Milestone II Project Report on Text Difficulty Prediction
by: Prashant Sanghal

Project Description

Apply supervised and unsupervised learning techniques on Wikipedia text to predict sentences which will need to be simplified for readers to make it easier to understand. Readers may include students, children, adults with learning/reading disability, and non-native English speakers.

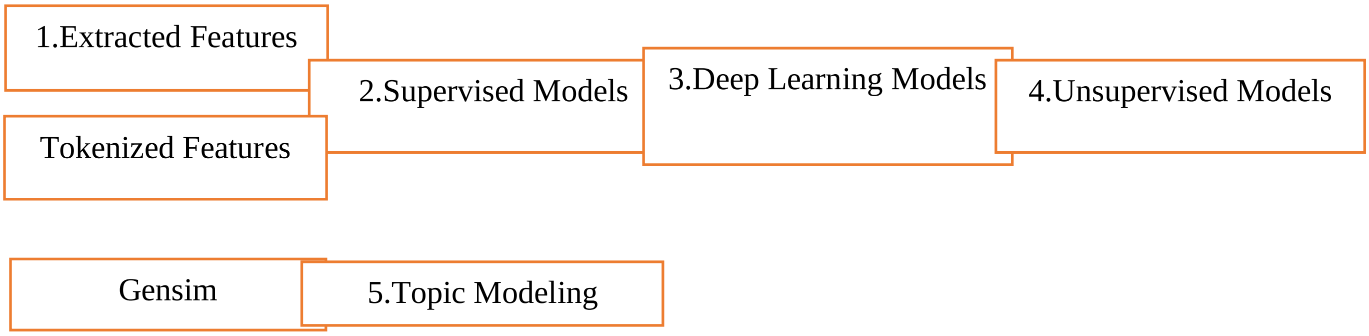
Project Workflow:

This project contains 5 jupyter notebooks. It begins with extracting features from the original text and then goes on to implementing supervised and unsupervised learning models using extracted features and text tokenizers such as TFIDF, Sentence Piece, and Keras Tokenizer. The goal of doing this was to assess the effectiveness of feature representation in classifying text difficulty as well understand which steps in manual feature extraction worked well Vs could be improved in future.

Please refer to following jupyter notebooks for code implementation.

1. Text Difficulty-Feature Extraction-Final
2. Text Difficulty-Supervised Models-Final
3. Text Difficulty- Deep Learning-Final
4. Text Difficulty-Unsupervised Models-Final
5. Text Difficulty-Topic Modelling-Final

Features extracted from the first notebook “Text Difficulty-Feature Extraction-Final” has been used extensively in all notebooks to save computational run time. Here is workflow diagram.



1.Extracted Features

2.Supervised Models

3.Deep Learning Models

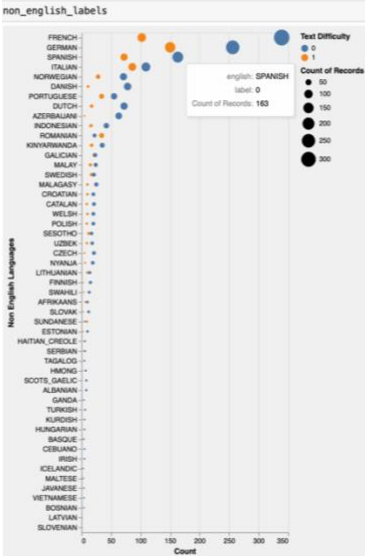
4.Unsupervised Models

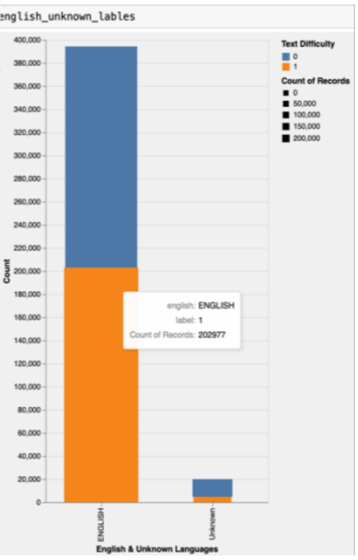
Tokenized Features

Gensim

5.Topic Modeling

1. **Feature Extraction:** Pre-processing text such as removing non-alpha numeric characters, digits, stopwords (not ideal for our use case) was a critical step. Then, vectorizing a normalized text in to numbers was another. But in all of this, another important consideration was to preserve the context of the text while also avoiding the curse of dimensionality. Hence, applied PCA on normalized-vectorized text to see what impact it would have on the model performance. Interestingly, PCA features showed up in the top 10 features which contributed in



