Data Analytics Project

Professor: D.Gounopoulos

- **Panagiotopoulos Georgios :** CS2190012
- **Stefou Theodoros :** CS2190002

Fall 2020

The code for this project can be found in this github repository:
https://github.com/giorgospan/Data-Analytics

We ran the code on a machine with the following specifications*:

- OS: Ubuntu 16.04 LTS
- CPU: AMD Ryzen 5 2600 3.4GHz
- RAM: 16GB 3000MHz
- Python: 3.5.2

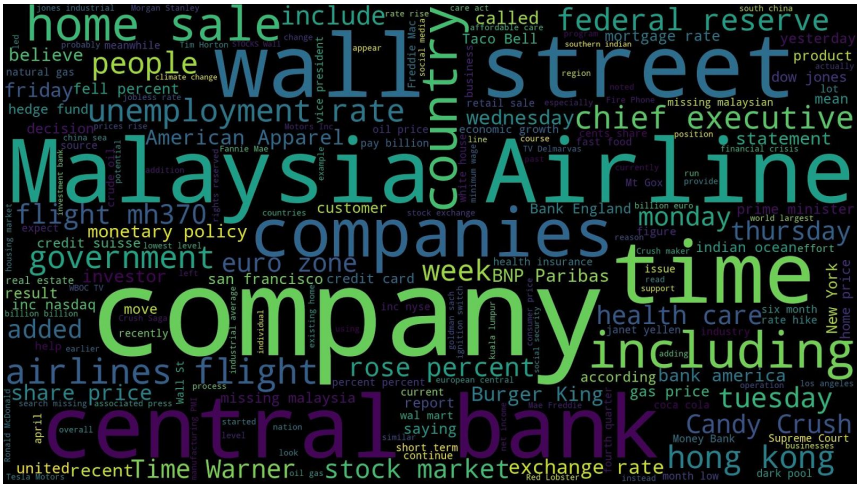
*For the Neural Network implementation of the last requirement, we ran our experiments on Google Colab.

Requirement 1: Text classification

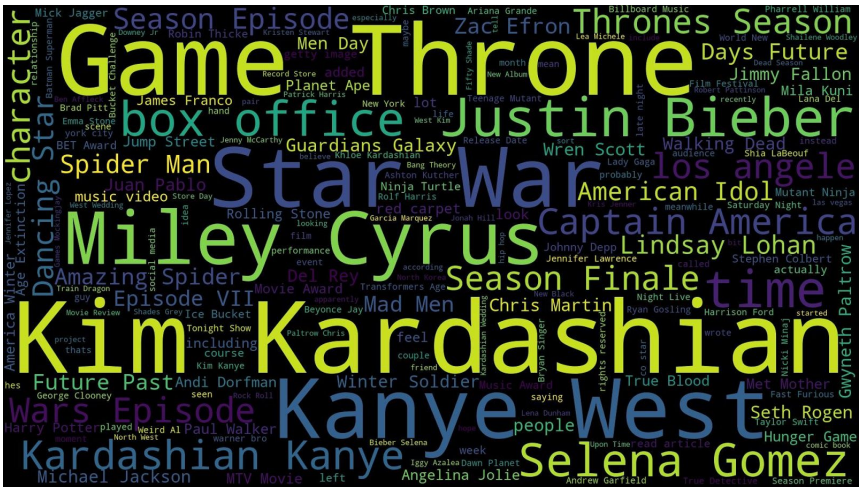
Question 1a: Get to know the Data: WordCloud

We removed any punctuation and stop words in the Title and Content columns (in that order). For stop words, we used the ones provided in nltk.corpus and added a few of our own (can be found in the file stopwords.txt). For each article, we fed 20*Title + Content to the wordcloud API. The following pictures are the results we produced.

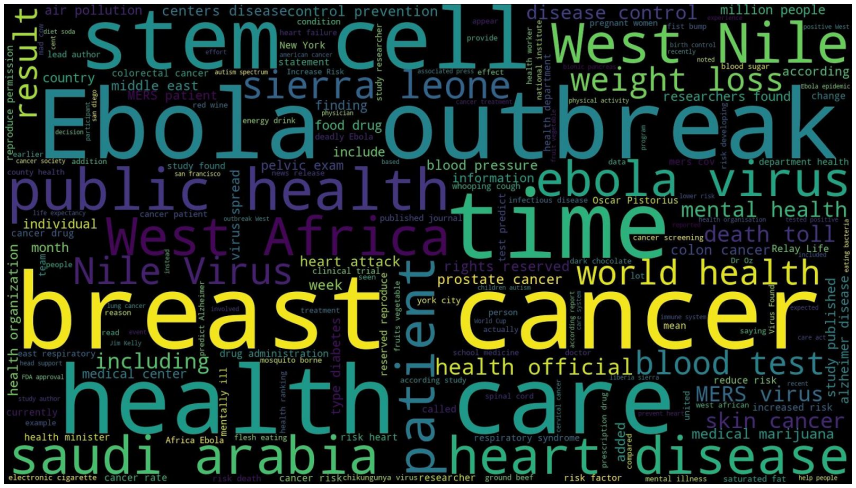
Business



Entertainment



Health



Technology

Question 1b: Classification Task

For this question, we used the same stop words as in the previous question. Our heuristic for taking the title into consideration is to prepend it 3 times to the content. We also tried prepeding the title more times (up to 20) and it did not make a difference.

To vectorize the sentences we chose sklearn's TfidfVectorizer over CountVectorizer. The main difference is that the former normalizes the word values by taking into account the frequency of the word in the corpus.

