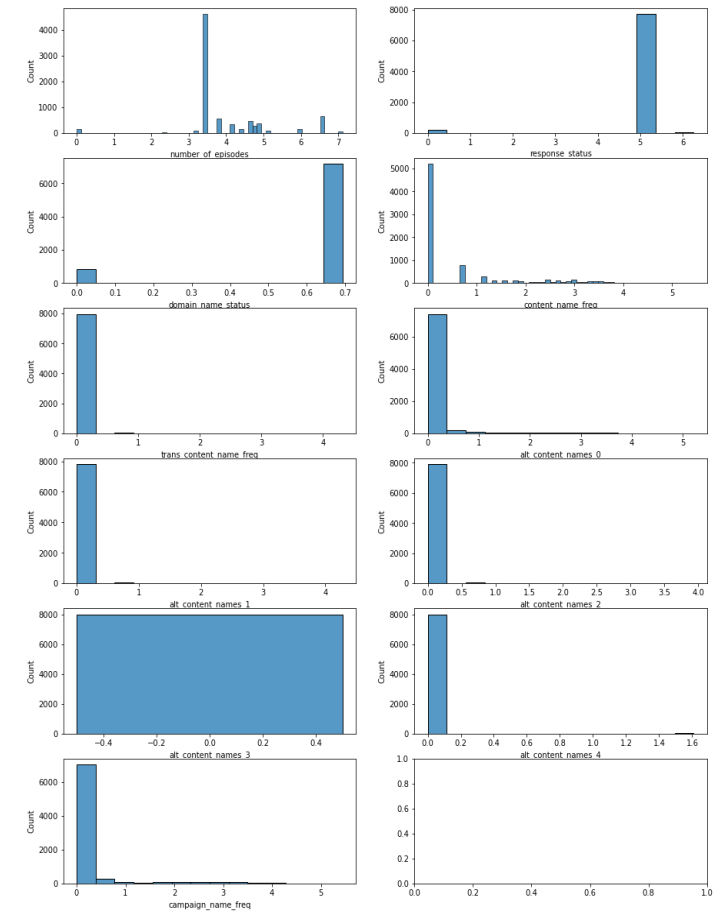
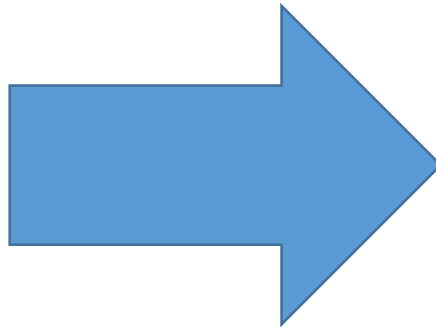
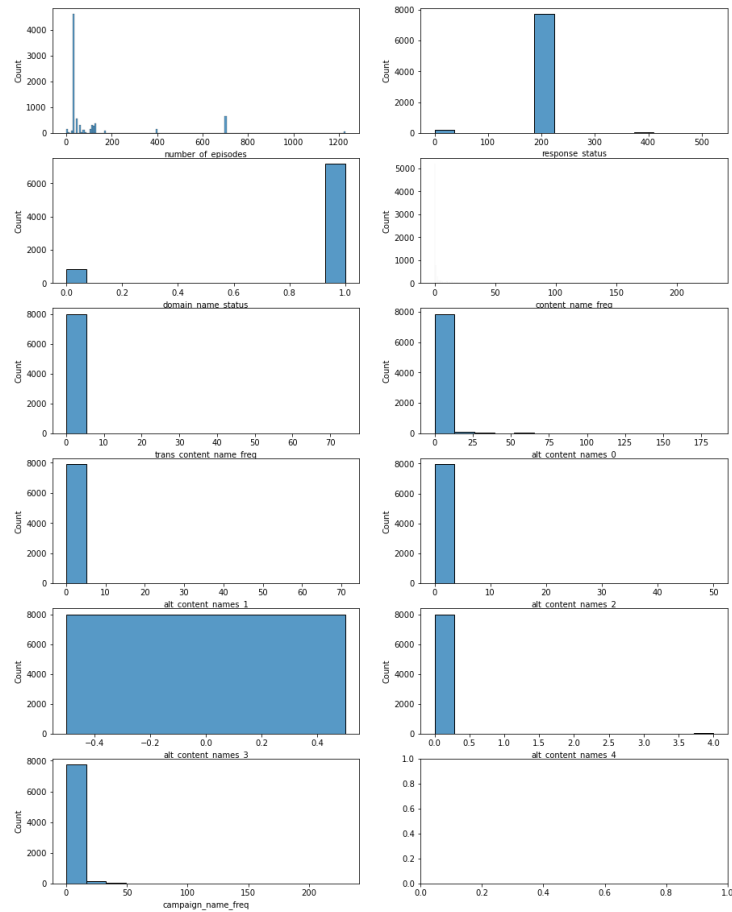
- Lets try to find this null values come from which class on title and content



New Dataframe Data May help?

- Can we build a simple model on what we get from previous features and see if it could help
- Numerical features
 - ['number_of_episodes', 'response_status', 'domain_name_status', 'content_name_freq', 'trans_content_name_freq', 'alt_content_names_0', 'alt_content_names_1', 'alt_content_names_2', 'alt_content_names_3', 'alt_content_names_4', 'campaign_name_freq']
 - Transformed using log+1 to over come zeros and normalize it
 - Scaled to be between 0 to 1
- Categorical like [content_name, compagin_name, response_status]
 - For each categorical we apply one hot encode and concatenate them horizontally

New Dataframe Data May help?



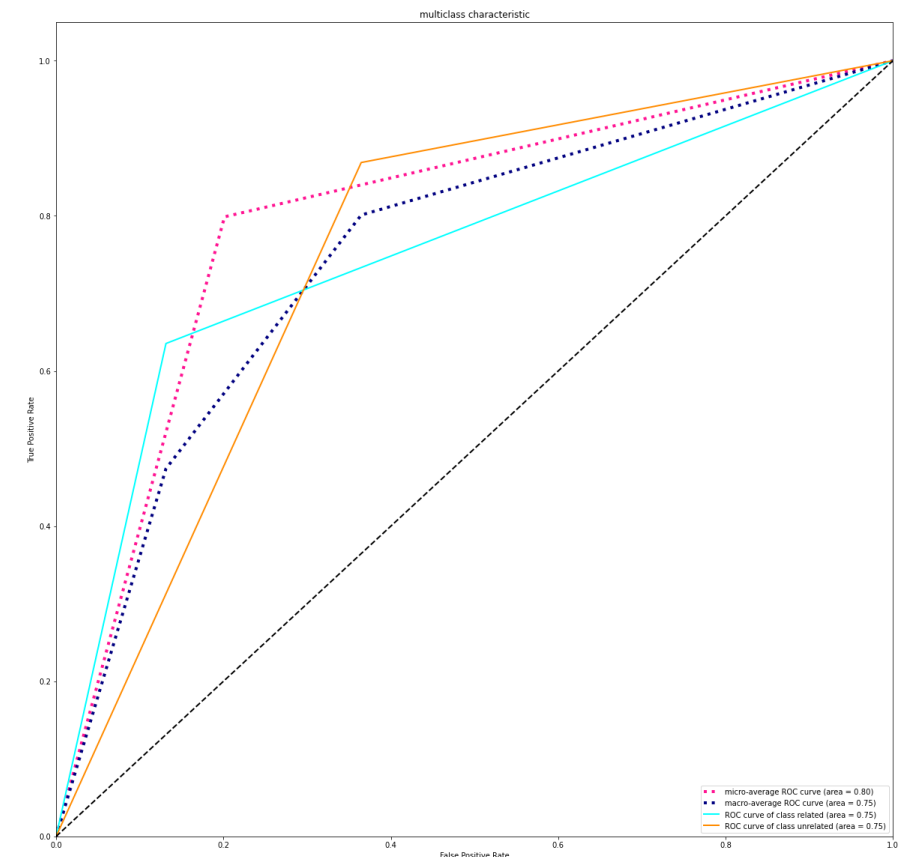
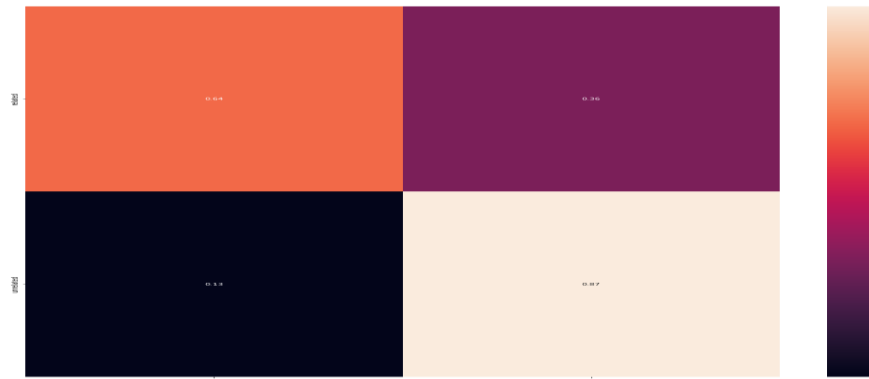
New Features May help?

- Lets try simple SGD and Gridsearch (we are going to take about them)
- splited the date to 80% train 20% test (validation part added later to train after making sure no overfit) (we are going to take about them)

New Features May help?

