Big Data Mining Techniques - Report

Authors

Marios Papamichalopoulos	**CS3190006**
Christos Charalampos Papadopoulos	**CS3190009**

Github Link

Data Mining

Libraries Used

- sklearn
- PIL
- pandas
- numpy
- wordcloud
- datasketch
- fuzzywuzzy

Requirement 1 - Text Classification

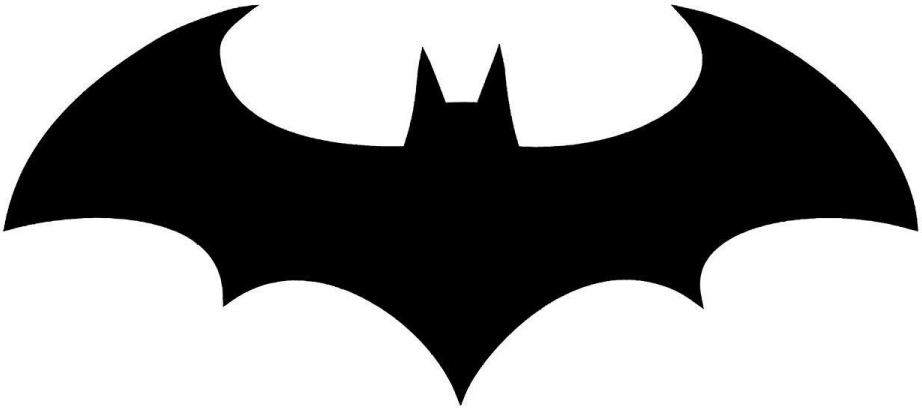
Question 1a: Wordcloud

Due to the fact that a title of a news article has more impact than the content, we gave a weight factor of 5 to the title, by adding five times the headlines when generating the wordclouds.

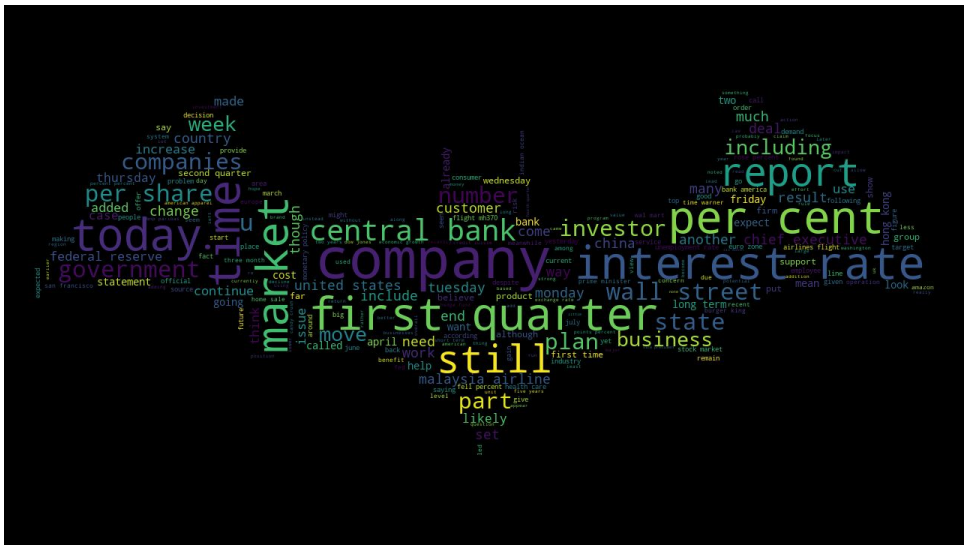
We have also used stopwords as well as added some extra of our own to the default dictionary, so as to eliminate those that were not taken into consideration and we felt were irrelevant to the category of the article. Such words appeared in great frequency and ruined some of the wordclouds.

Following we present the mask we used for the wordclouds and the wordclouds themselves for each category required by the assignment.