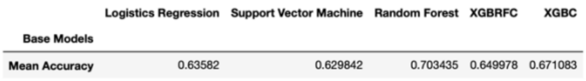


predicting text difficulty. In addition, also utilized additional features such as dale chall, concreteness ratings in identifying statistical measures of the normalized text such as average length of a sentence/word, number of unknown words, count of syllables, POS tags etc based on lemmas in building above feature set. It was interesting to note that not all words in the normalized text were English words. We had unknown words as well as foreign words which would make text difficult for many native speakers. It was hoped that keeping stopwords in the text as well maintaining a count of Nouns, Adjectives etc would help us classify text better. Please refer to “extracted_train_features_with_stopwords_pca.csv” file to see the complete range of features.

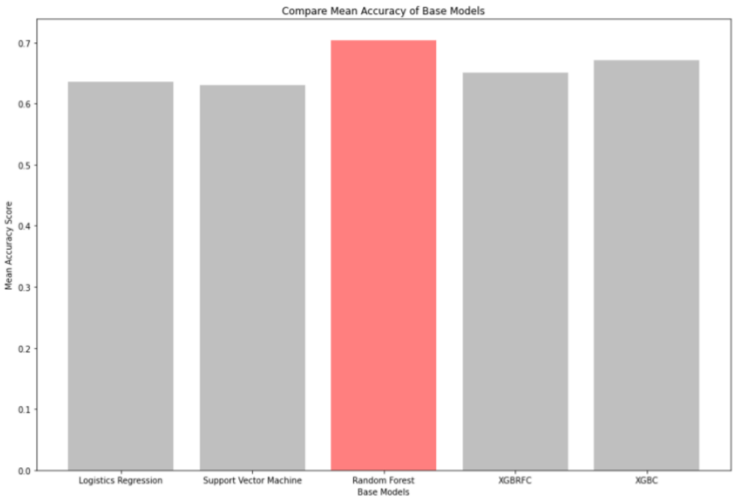
Now, let’s see how these features performed on supervised models.

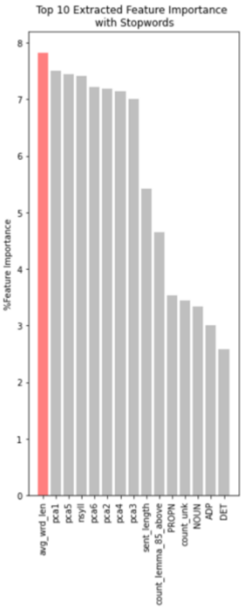
2. Supervised Models:

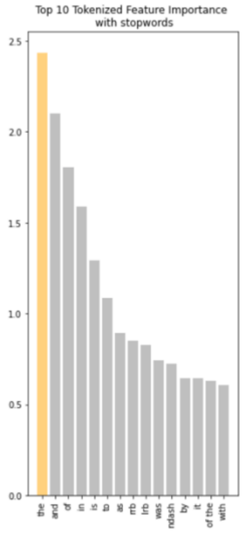
Knowing which features responded well in predicting text difficulty started off with building a threshold model first. Example: a sentence length of 15 had a precision score of 72% which was good to know because then we can compare it with feature importance rated by other models. Besides above, other base models to try out were, Logistic regression, Support Vector Machine, Random Forest and XG Boost. The goal was to see which model without any hyperparameter tuning would result in highest model accuracy just based on feature sets presented to it. Example: Is it going to be Logistic regression which uses ‘sigmoid function’ as output layer in neural net to classify binary labels or perhaps random forest which builds multiple decision trees at no cost of normalizing features and has multiple tuning parameters to give stable predictions.