- Seem not so much but we will use it later
 - All from the data we created plus the hot encode old one

Classification Report				
	precision	recall	f1-score	support
related	0.67	0.64	0.65	480
unrelated	0.85	0.87	0.86	1120
accuracy			0.80	1600
macro avg	0.76	0.75	0.76	1600
weighted avg	0.80	0.80	0.80	1600



New Features May help?

- So the numerical and categorical features Not going to help much but it give a fair result will use it for good later.
- What about the content we collected and clean it
- We cant feed it to model as its is a sentences or words
- Lets see how we are going to transform it with some algorithms for NLP

CountVectorizer & TfidfVectorizer

- **CountVectorizer:-**

- Counts the frequency of all words in our corpus, sorts them and grabs the most recurring features (using `max_features` hyperparameter). But these results are mostly biased and our model might lose out on some of the important less frequent features. These are all boolean values. Ex. SEO People used to take advantage of this.

- **TfidfVectorizer:-**

- TFIDF is a statistical measure said to have fixed the issues with CountVectorizer in some way. It consists of 2 parts, TF (Term Frequency) multiplied with IDF (Inverse Document Frequency). The main intuition being some words that appear frequently in 1 document and less frequently in other documents could be considered as providing extra insight for that 1 document and could help our model learn from this additional piece of information. In short, common words are penalized. These are relative frequencies identified as floating point numbers.

Word tokenizer & padding sequences

- **Tokenization**

- is one of the most common tasks when it comes to working with text data, Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

- **Padding sequences**

- Used to ensure that all sequences in a list have the same length. By default this is done by padding 0 in the beginning of each sequence until each sequence has the same length as the longest sequence
longest sequence will be 535446 is the longest splitted statement form saved csv file, so big to fill it in memory even it is in sparse shape so we will reduce it to max 5000

Pipeline and test split

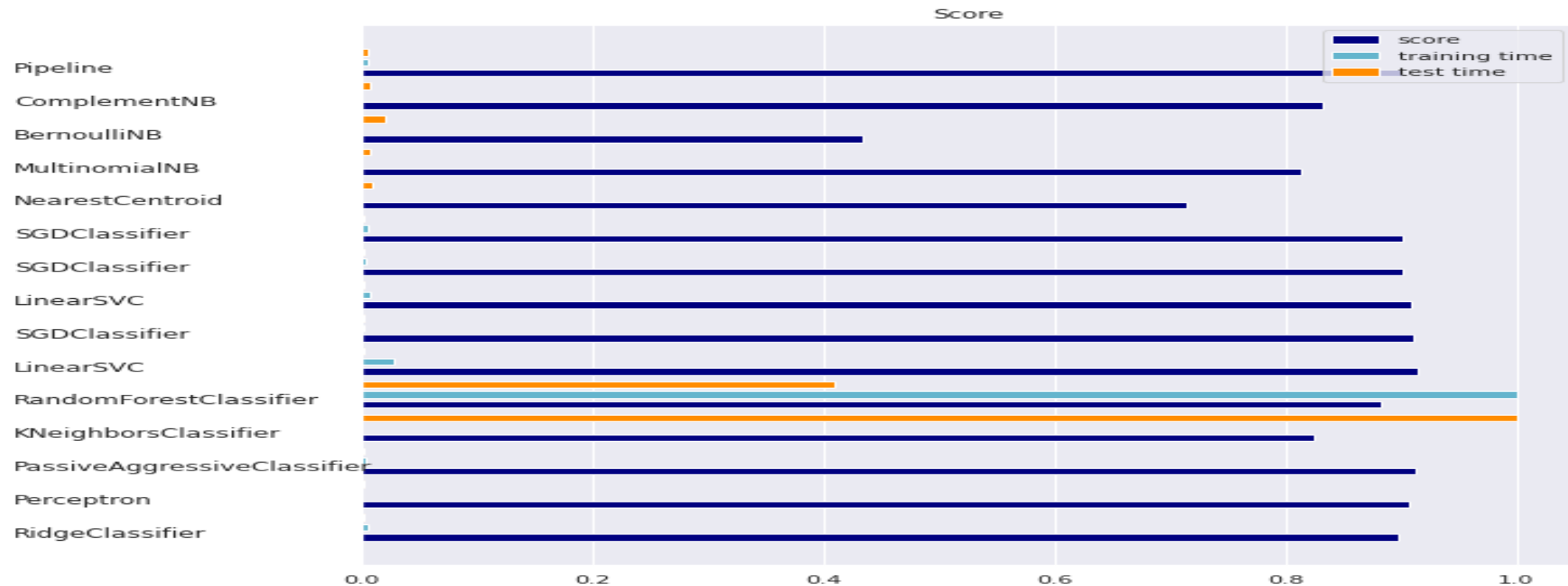
- **Train test split:-**
 - Data will be splitted 0.2 for test and rest for train with stratify option
- **Pipeline:-**
 - CountVectorizer and TfidfVectorizer:-
 - will be saved in one single pipeline as pkl file
 - Tokenizer:-
 - will be saved in one single pipeline as pkl file
- **Label encoder:-**
 - Representation for each class as a single unique number, will be saved in one single pipeline as pkl file
- **Model parameters:-**
 - Vocab size, max sentence length, number of classes all will be in a dictionary format will be saved in json file

Imbalance Dataset

- As we can see our data distributed between 2 classes 70% to unrelated and 30% to related, seem imbalance one and we need to over come this problem, As Most machine learning algorithms are not very useful with biased class data. But, we can modify the current training algorithm to take into account the skewed distribution of the classes. This can be achieved by giving different weights to both the majority and minority classes. The difference in weights will influence the classification of the classes during the training phase. The whole purpose is to penalize the misclassification made by the minority class by setting a higher class weight and at the same time reducing weight for the majority class.
- Use class weights when you have an imbalanced dataset and want to improve single-label classification results
- Class weights give all the classes equal importance on gradient updates, on average, regardless of how many samples we have from each class in the training data. This prevents models from predicting the more frequent class more often just because it's more common.

Which model we are going to use?

- As I decided before what I am going to use (non linear, supervised classification model, robust to noise) but lets see benchmark models first and the maximum accuracy we can get



Which model we are going to use?

- Seem SGD, Linear SVM Give the highest score
- But as you can see SGD applied many times (actually it was fastest one too try it many times) each one give different score
- Each one was applied with different Hyper parameters
- So lets find how we will find this perfect hyper parameters

Which model we are going to use?

- And what about the 3 features we tried before?
- Lets concatenate them to the Tdi-idf features and feed them to the model so our input shape will be
 - 5000 features form tdf-idf
 - 76 features as sparse matrix from content_name & compagin_name, response States
 - 11 features from Numerical Features
- In total each sample will be 5087 features for upcoming three models

Tune Hyperparameters with GridSearchCV

- Grid Search, cross-validation is also performed. Cross-Validation is used while training the model. As we know that before training the model with data, we divide the data into two parts – train data and test data. In cross-validation, the process divides the train data further into two parts – the train data and the validation data.
- The most popular type of Cross-validation is K-fold Cross-Validation.
- It is an iterative process that divides the train data into k partitions. Each iteration keeps one partition for testing and the remaining k-1 partitions for training the model. The next iteration will set the next partition as test data and the remaining k-1 as train data and so on.
- In each iteration, it will record the performance of the model and at the end give the average of all the performance. Thus, it is also a time-consuming process.
- Thus, GridSearch along with cross-validation takes huge time cumulatively to evaluate the best hyperparameters.
- GridSearchCV can be used on several hyperparameters to get the best values for the specified hyperparameters.
 - Estimator:- the model
 - param_grid:- A dictionary with parameter names as keys and lists of parameter values.
 - Scoring:- The performance measure. For example, 'r2' for regression models, 'precision' for classification models.
 - Cv:- An integer that is the number of folds for K-fold cross-validation.