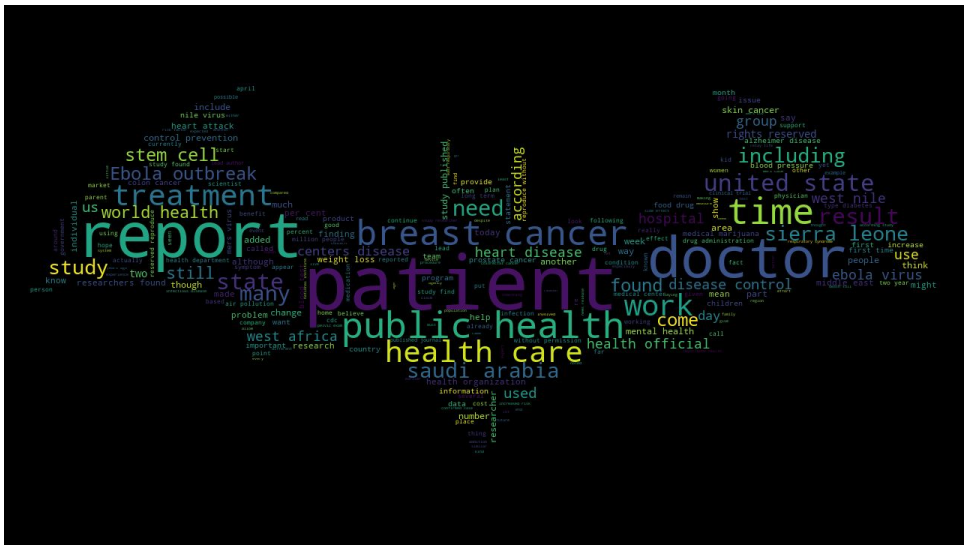
Mask used



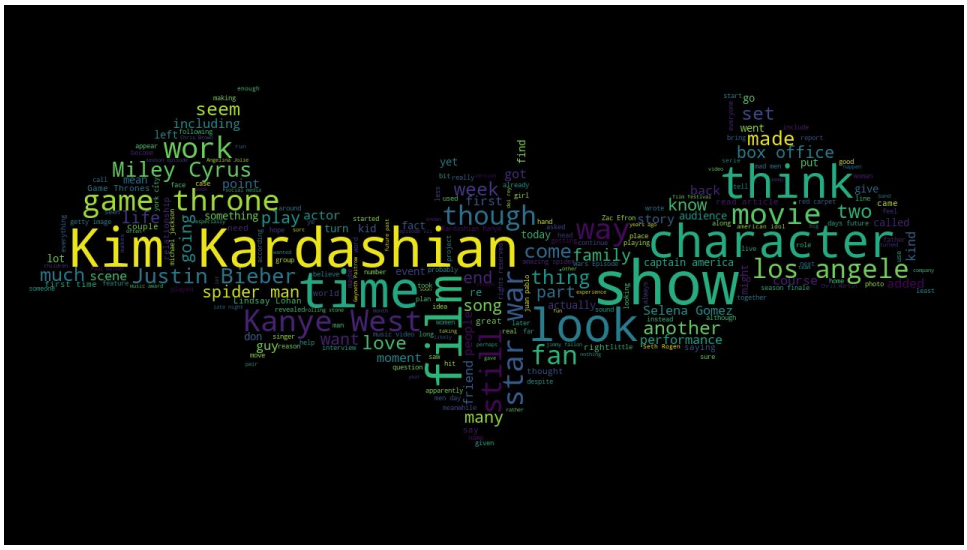
Business



Health



Entertainment



Question 1b: Classification Task

For this task we used the methods the assignment required. For the Bag of Words preprocessing we used the **CountVectorizer** of sklearn. For SVD, **TruncatedSVD**.

For the SVM classifier we used sklearn's ***LinearSVC***. We chose ***LinearSVC*** over ***svm.SVC*** because of the fact that the first one implements a **one vs many** approach, compared to the

one vs one approach of the later one thus being a lot faster since it has to construct one classifier for each label.

For the Random Forest classifier we used sklearn's ***RandomForestClassifier.*** Random Forest in general was a lot slower than SVM and has worse metrics. We tweaked a bit with its *depth* and *components* parameters, but the metrics would not surpass SVM.

From the results we can see that using *Singular Value Decomposition* makes the model have worse metrics. This is as we expected, since SVD reduces the dimensions of the features for better performance and it is useful when we have a lot more data than the ones we had in the assignment. This can be seen practically when running Random Forest, where without SVD the 5-fold cross validation lasts much more compared toS Random Forest using SVD.

Beat the benchmark

The method we chose to beat the models required by the assignment was SVM, but with some minor preprocessing data tweaks to achieve higher metrics:

- We applied weight to the title of the train_set of a factor 10 and a factor of 2 for the content. The way we applied the weight was by including 10 times the title in the train set and 2 times the content and not by multiplying the vector by the weight factor. Increasing the weight did not give any better results, plus the method we used gave more weight to the words that were included in the title. So if a word was both in the title and in the content section it would have more impact.
- We used a different vectorization technique, and specifically term frequency – inverse term frequency (**Tfidf)**. This improved our metrics by a lot given that it evaluates how important a word is in the document.
- We experimented with different *C* parameters (0.1, 10, 100) but to no avail. We also changed the loss functions but the algorithm could not converge. Hence, we chose to implement sklearn's LinearSVC with the default parameters.

At the time of writing our beat the benchmark method is ranking 4 with two submissions.