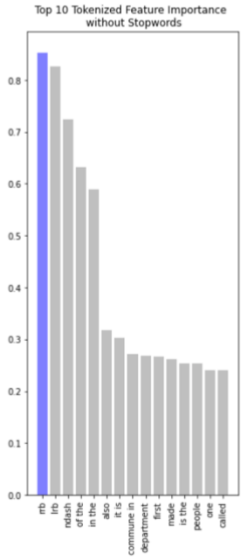


	Logistics Regression	Support Vector Machine	Random Forest	XGBRFC	XGBC
Base Models					
Mean Accuracy	0.63582	0.629842	0.703435	0.649978	0.671083









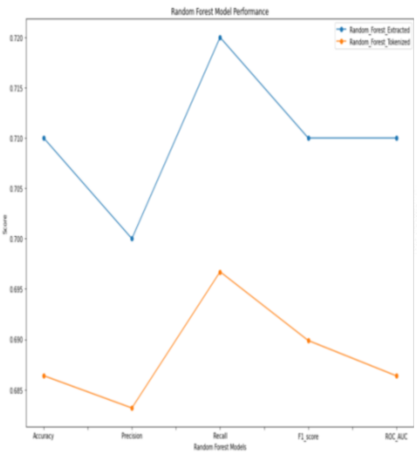
i. *Base model Selection*:
It took no more than 16 minutes to figure out that Random Forest was the base model to apply GridSearchCV on.

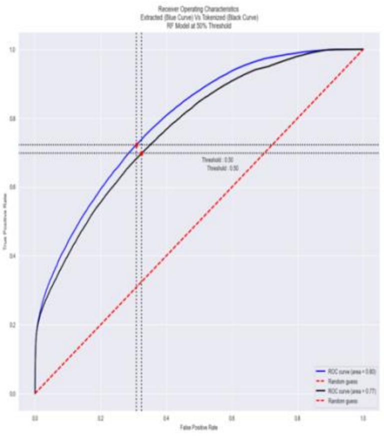
ii. *Hyperparameter Tuning:*
Important features were n_estimators, and max_depth of the tree. Interestingly, hyperparameter tuning on max_depth didn’t result in best model accuracy. It was best for the model to decide max_depth on its own.

iii. *Comparing Extracted and Tokenized Random Forest Models:*

Using extracted features gave us the best model performance on Random Forest. But why so? can probably be better explained by doing a side-by-side comparison of important features . Example: Extracted model used average word length/PCA and other statistical measures to classify text while tokenized model used TFIDF feature representation to classify text "the", 'lrb', 'rrb' etc. Moreover, selection of parameters such as max_features, max_df, min_df and stopwords can impact feature importance. We will see a better example of tokenized model performance in deep learning (RNN) because unlike random forest, it can capture long-short-term dependencies.

iv. *Random Forest Model Evaluation:*





Extracted model recorded >70% model accuracy, ROC curve area 80%. It was able to predict positive predictions more accurately than the tokenized model. Possibly, removing corrupted data such as unknown words & other foreign words might result in better model performance. Besides above, some of the extracted feature set contributed >7% in text difficulty when compared to tokenized model which was under 2.5%. Since, extracted and tokenized model both consists of stopwords, we can see how stopwords can make sentence longer and add more words for the learners. Tokenized model without stopwords was able to identify 'rrb' and 'lrb' as difficult text, which was an interesting discovery.

3. **Deep Learning Models:** Again**,** a part of supervised learning family it was interesting to build a sequential neural network with extracted features and try identify best epoch and batch_size on a 10% training sample. An epochs= 8 and batch_size = 1000 have seemed to worked well throughout this implementation when used with both cross-validation and train-test-split methods. However, building a tokenized deep learning model was not the same as building a model with tabular data.

i. Keras Tokenizer:

Thankfully, keras has provided good documentation on keras tokenizer which does a lot of text processing under the hood. Example, it can convert text in to sequences as well as add padding to return same length sequences. Then, in the embedding layer, we can restrict vocabulary_size, set input and output dimensions and stack other layers sequentially such as RNN, CNN with different number of neurons, and activation functions of our choice to help make model learn better. I chose single dimension convolutional layer with maxpooling to improve local feature learning and bidirectional GRU with attention layer to enhance global feature learning. I also applied recurring dropout of 0.2 and activation function "relu" to avoid vanishing gradient problem in CNN's, which usually not a problem in RNN's (LSTM/GRU). Finally, while compiling the model, used 'Adam" as the optimizer to help model converge faster towards the local minima and binary-cross entropy as a loss-function to optimize classification task. Furthermore, to save computational resources, applied EarlyStopping call back which was super helpful. However, one of the challenges is that this callback could not be applied while using GridSerachCV or evaluating cross_val_score. For that, it had to go through train_test_split method, which was an extra step and not fun!