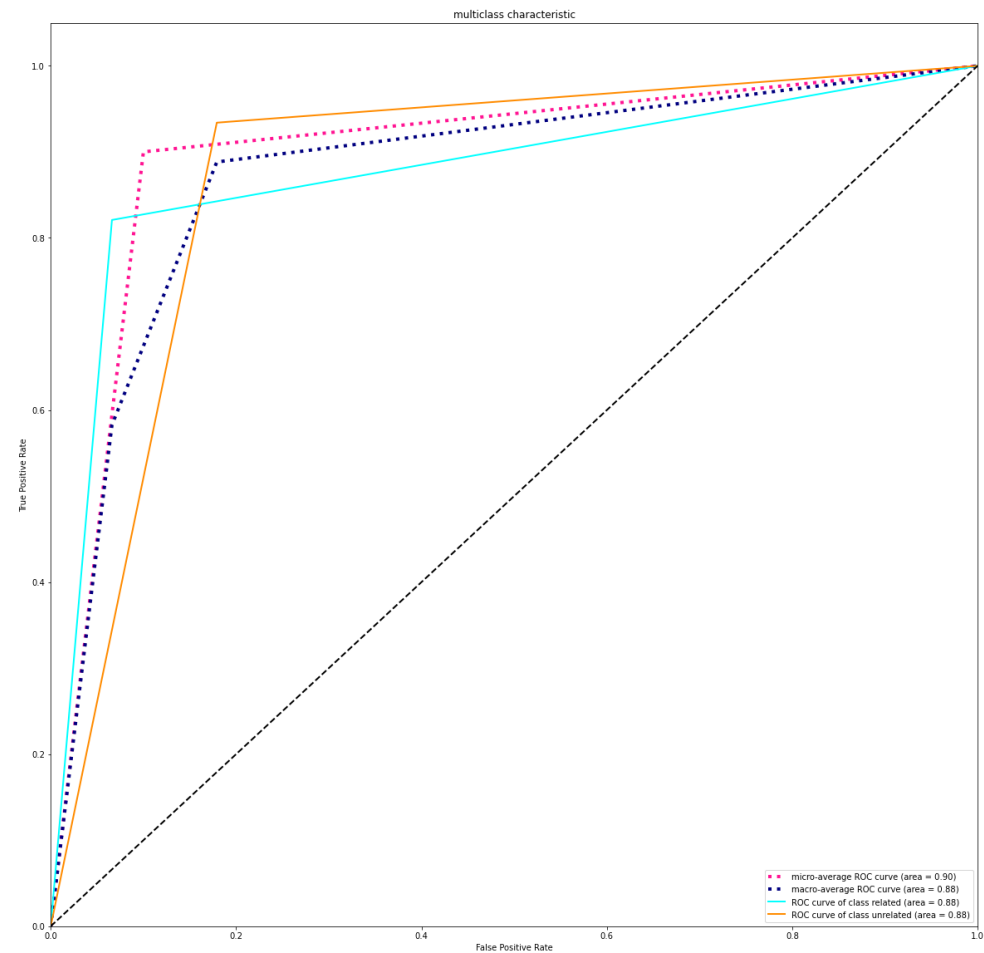
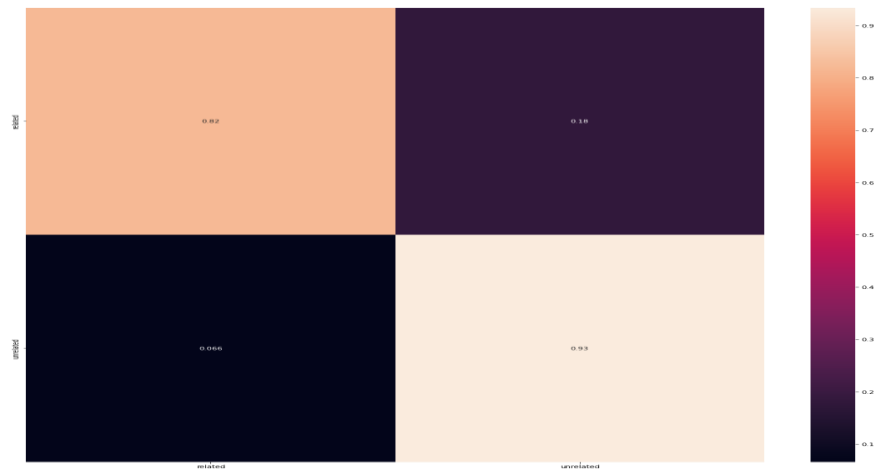
SGD Classifier

- SGD Classifier one vs all estimator, implements regularised linear models with Stochastic Gradient Descent.
- SGD is a This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning, see the `partial_fit` method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.
- This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the `loss` parameter; by default, it fits a linear support vector machine (SVM).
- The regularizer is a penalty added to the loss function that shrinks model parameters towards the zero vector using either the squared euclidean norm L2 or the absolute norm L1 or a combination of both (Elastic Net). If the parameter update crosses the 0.0 value because of the regularizer, the update is truncated to 0.0 to allow for learning sparse models and achieve online feature selection.
- `penalty`:-
 - 'none', 'l2', 'l1', or 'elasticnet', The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM models. 'l1' and 'elasticnet' might bring sparsity to the model (feature selection) not achievable with 'l2'.
- `alpha`:-
 - Constant that multiplies the regularization term. Defaults to 0.0001 Also used to compute `learning_rate` when set to 'optimal'.
- `max_iter`:-
 - The max number of epochs over the training data

SGD Results

- Results on Testset

Classification Report				
	precision	recall	f1-score	support
related	0.84	0.82	0.83	480
unrelated	0.92	0.93	0.93	1120
accuracy			0.90	1600
macro avg	0.88	0.88	0.88	1600
weighted avg	0.90	0.90	0.90	1600



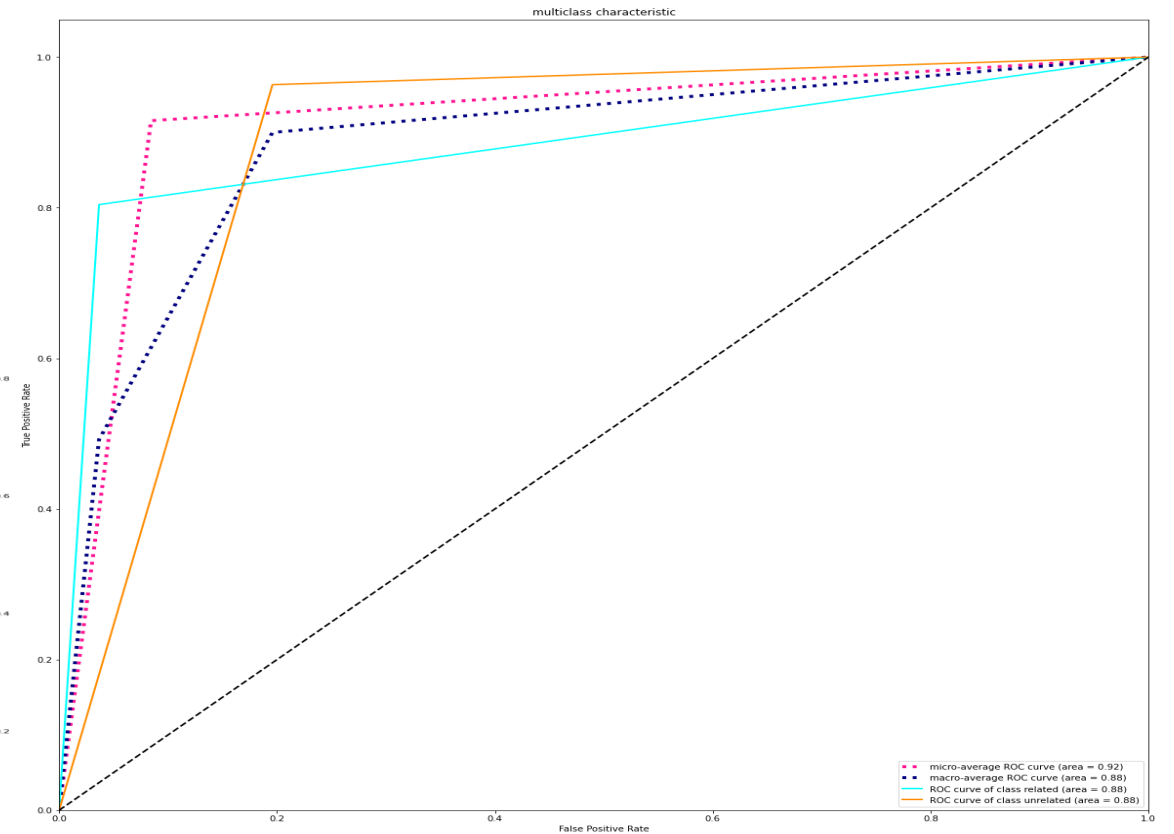
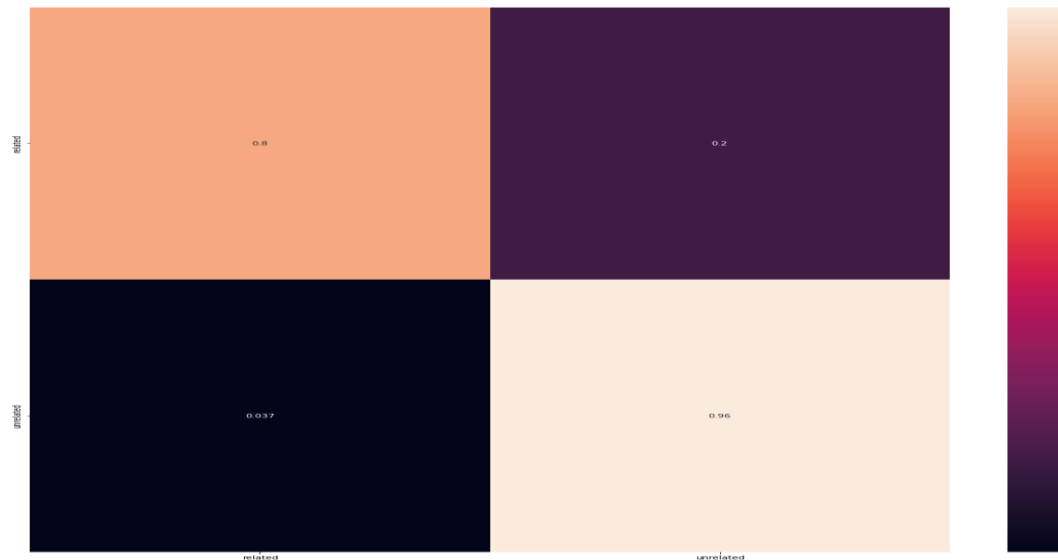
SVM Classifier

- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N the number of features) that distinctly classifies the data points
- To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.
- Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes.
- The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.
- C:-
 - Penalty parameter C of the error term.
- kernel:-
 - Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).
- degree:-
 - Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

SVM Results

- Results on Testset
 - by the way class weights did not apply here to see it effect on recall part

Classification Report				
	precision	recall	f1-score	support
related	0.90	0.80	0.85	480
unrelated	0.92	0.96	0.94	1120
accuracy			0.92	1600
macro avg	0.91	0.88	0.90	1600
weighted avg	0.92	0.92	0.91	1600



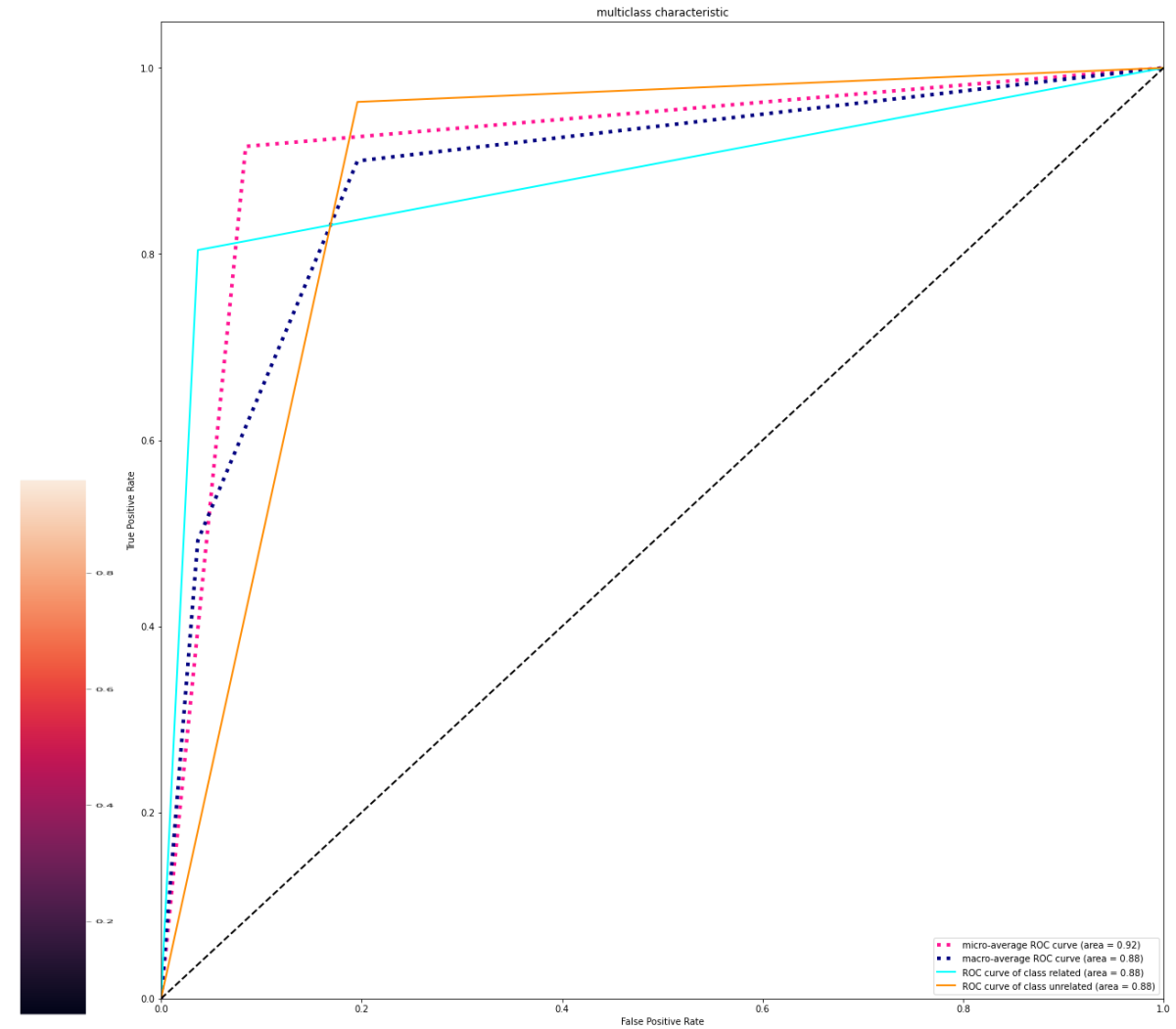
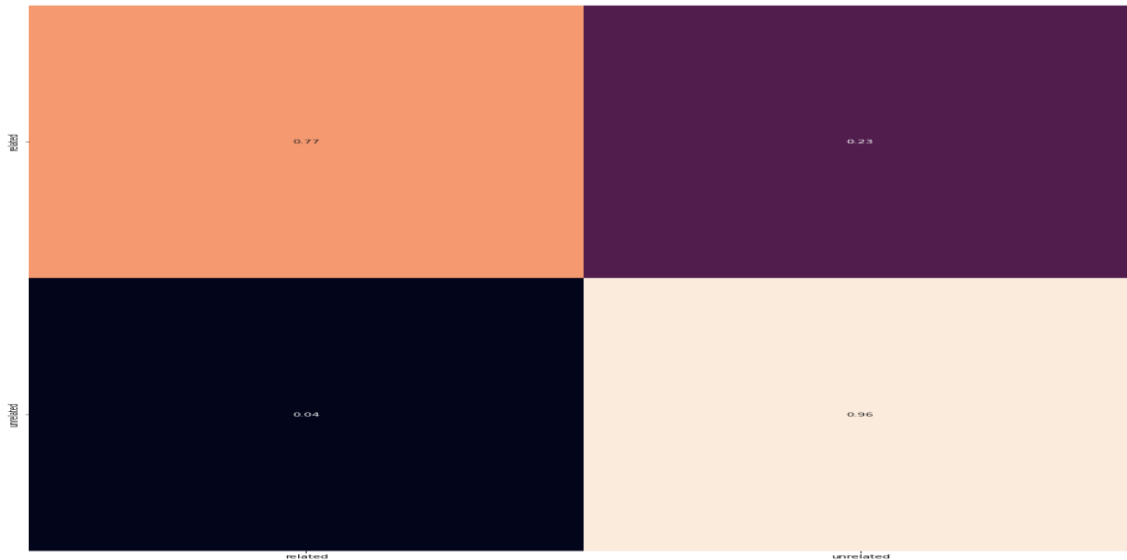
XGBoost

- XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.
- It's vital to an understanding of XGBoost to first grasp the machine learning concepts and algorithms that XGBoost builds upon: supervised machine learning, decision trees, ensemble learning, and gradient boosting.
- Supervised machine learning uses algorithms to train a model to find patterns in a dataset with labels and features and then uses the trained model to predict the labels on a new dataset's features.
- Decision trees create a model that predicts the label by evaluating a tree of if-then-else true/false feature questions, and estimating the minimum number of questions needed to assess the probability of making a correct decision. Decision trees can be used for classification to predict a category, or regression to predict a continuous numeric value. In the simple example below, a decision tree is used to estimate a house price (the label) based on the size and number of bedrooms (the features).
- A Gradient Boosting Decision Trees (GBDT) is a decision tree ensemble learning algorithm similar to random forest, for classification and regression. Ensemble learning algorithms combine multiple machine learning algorithms to obtain a better model.
- Both random forest and GBDT build a model consisting of multiple decision trees. The difference is in how the trees are built and combined
- `learning_rate`:-
 - The learning rate is the shrinkage you do at every step you are making. If you make 1 step at $\eta = 1.00$, the step weight is 1.00
- `max_depth`:-
 - The maximum depth of a tree, same as GBM.
 - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- `n_estimators`:-
 - The number of trees (or rounds) in an XGBoost model is specified to the `XGBClassifier` or `XGBRegressor` class