Statistic Measure	SVM (BoW)	Random Forest (BoW)	SVM (SVD)	Random Forest (SVD)	My Method
Accuracy	0.95388881	0.93038150	0.90776868	0.92231316	0.97595599
Precision	0.95339697	0.93432877	0.90165589	0.92138342	0.97463160
Recall	0.94550724	0.91409036	0.89305868	0.90397534	0.97296178
F-Measure	0.94930727	0.92343407	0.89718359	0.91206345	0.97378738

Requirement 2 - Nearest Neighbor Search and Duplicate Detection

For this requirement we used sklearn's pairwise metrics cosine similarity and jaccard similarity. For the exact jaccard and LSH cosine we implemented our own algorithm. For the LSH Jaccard we used the MinHash algorithm that was recommended in the assignment's tips. For the cosine similarity the preprocessing involved removing stopwords. That is not true for the jaccard similarity.

Cosine Similarity

The exact cosine similarity done using sklearn's library is a lot more optimized than our method, thus the shorter total times compared to the LSH cosine similarity.

In the LSH cosine the higher the *K* the less the duplicates found. This is normal based on the operation of the LSH structure. By increasing the *K* we create more random hyperplanes and thus more buckets so the queries that are kind of similar end up hashing on different ones resulting in losing some duplicates.

Jaccard Similarity

As in the cosine similarity, our own implementation is not optimized compared to the one we took from datasketch. The LSH Jaccard for 64 permutations is faster than the exact jaccard by a factor of 10. Also, the MinHash LSH Jaccard seems to be finding more duplicates than the exact one. We believe this has to do with the number of random permutations.

Question 2a: De-duplication with Locality Sensitive Hashing

Type	Build Time	Query Time	Total Time	# Duplicates	Parameters
Exact-Cosine	-	7.2177 sec	7.2177 sec	1728	-
Exact-Jaccard	-	8260.67 sec	8260.67 sec	4107	-
LSH-Cosine	45.29 sec	210.38 sec	255.67 sec	1566	K = 1
LSH-Cosine	45.09 sec	75.26 sec	120.35 sec	1426	K = 2
LSH-Cosine	37.14 sec	36.65 sec	73.79 sec	1348	K = 3
LSH-Cosine	37.23 sec	20.30 sec	57.53 sec	1243	K = 4
LSH-Cosine	39.80 sec	11.49 sec	51.29 sec	1100	K = 5
LSH-Cosine	37.57 sec	7.42 sec	44.99 sec	1069	K = 6
LSH-Cosine	37.64 sec	5.65 sec	43.30 sec	965	K = 7
LSH-Cosine	37.75 sec	4.84 sec	42.59 sec	910	K = 8
LSH-Cosine	37.74 sec	4.46 sec	42.21 sec	843	K = 9

LSH-Cosine	37.76 sec	4.22 sec	41.99 sec	795	K = 10
LSH-Jaccard	541.92 sec	81.06 sec	622.98 sec	5364	Perm = 16
LSH-Jaccard	707.54 sec	56.81 sec	764.35 sec	5362	Perm = 32
LSH-Jaccard	933.88 sec	77.04 sec	1010.92 sec	5365	Perm = 64

Question 2b: Same Question Detection

We used 2 different string similarity metrics, the first being Jaccard similarity and the second a fuzzy string similarity metric based on Levenshtein distance which is the minimum number of characters which need to change in order for two strings to become identical. At the time of writing this report we are 8th on the Kaggle leaderboard

We used these two methods along with simple TfIdf vectorization and achieved the following results:

Method	Accuracy	Recall	F-Measure	Precision
SVM (TfIdf + J + F)	0.7838829128033756	0.7645194549855236	0.7666338247592719	0.7691512958207491

Method	Accuracy	Recall	F-Measure	Precision
SVM (TfIdf + F)	0.7522685133733437	0.7108306482490578	0.7190323616988417	0.7417648080900642

Method	Accuracy	Recall	F-Measure	Precision
SVM (TfIdf + J)	0.7561553889755768	0.7188310162429138	0.7263554973379323	0.7439743795969307

Requirement 3 - Sentiment Analysis

The classifier method we used was the one we used for 'Beat my Benchmark' in **Question 1b.**

For the Deep Learning method we created a keras model consisting of 3 layers: An embedding layer, a 1D convolution and a LSTM layer. We trained our model for 50 epochs with a small starting learning rate (0.0001) to achieve better results (With higher learning rates the

classifier overfits the dataset in a few epochs). Note here that we used the training loss (binary_crossentropy) to evaluate our model and not the Kaggle metric (plain accuracy) because accuracy tends to give inconsistent results where the classifier achieves a high accuracy on the test set but is not robust (especially if the test set remains the same through the evaluation phase) and may generally have poor performance on new unseen data. Moreover the dataset size was mediocre in terms of deep learning training requirements and our results pretty much match those of non deep learning techniques. We have included the kaggle notebook which we used to create the keras model. At the time of writing this report we are 10th on the Kaggle leaderboard

Method	Accuracy	Recall	F-Measure	Precision
SVM (Tfldf)	0.88896	0.88896	0.88894987739	0.88910363018
Keras CNN+LSTM	0.89336	0.89336	0.89335249602	0.89346229292