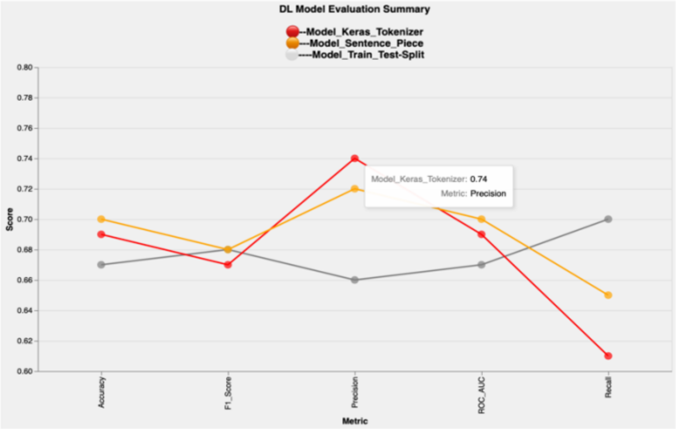
ii. *Sentence Piece Tokenizer:*

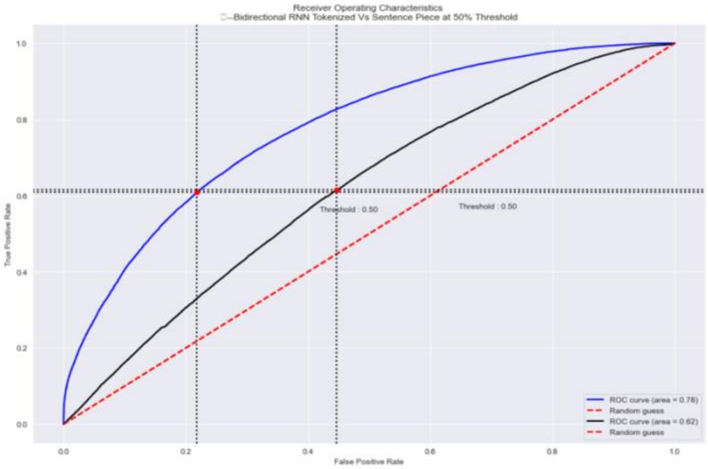
Sentence Piece is a sub-word tokenization technique (a.k.a Byte pair encoding) which can deal with unknown words more effectively. So, keeping the same specs as above, used a different tokenizer to compare the difference. Let's see how the three deep learning models compared to each other.

iii. *Deep learning model evaluation:*

CNN-BiGRU-Attention with both tokenizers offered comparable model accuracy and precision scores on the validation dataset. However, model performance was extremely sensitive to hyperparameters such number of neurons, bidirectional layer and batch_size. Our simple NN architecture with GridsearchCV was indeed very helpful in picking out the best epochs and batch_size for faster run time. One thing which caught me by a surprise was a significant difference in ROC curve area between the two tokenizers which may require further analysis.

Just for reference, the idea of building a CNN-BiGRU has been discussed extensively in the paper enclosed below.





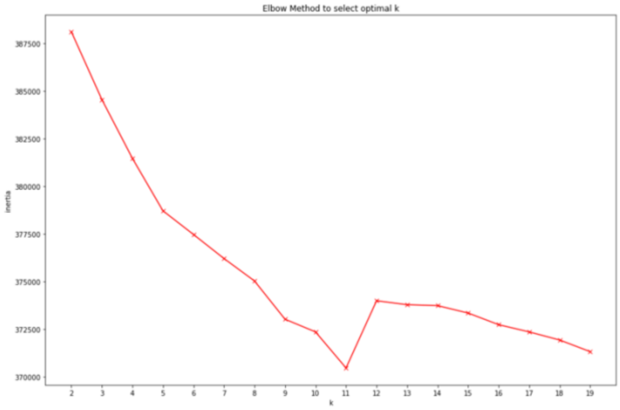
https://www.researchgate.net/publication/329189864_A_Bi-Directional_LSTM-CNN_Model_with_Attention_for_Aspect-Level_Text_Classification