XGBoost Results

- Results on Testset

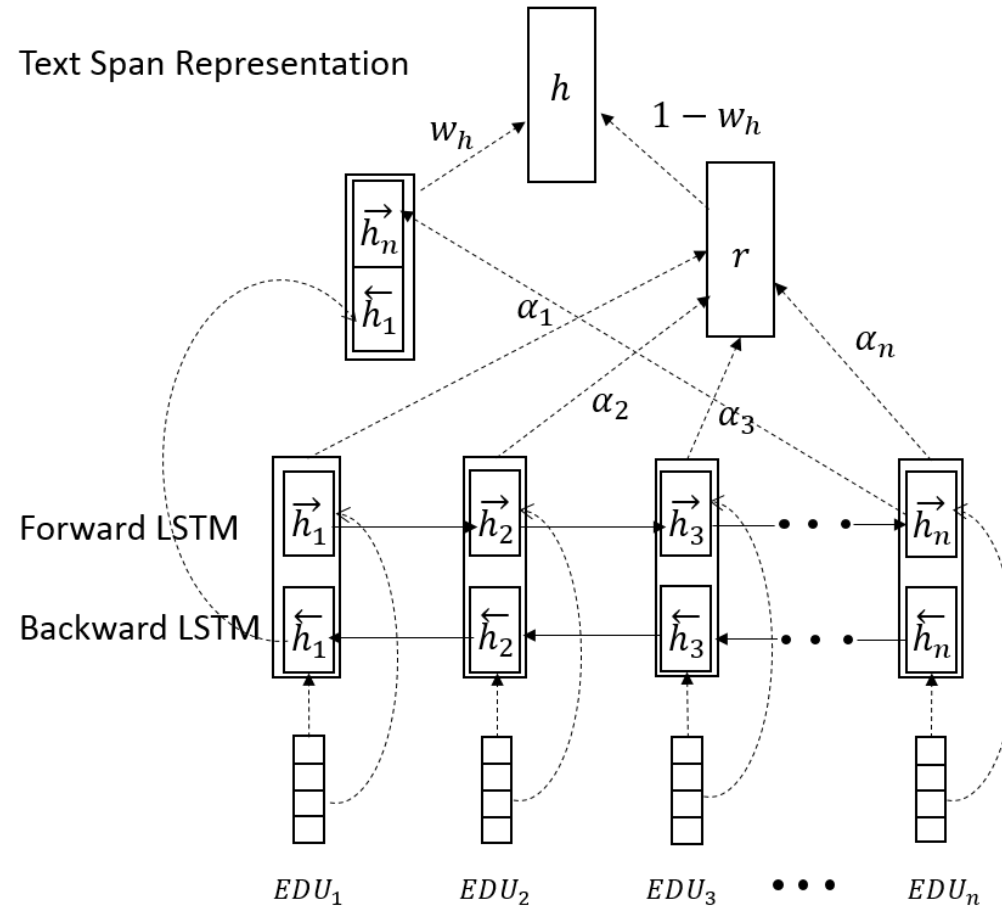
Classification Report				
	precision	recall	f1-score	support
related	0.89	0.77	0.82	480
unrelated	0.91	0.96	0.93	1120
accuracy			0.90	1600
macro avg	0.90	0.86	0.88	1600
weighted avg	0.90	0.90	0.90	1600



Word embedding's

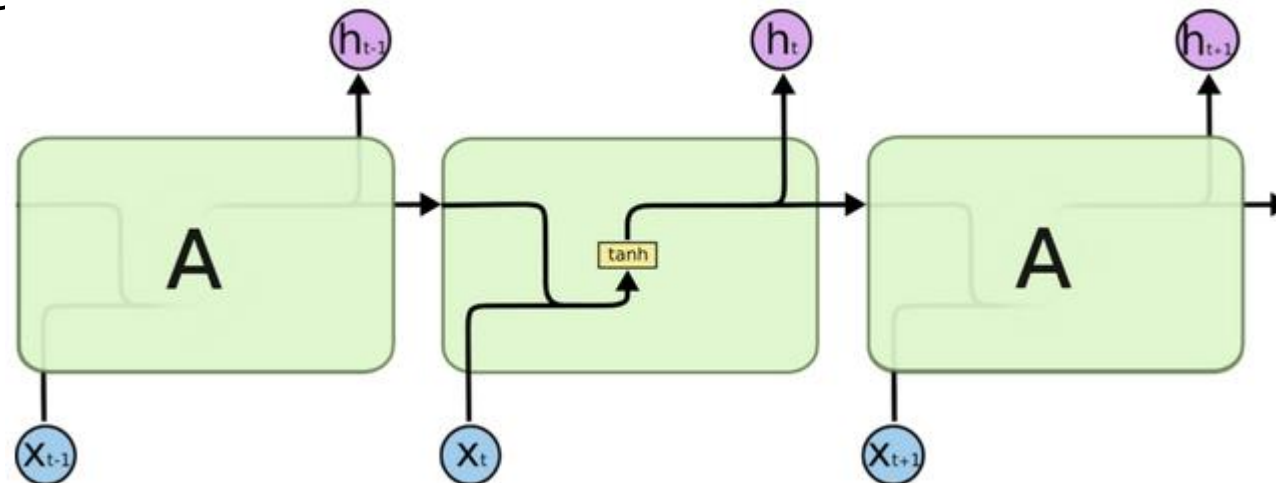
- Word embeddings can be thought of as an alternate to one-hot encoding along with dimensionality reduction.
- As we know while dealing with textual data, we need to convert it into numbers before feeding into any machine learning model, including neural networks. For simplicity words can be compared to categorical variables. We use one-hot encoding to convert categorical features into numbers. To do so, we create dummy features for each of the category and populate them with 0's and 1's.
- Similarly if we use one-hot encoding on words in textual data, we will have a dummy feature for each word, which means 10,000 features for a vocabulary of 10,000 words. This is not a feasible embedding approach as it demands large storage space for the word vectors and reduces model efficiency.
- Embedding layer enables us to convert each word into a fixed length vector of defined size. The resultant vector is a dense one with having real values instead of just 0's and 1's. The fixed length of word vectors helps us to represent words in a better way along with reduced dimensions.

Bi-LSTM With Attention



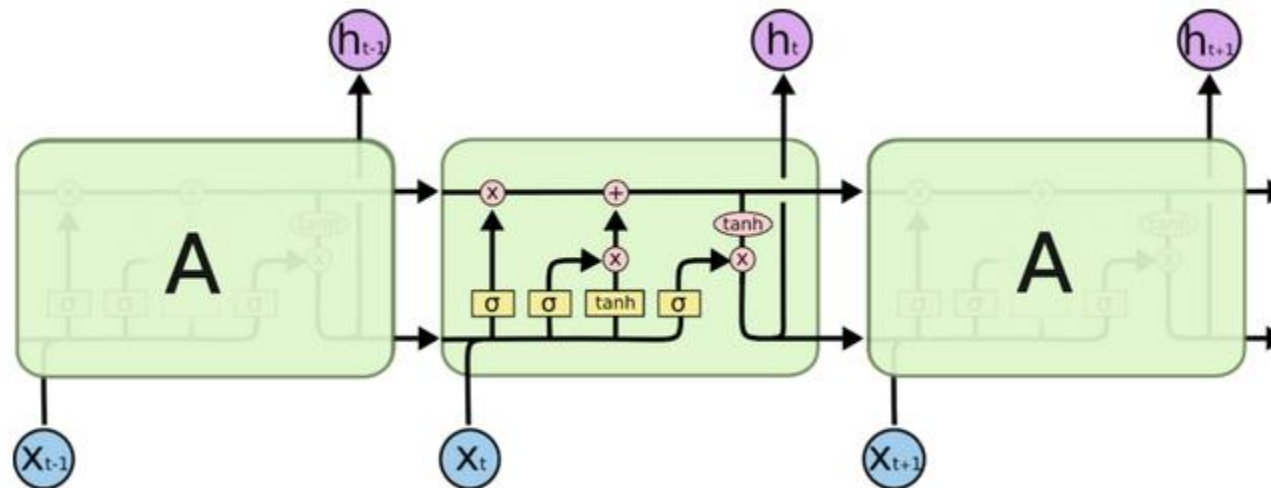
RNN

- RNN(recurrent neural network) is a type of neural network that we use to develop speech recognition and natural language processing models. Recurrent neural networks remember the sequence of the data and use data patterns to give the prediction.
- RNN uses feedback loops which makes it different from other neural networks. Those loops help RNN to process the sequence of the data. This loop allows the data to be shared to different nodes and predictions according to the gathered information. This process can be called memory.
- RNN and the loops create the networks that allow RNN to share information, and also, the loop structure allows the neural network to take the sequence of input data. RNN converts an index



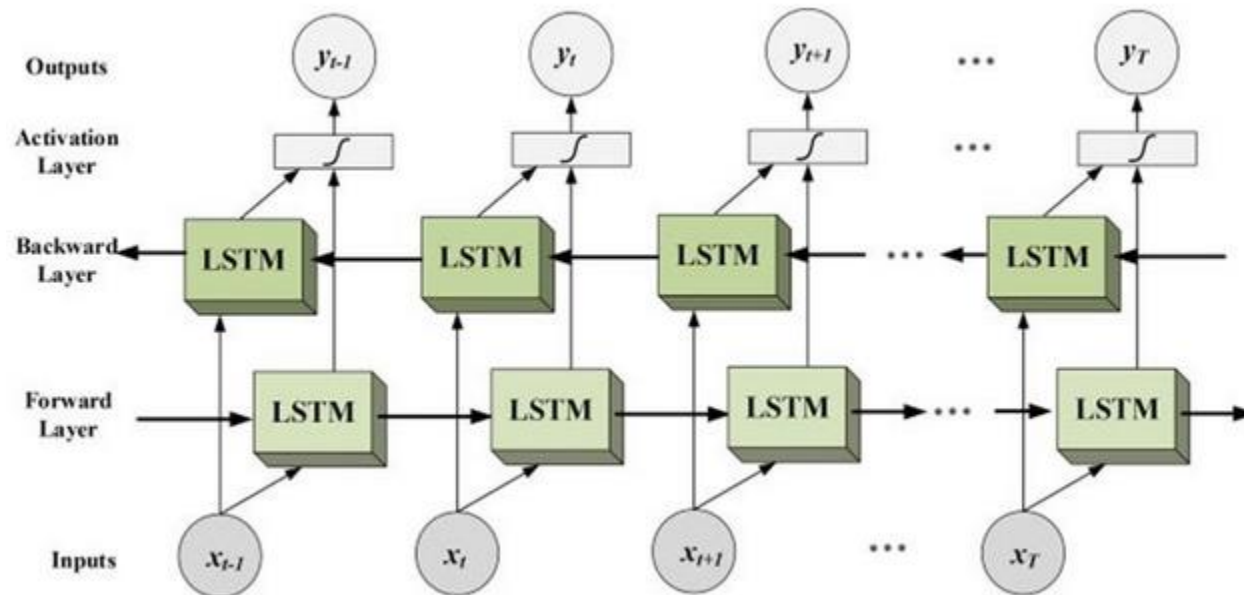
LSTM

- Long short term memory networks, usually called LSTM – are a special kind of RNN. They were introduced to avoid the long-term dependency problem. In regular RNN, the problem frequently occurs when connecting previous information to new information. If RNN could do this, they'd be very useful. This problem is called long-term dependency.
- To remember the information for long periods in the default behaviour of the LSTM. LSTM networks have a similar structure to the RNN, but the memory module or repeating module has a different LSTM. The block diagram of the repeating module will look like the image below.