Beat the Benchmark

In order to beat the benchmark, we used KNN with the bag of words technique.
Since Linear SVM worked very well on our data, it seems that this problem is well suited to be modelled spatially. Therefore, KNN seems ideal for this kind of task. Unfortunately it didn't surpass our previous results (only for 0.003) and also took a lot of time to complete (11+ minutes). This came as no surprise because KNN does not generalize over data in advance (lazy learner) and we are also dealing with a lot of data.

Evaluation Results

In the following table you can see the metrics we achieved for the various algorithms we tested using 5-Fold.

Statistic Measure	SVM (BOW)	Random Forest (BOW)	SVM (SVD)	Random Forest (SVD)	KNN (BOW)
Accuracy	0.9759	0.7504	0.8886	0.9390	0.9722
Precision	0.9744	0.8643	0.8818	0.9355	0.9701
Recall	0.9730	0.6655	0.8713	0.9276	0.9688
F-Measure	0.9737	0.7134	0.8761	0.9314	0.9694

Model Parameters:

- SVM: We decided to set the value of tolerance to1e-5, as it produced slightly better results than the default value (1e-4)

- Random Forest: After experimenting with various parameter values, we found that n_estimators=100 and max_depth=18 led to the best accuracy-performance ratio for our model

- KNN: After various experiments, we settled for n_neighbors=10.

Output File

For this task we used SVM with BOW representation and produced the testSet_categories.csv file.

Our team's submission on Kaggle can be found under the username "*Thodoris Stefou - George Panagiotopoulos*".

Requirement 2: Nearest Neighbor Search and Duplicate Detection

Question 2a: De-Duplication with Locality Sensitive Hashing

For this question we decided to not use stop words. The reason for this is that for the next question, we got worse results when using stop words and since these two are related, we decided to go without them.

The implementations we used are the following:

- Exact-Cosine: sklearn.metrics.pairwise.cosine_similarity
- Exact-Jaccard: Python sets and 10 processes as workers
- LSH-Cosine: This github repository
- LSH-Jaccard: datasketch

Evaluation Results

Type	BuildTime	QueryTime	TotalTime	#Duplicates	Parameters
Exact-Cosine	0	42.78	42.78	1373	-
Exact-Jaccard*	0	26m40s	26m40s (~260-300m)	313	-
LSH-Cosine	21m55s	11m15s	33m5s	1243	K=1
LSH-Cosine	11m42s	6m22s	18m4s	1114	K=2
LSH-Cosine	8m5s	4m1s	12m6s	971	K=3
LSH-Cosine	4m6s	1m52s	5m58s	833	K=4
LSH-Cosine	4m25s	50s	5m15s	762	K=5
LSH-Cosine	3m49s	32s	4m21s	710	K=6
LSH-Cosine	3m52s	22s	4m16s	587	K=7
LSH-Cosine	3m40s	15s	3m55s	540	K=8
LSH-Cosine	3m39s	12s	3m51s	482	K=9
LSH-Cosine	3m40s	11s	3m51s	449	K=10
LSH-Jaccard	6m8s	4s	6m12s	618	P=16
LSH-Jaccard	9m2s	5s	9m7s	499	P=32
LSH-Jaccard	14m3s	8s	14m11s	538	P=64

- **Exact-Cosine:** Batches consisting of 1000 test vectors
  We used the cosine_similarity function found in the sklearn.metrics.pairwise module.

- **Exact-Jaccard:** We utilized python's multiprocessing module and equally distributed the workload among 10 worker processes.
  We wrote our own implementation which uses python sets and divides the length of the intersection over the length of the union.

  The similarity threshold for all combinations of methods and parameters is 0.8 .

Question 2b: Same Question Detection

For this question we used the Linear SVM implementation of sklearn. For the training, we tried several heuristics between the pairs of vectors. The following table shows the results we got for each heuristic using 5-fold validation, X1 and X2 being the vectors of the first and second question respectively. (For all rows)

Evaluation Results

Heuristic*	Accuracy	Precision	Recall	F-Measure
$X1 * X2$	0.7702	0.7619	0.7324	0.7411
$X1 + X2$	0.7523	0.7355	0.7249	0.7290
$X1 - X2$	0.5736	0.5017	0.5013	0.4873
$\|X1 + X2\|$	0.7523	0.7355	0.7249	0.7290
$\|X1 - X2\|$	0.7796	0.7644	0.7714	0.7673
$\|X1 - X2\|^{0.7}$	0.7808	0.7659	0.7740	0.7690

*Element by element operations

Output File

Our team's submissions on Kaggle can be found under the username "*Thodoris Stefou - George Panagiotopoulos*".

Requirement 3: Sentiment Analysis

We noticed that the files contained the HTML tag
. That is why we removed it from the test and training set.

1. For the classic Machine Learning technique we used LinearSVC and the TfidfVectorizer.

2. For the deep learning technique we used Keras library in order to build a CNN network.

NN Fine-Tuning:

- We opted for pre-trained word embeddings (GloVe) instead of initializing our embedding layer from scratch and learning its weights during training. GloVe vectors lead to slightly higher accuracy.

- We opted for only 2 epochs, as the size of the dataset is relatively small (25.000 imdb reviews)

- We used an 1-D convolutional layer with max pooling with dropout rate of 25%.

- We decided to use *sigmoid* as an activation function and Binary Cross-Entropy for calculating the loss

The table below presents the metrics we extracted from the two algorithms using 5-fold cross validation.

Statistic Measure	SVM	Neural Network
Accuracy	0.8942	0.8775
Precision	0.8943	0.8780
Recall	0.8943	0.8773
F-Measure	0.8942	0.8774

Output File

Our team's submissions on Kaggle can be found under the username "*Thodoris Stefou - George Panagiotopoulos*".