4. Unsupervised Learning:

So far, we had known which text was difficult and which text wasn't. Now, in this part of the learning, we don't have target labels. So, the first place to begin for us was to use the normalized text from the extracted feature set, and convert it in to a TFIDF vectorizer (this is same as CounterVectorizer + TFIDFTransformer) and then apply K-means clustering to see if we can find some patterns in the dataset. But, just finding clusters will not be sufficient in classifying text difficulty. So, decided to use k-means for intuition building and then applying topic modeling (LDA) to this unfold deeper. Let's focus on K-Means first:

i. K-Means Clustering:

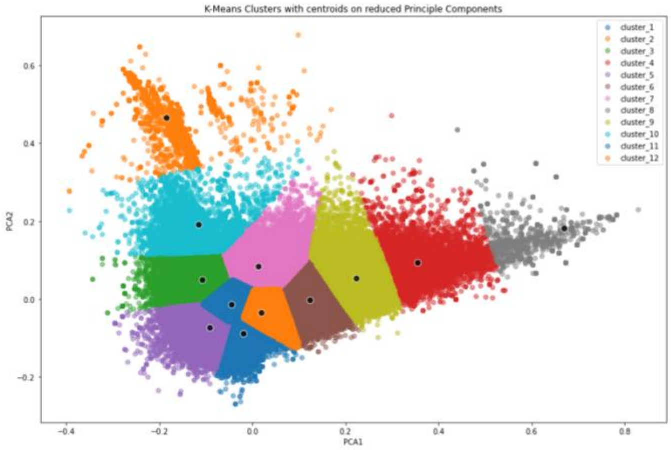
It starts with specifying a range of clusters which we think the dataset might contain. We can use sqrt of total features as a range. But it would have been computationally very expensive given our large feature set. So, chose between (2, 20) as a K range to start clustering.

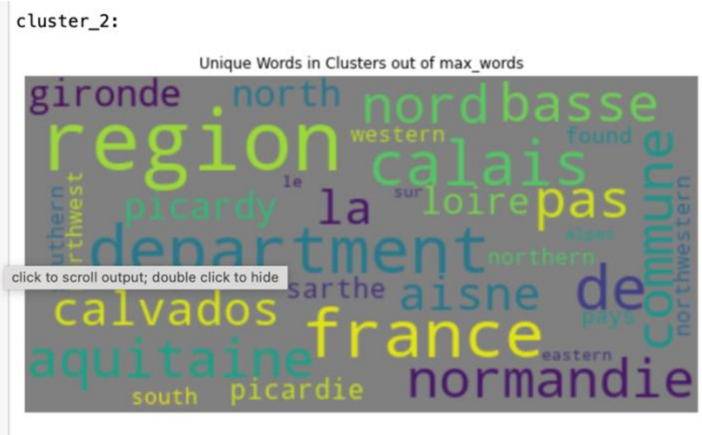


Elbow method, which helps calculate the change in inertia as we increase the clusters. At K = 12 onwards it showed marginal decrease in inertia, showed lower Davies_score of 5.80 and higher calinski_score of 1801 which seemed optimal value to fit k-mean clusters on the trained TFIDF array (X). The central words derived from k-means clustering with top 100 words and predicted labels showed some patterns to evolve in the clusters, example cluster_4 “championship, award, best, team” was a good start to go a step further and tryout topic modeling.

ii. *Model Evaluation:*

But before that, it was also important to view these clusters as a scatter plot on a 2-dimensional principle components, which was interesting to see. The 12 clusters were dense and seemed separated from other clusters. But this dense structure could also be due to high frequency of stopwords present in our normalized text. So, let’s remove these stopwords and basic English words we pre-processed earlier and plot them on a word cloud. Example: cluster_2, seems to be a good candidate for making text difficulty high for many learners including me.




cluster_2:

Another important observation was that some of the unique words contained in the word cloud after filtering out stopwords were either a bi-gram of remaining stopwords or stopwords combined with other words (both easy and tough). How to account for this representation, was also challenging. Example “the th” “lrb_born” etc. But, K-means was indeed helpful in exploring pattern in the text but lacks the ability to capture context behind the text for which we would need to apply topic modeling, using LDA.