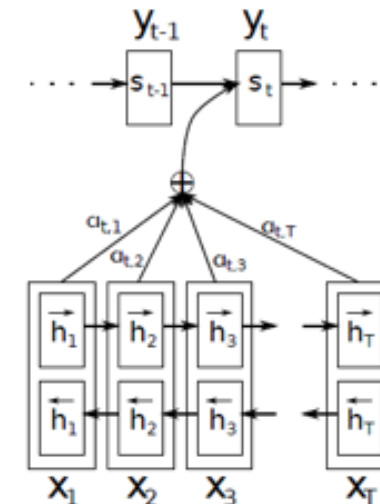
Bi-LSTM

- Bidirectional long-short term memory(bi-lstm) is the process of making any neural network o have the sequence information in both directions backwards (future to past) or forward(past to future).
- In bidirectional, our input flows in two directions, making a bi-lstm different from the regular LSTM. With the regular LSTM, we can make input flow in one direction, either backwards or forward. However, in bi-directional, we can make the input flow in both directions to preserve the future and the past information.



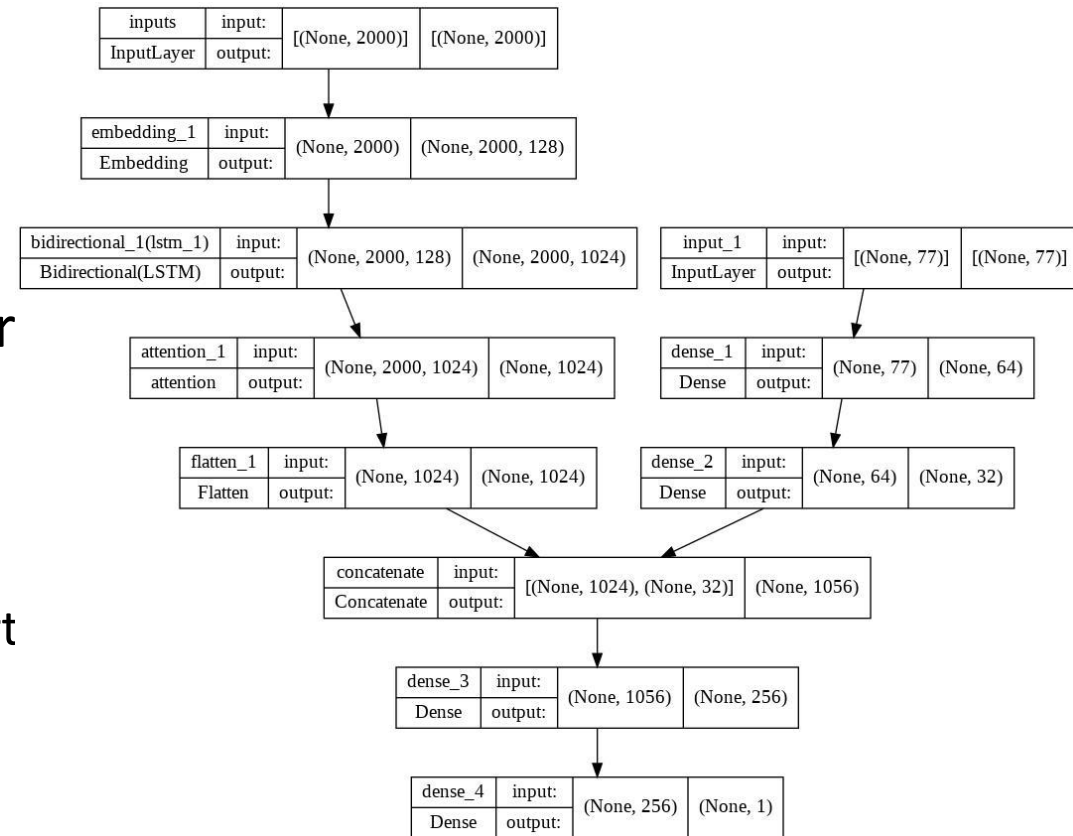
Attention Layer

- A mechanism that can help a neural network to memorize long sequences of the information or data can be considered as the attention mechanism and broadly it is used in the case of Neural machine translation(NMT). As we have discussed in the above section, the encoder compresses the sequential input and processes the input in the form of a context vector. We can introduce an attention mechanism to create a shortcut between the entire input and the context vector where the weights of the shortcut connection can be changeable for every output.
- Because of the connection between input and context vector, the context vector can have access to the entire input, and the problem of forgetting long sequences can be resolved to an extent. Using the attention mechanism in a network, a context vector can have the following information:
 - Encoder hidden states;
 - Decoder hidden states;
 - Alignment between source and target.



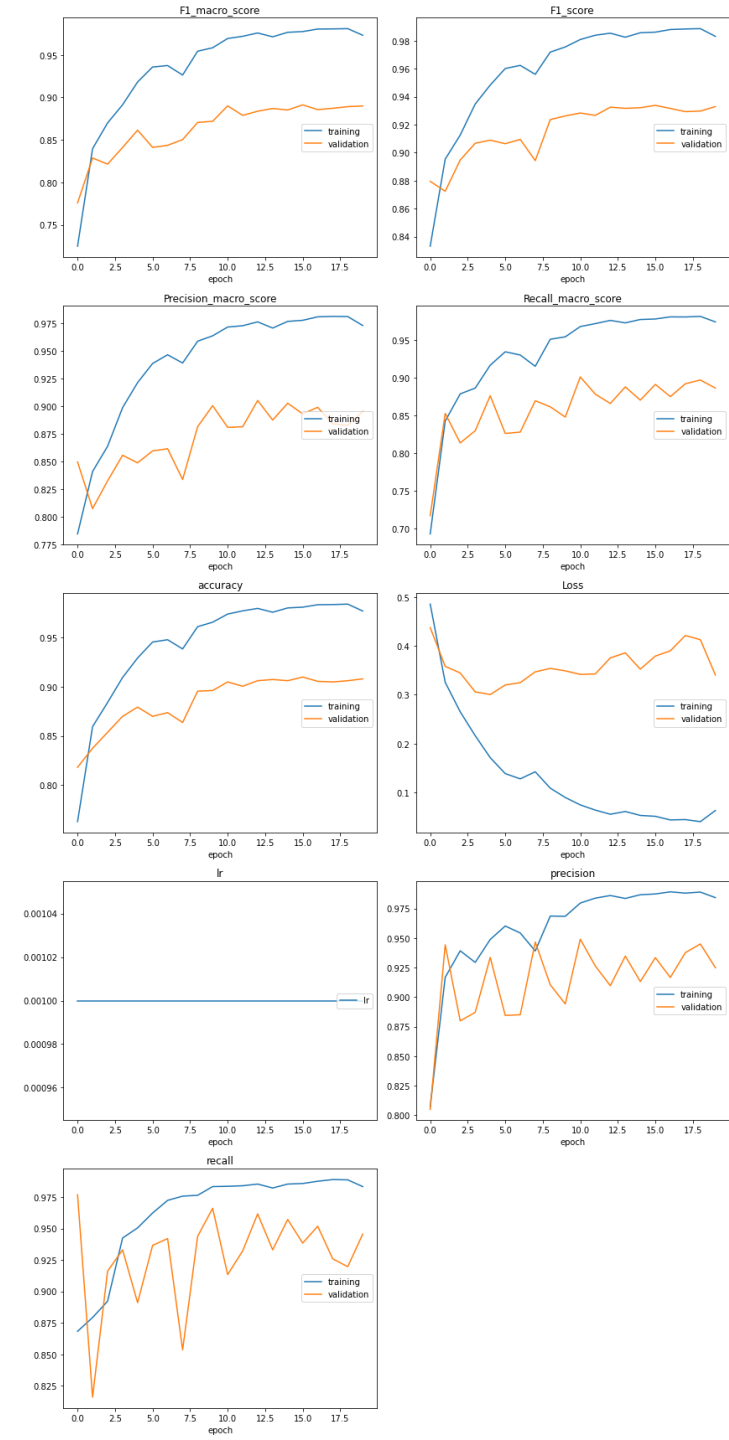
BI-LSTM-Attention

- The model will be like this like fork
 - First branch for the embedding text part
 - Second for
 - [content_name ,compagin_name , Numeric_features]
 - In the code there is an option to make it wor
 - Using the content text only
 - Using text and the numeric from dataframe
 - The upcoming result was applied on
 - content text only, so we can compare it with bert