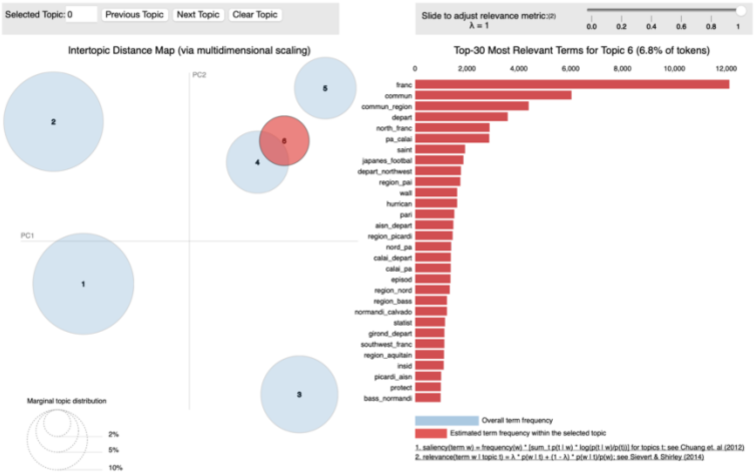
5. Topic Modeling using Gensim

i. *Preparing data for LDA*

This step started with getting normalized text from the feature set as usual. Then, building a bi-gram (minimum count 20) corpus using gensim library from which we then extracted lemmas using spacy. The output of this pre-processing step was a dictionary of lemmatized words (no_below 10 and no_above 60%) and a corpus of bag of words which was used as an input to the LDA model. The number of topics were set to 6 (1/2 of the optimal k-clusters found from unsupervised learning model) and per_word_topics was set to True so that we can see the words per each topic.

ii. *Model Evaluation:*

We obtained a lower topic model perplexity score of -ve 8.81 and coherence score of 0.33 after preprocessing and lemmatization. This, to some extent preserved topic context and showed us relevant words contained under each topic. Example: topic 1 which represents 28.2% tokens appears to relate to 'air travel'. Moreover, in the word cloud we can also see some longer words which may be contributing to text difficulty as observed from supervised learning model where sentence length and average word length accounted for >7% in predicting text difficulty.



1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Furthermore, by reducing number of topics from 12 to 6, average coherence score reduced slightly while also reducing overlapping topics, which was an interesting observation

iii. *LDA Problem:*

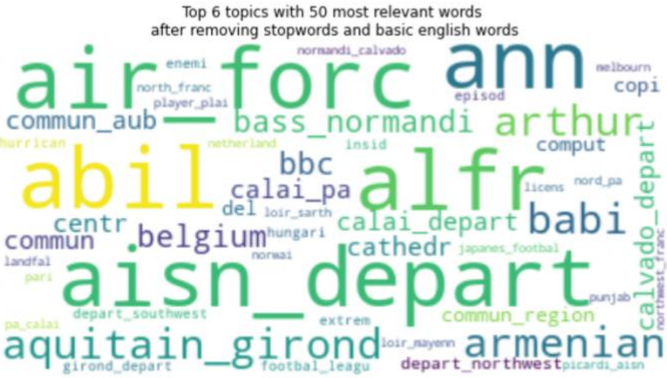
From pyLDAvis visualization above, Topics 4 & 6 has a bit of an overlap while topic 1,2 and 5 are well separated as shown in the plot above.

Let’s see if we see any words in each topic that seems difficult to comprehend using word cloud. In future, we intend to build a vocabulary out of word cloud text to further evaluate text difficulty by above topics. It's possible that some topics contain more difficult text than others.

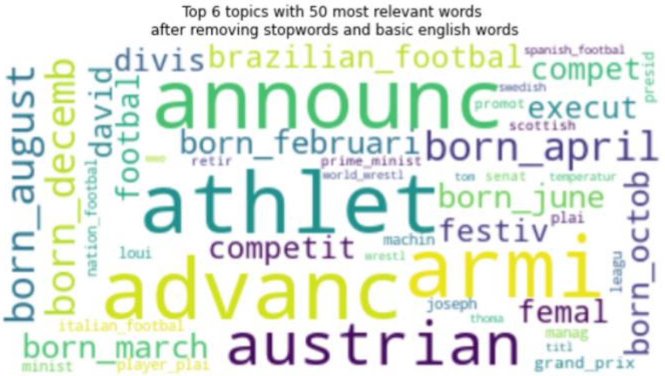
Word Cloud of six topics found in Wikipedia text by importance.