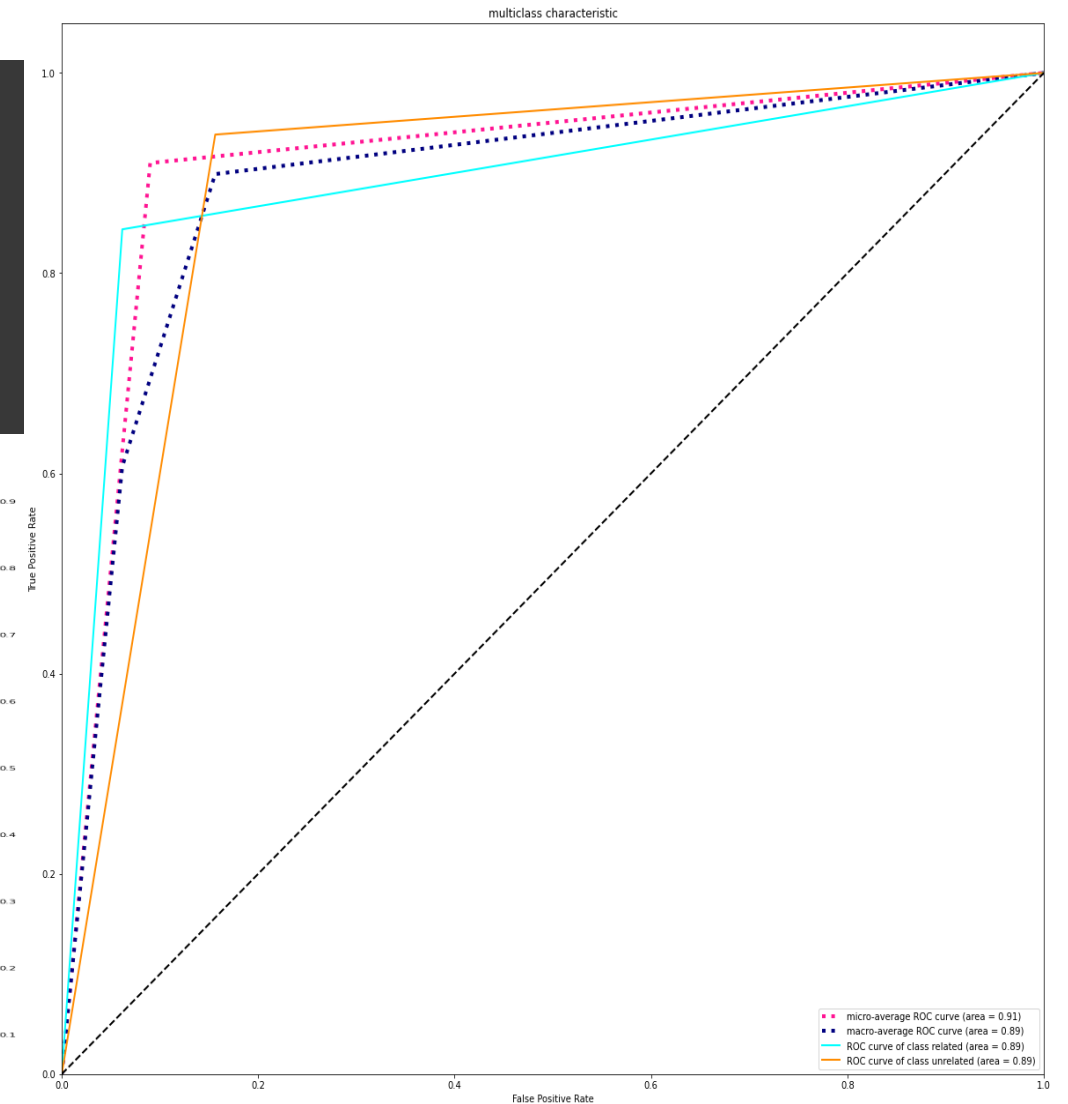
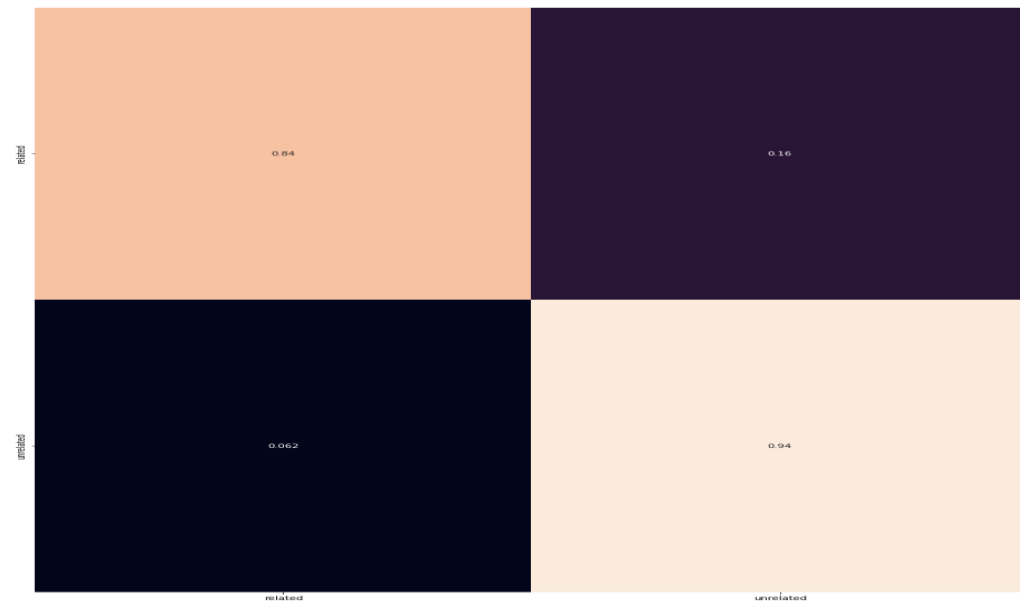
BI-LSTM-Attention Results

```
F1_macro_score
  training      (min: 0.725, max: 0.981, cur: 0.973)
  validation    (min: 0.776, max: 0.891, cur: 0.898)
F1_score
  training      (min: 0.833, max: 0.989, cur: 0.983)
  validation    (min: 0.872, max: 0.934, cur: 0.933)
Precision_macro_score
  training      (min: 0.785, max: 0.981, cur: 0.973)
  validation    (min: 0.807, max: 0.905, cur: 0.895)
Recall_macro_score
  training      (min: 0.693, max: 0.981, cur: 0.974)
  validation    (min: 0.717, max: 0.901, cur: 0.886)
accuracy
  training      (min: 0.763, max: 0.984, cur: 0.977)
  validation    (min: 0.818, max: 0.910, cur: 0.908)
Loss
  training      (min: 0.040, max: 0.485, cur: 0.063)
  validation    (min: 0.300, max: 0.437, cur: 0.340)
lr
  lr            (min: 0.001, max: 0.001, cur: 0.001)
precision
  training      (min: 0.807, max: 0.989, cur: 0.984)
  validation    (min: 0.805, max: 0.949, cur: 0.925)
recall
  training      (min: 0.868, max: 0.989, cur: 0.983)
  validation    (min: 0.816, max: 0.977, cur: 0.946)
200/200 [=====] - 212s 1s/step - loss: 0.0632 - 
Model take 4342.031024217606 S to train
```



BI-LSTM-Attention Results

Classification Report				
	precision	recall	f1-score	support
related	0.85	0.84	0.85	480
unrelated	0.93	0.94	0.94	1120
accuracy			0.91	1600
macro avg	0.89	0.89	0.89	1600
weighted avg	0.91	0.91	0.91	1600



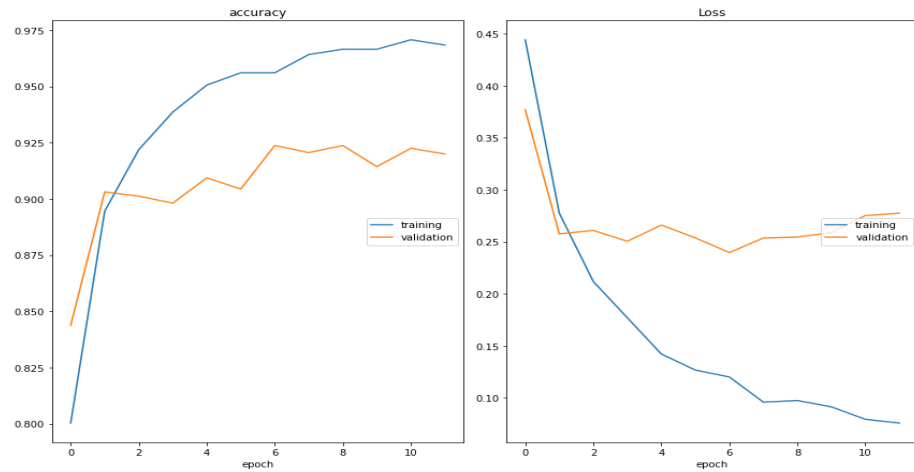
Bert

- BERT is an open source machine learning framework for natural language processing (NLP). BERT is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context.
- BERT, which stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. (In NLP, this process is called attention.)
- Using this bidirectional capability, BERT is pre-trained on two different, but related, NLP tasks: Masked Language Modeling and Next Sentence Prediction.
- The goal of any given NLP technique is to understand human language as it is spoken naturally. In BERT's case, this typically means predicting a word in a blank. To do this, models typically need to train using a large repository of specialized, labeled training data. This necessitates laborious manual data labeling by teams of linguists.

Bert

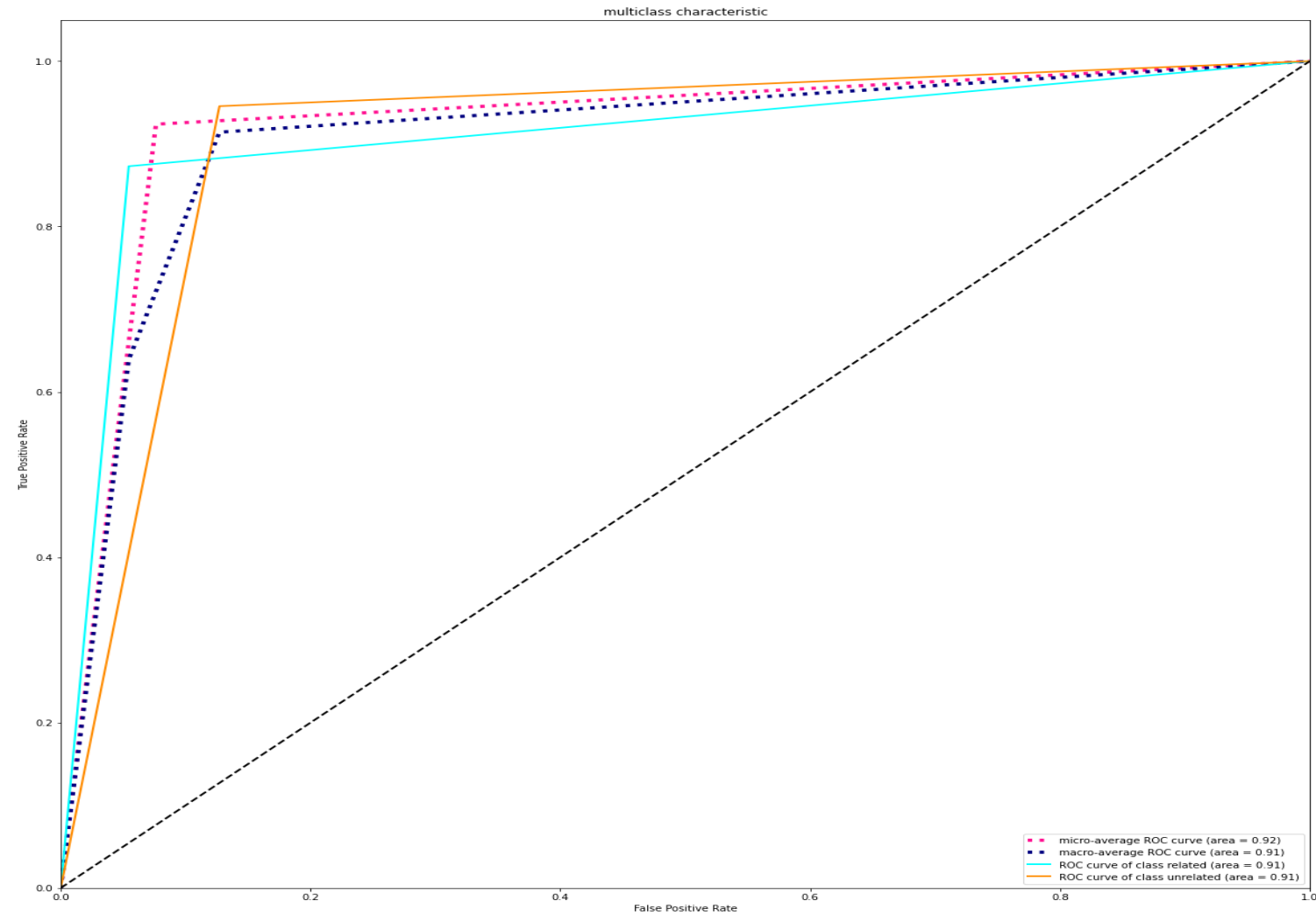
- BERT, however, was pre-trained using only an unlabeled, plain text corpus (namely the entirety of the English Wikipedia, and the Brown Corpus). It continues to learn unsupervised from the unlabeled text and improve even as its being used in practical applications (ie Google search). Its pre-training serves as a base layer of "knowledge" to build from. From there, BERT can adapt to the ever-growing body of searchable content and queries and be fine-tuned to a user's specifications. This process is known as transfer learning.
- As mentioned above, BERT is made possible by Google's research on Transformers. The transformer is the part of the model that gives BERT its increased capacity for understanding context and ambiguity in language. The transformer does this by processing any given word in relation to all other words in a sentence, rather than processing them one at a time. By looking at all surrounding words, the Transformer allows the BERT model to understand the full context of the word, and therefore better understand searcher intent.
- This is contrasted against the traditional method of language processing, known as word embedding, in which previous models like GloVe and word2vec would map every single word to a vector, which represents only one dimension, a sliver, of that word's meaning.
- These word embedding models require large datasets of labeled data. While they are adept at many general NLP tasks, they fail at the context-heavy, predictive nature of question answering, because all words are in some sense fixed to a vector or meaning. BERT uses a method of masked language modeling to keep the word in focus from "seeing itself" -- that is, having a fixed meaning independent of its context. BERT is then forced to identify the masked word based on context alone. In BERT words are defined by their surroundings, not by a pre-fixed identity. In the words of English linguist John Rupert Firth, "You shall know a word by the company it keeps."

BERT Results



```
accuracy
  training      (min:  0.800, max:  0.971, cur:  0.968)
  validation    (min:  0.844, max:  0.924, cur:  0.920)
Loss
  training      (min:  0.076, max:  0.444, cur:  0.076)
  validation    (min:  0.240, max:  0.377, cur:  0.278)
200/200 [=====] - 102s 510ms/step - loss: 0.07
Epoch 12: early stopping
Model take 1220.5935561656952 S to train
```

Classification Report				
	precision	recall	f1-score	support
related	0.87	0.87	0.87	480
unrelated	0.95	0.95	0.95	1120
accuracy			0.92	1600
macro avg	0.91	0.91	0.91	1600
weighted avg	0.92	0.92	0.92	1600



Run Code

- There are 3 main:-
 - Main:- for run app flask load all model and pre processing.pkl to run the flask app
 - Main1:- to request and visualize dataset
 - Preprocessing:-
 - Label encoder
 - CountVectorizer & TfidfVectorizer
 - Build and Train SGD, SVM Classifier
 - Test SGD, SVM Classifier & Evaluate model
 - Main2:-
 - Preprocessing:-
 - Word tokenizer
 - Padding sequences
 - Build Bi-LSTM-Attention
 - Train Bi-LSTM-Attention
 - Test Bi-LSTM-Attention Model & Evaluate

Run Code

- Most Important part is to Generate the pkl files and model weights from notebook the 3 ones
- The Hybrid Mode depend on 2 flags
 - `using_cat_feats = True`
 - `using_numeric_feats = True`
- Making them true or one of them will active the Hybrid Model training
- And make sure you active them too if you are using the FLASK Webpage for inference

Flask App












- The flask web page is simple just enter the text, select the model and press classify
- Each model from previous slides has it's own pipeline
 - SGD classifier:-
 - Preprocsssing, Countvectorizer, TDF vectorizer
 - SVM classifier:-
 - Preprocsssing, Countvectorizer, TDF vectorizer
 - XGBoost classifier:-
 - Preprocsssing, Countvectorizer, TDF vectorizer
 - Bi-LSTM- attention classifier:-
 - Preprocsssing, Tokenizer, padding sequences
 - Bert classifier:-
 - Preprocsssing, bert tokenizer

Notes

- SGD model.pkl (24 MB).
- SVM model.pkl (30 MB).
- Preprocessing Pipeline.pkl (20 MB).
- Bi-Lstm-attention model weights.h5 (2000 MB) Was removed to reduce the size for uploading.
- Bert model weights.h5 (428 MB) Was removed to reduce the size for uploading.
- All you need to run each notebook for each class to generate weights

Notes

- Weights Result should be like this

 FeaturesEncoder	4/30/2022 3:50 AM	File folder	
 bert_model.h5	4/28/2022 11:50 PM	H5 File	427,953 KB
 label_encoder.pkl	4/30/2022 3:49 AM	PKL File	1 KB
 lstm_attention_model_weights.h5	4/28/2022 10:05 PM	H5 File	277,984 KB
 lstm_clf_params.json	4/28/2022 10:01 PM	JSON File	1 KB
 minmax_scaler.pkl	4/30/2022 3:49 AM	PKL File	1 KB
 preprocessing_pipeline.pkl	4/30/2022 3:50 AM	PKL File	18,534 KB
 SGD_clf.pkl	4/30/2022 3:52 AM	PKL File	4,093 KB
 SVM_clf.pkl	4/29/2022 4:37 AM	PKL File	23,879 KB
 tokenizer.pkl	4/28/2022 10:00 PM	PKL File	29,762 KB
 XGBoost_clf.pkl	4/29/2022 5:03 AM	PKL File	157 KB

Notes

Some Notes Book are trained on Kaggle for net speed & colab For Gpu

Most Important

The Dataset I USED FROM YOU WAS
REMOVED AFTER SENDING THE TASK

And may I ask for feedback that will help me to advance
my skills

Thanks In Advance