Some of these texts could be long words, foreign words, names which would be unfamiliar to many readers, hence could be contributing to text difficulty.

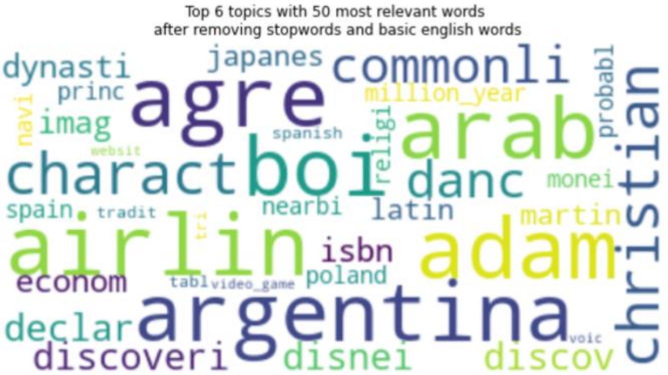
topic_1:



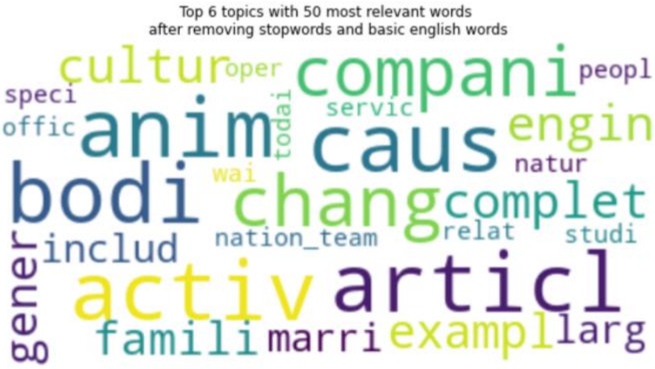
topic_2:



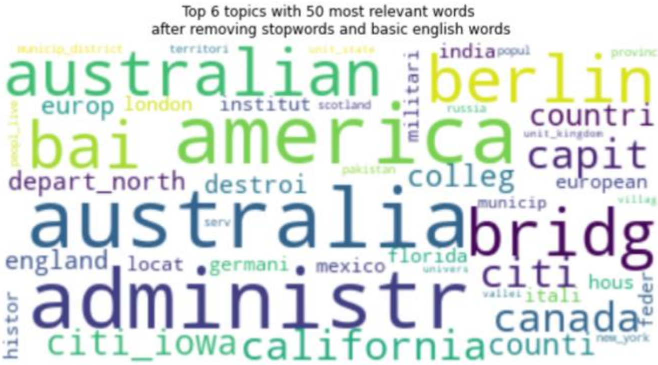
topic_3:



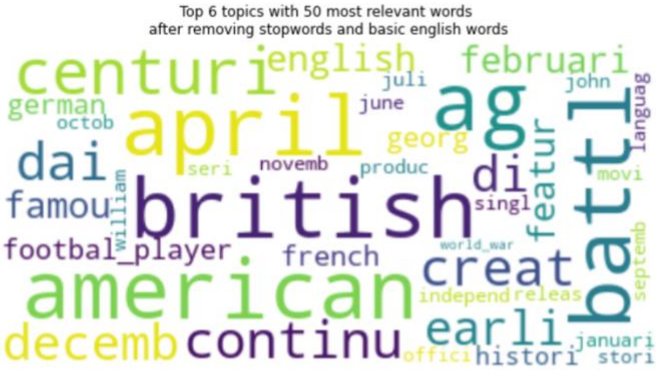
topic_4:



topic_5:



topic_6:


Top 6 topics with 50 most relevant words
after removing stopwords and basic english words
centuri
english
februari
juli
john
german
june
ag
languag
octob
april
georg
dai
seri
novemb
produc
di
movi
famou
william
british
singl
featur
battl
footbal_player
french
world_war
septemb
american
creat
independ
releas
decemb
continu
earli
offici
histori
januari
stori