A Project Report

On

Sentiment Analysis

BY

Vashist SLN 2017B3A70381H

Suhas Reddy N. 2017B4A70885H

Keshav Kabra 2018AAPS0527H

Under the supervision of

Dr. Lov Kumar

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS OF

CS F366: LAB PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(05-2021)

ACKNOWLEDGMENTS

As part of our project we were involved with the algorithms used, code written and their application together to industry standard data. We needed to acquire a bit of extra knowledge in the fields Machine Learning and Information Retrieval and had to better understand the techniques. It was a challenging experience to practically implement the algorithms learned and we learned more about the common issues faced by entry level data scientists.

We would like to thank Dr. Lov Kumar and Dr. Mrityunjay Singh for providing us with this great opportunity to gain hands-on experience in this field of research.

It has been very heavy but at the same time really rewarding to work with the State-of-the-Art Machine Learning algorithms.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Sentiment analysis** " submitted by **Mr. Vashist SLN (2017B3A70381H), Mr.Suhas Reddy N. (2017B4A70885H) and Mr. Keshav Kabra (2018AAPS0527H)** in fulfillment of the requirements of the course CS F366, Lab Oriented Project Course, embodies the work done by them under my supervision and guidance.

Date: 28-04-2021

(Dr. Lov Kumar)

BITS- Pilani, Hyderabad Campus

(Dr. Mrityunjay Singh)

BITS- Pilani, Hyderabad Campus

Motivation

Twitter has over 330 million active users(monthly), which allows businesses to reach a wider audience and connect with customers without intermediaries. On the downside, there's so much information that it's difficult for brands to detect negative social mentions that could harm their business. Another example would be the people's sentiments towards a newly elected political leader through their tweets.

This is where sentiment analysis comes in. It involves monitoring emotions in public conversations on social media platforms, and has become a key strategy in social media marketing.

Mining the data that these tweets contain gives the wielder an edge over competitors, allowing them to change their tactics to better their status quo.

For our Analysis we have used a standard Twitter Sentiment Analysis dataset from Kaggle, and generalised the trained model for a wider use.

Abstract

In this project we aim to build a Machine Learning Model to predict the sentiment which the tweet projects. We analyse the dataset containing tweets and corresponding sentiments using Classification Algorithms. We Classify the tweets based on the usage of Hate Speech, as Racist or Non-Racist. Our Preprocessing involves different word and sentence embedding techniques. We then used class balancing techniques to handle class imbalance. We have used the SMOTE techniques to restore class balance.

We then used some widely popular Feature Selection techniques to identify the most important features for our classification problem. After getting the important features, we went ahead and applied 11 different models, some of which are Multinomial Naive Bayes, Logistic Regression, Bernoulli Naive Bayes to name a few. We see the best combination of word embedding techniques, feature selection techniques and ML models to obtain the highest accuracy and highest Area Under the curve, and compare the accuracies of the different models We have used different metrics to do a more comprehensive comparative analysis of the models and the techniques used.

CONTENTS

Word Embedding:

Word embedding captures context of a word from the dataset, semantic and syntactic similarity and relation with other words.They are vector representations of a word. A Word Embedding format generally tries to map a word using a dictionary to a vector. Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between vectors of similar context words should be close to 1.

Why we need word embedding:

We observe that almost every algorithm used for sentiment analysis tends to work with numbers rather than text in its pure form. Seeing as this is the case, we felt the need to transform our data and obtain the numerical representation of our input data to be able to feed them into the algorithms which we plan to use for highly accurate classifications into different sentiments. Similar to Naive Bayes Algorithm for sentiment analysis, where we have binary valued feature for each unique word in the training set, we use other techniques which are not as 'naive'. For this, we have used the widely used word embedding techniques as mentioned below.

We have preprocessed our data and analysed using the Following Word Embedding techniques:

1. CBOW (Continuous Bag of Words)
2. SKG (Skip k gram embedding)
3. TF-IDF (Term Frequency Inverse Document Frequency)
4. GLOVE (Global Vectors for Word Representation)
5. GPT2 (Generative Pre-Training Model)
6. BERT (Bidirectional Encoder Representations from Transformer)
7. FAT

Continuous Bag of Words (CBOW) :

This is an architecture which uses neural networks to learn the word representations for each word.

This architecture uses future n words and past n words as well to create a word embedding.

The distributed reps of the context of the sentence are used to predict the word in the middle of the sentence. That is, the words surrounding a given word are used to predict the middle word. (neighbours)

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+n})$$

The above function is the objective function for CBOW. **It is unsupervised.**

Output snippet of CBOW embedding:

	A	B	C	D	E	F	G	H	I	J	K	L	M	
1	0.075288	-0.01758	0.140742	0.19071	-0.32246	0.012178	0.063018	-0.03004	0.076754	-0.02436	-0.09337	-0.02505	0.0045	0
2	0.053498	0.024916	0.086336	0.20038	-0.3536	-0.03351	0.124904	-0.08346	0.092639	-0.034	-0.08454	0.002725	0.048137	0
3	0.122161	-0.06528	0.103069	0.165504	-0.21475	-0.07889	-0.01612	0.027157	0.11197	-0.01099	-0.11928	-0.08787	0.015198	0
4	0.22022	0.013349	0.166815	0.16231	-0.28574	-0.00041	-0.01102	0.136571	0.125344	-0.02622	-0.03762	-0.0788	-0.10031	0
5	0.185686	-0.0441	0.100408	0.30571	-0.30867	-0.02716	-0.01529	0.014856	0.089417	0.062018	-0.08603	-0.10857	-0.02835	0
6	0.133843	0.07918	0.179201	0.208608	-0.36266	0.046368	-0.01563	0.057967	0.012434	-0.04999	-0.08745	-0.06612	-0.03333	0
7	-0.01268	-0.01019	0.035219	0.164428	-0.30144	-0.05622	0.152278	-0.13061	0.127783	-0.03028	-0.0927	0.007881	0.065059	0
8	0.118838	0.017218	0.142144	0.197104	-0.37154	0.020593	-0.02477	0.091007	-0.03982	-0.04952	-0.04463	-0.05659	-0.03723	0
9	0.243936	-0.06577	0.160935	0.20083	-0.29037	-0.01277	-0.02568	0.107671	0.164972	-0.00481	-0.04702	-0.08715	-0.07443	0
10	0.013428	-0.00183	0.051092	0.18599	-0.32366	-0.0493	0.135	-0.12111	0.111111	-0.02568	-0.08284	-0.00355	0.065723	
11	0.213513	-0.09053	0.099532	0.290641	-0.32476	-0.06338	0.000015	0.019311	0.047022	0.096538	-0.0563	-0.11412	-0.03196	0
12	0.234092	-0.04702	0.172933	0.222373	-0.31783	-0.00947	-0.00971	0.124391	0.107705	0.003669	-0.05895	-0.0768	-0.0879	0
13	0.083026	0.086367	0.165177	0.15527	-0.32595	0.009922	-4.2E-05	0.03146	0.013972	-0.06897	-0.10271	-0.03347	-0.00934	0
14	0.07068	-0.01752	0.084306	0.215746	-0.36079	-0.03258	0.089247	-0.03003	0.059651	-0.02685	-0.09416	-0.04243	0.015163	
15	0.179532	-0.03756	0.060811	0.299418	-0.3528	-0.01969	-0.00676	0.035061	0.061069	0.014076	-0.06596	-0.08848	-0.01836	0
16	0.164425	0.074326	0.09496	0.296798	-0.35525	0.03044	-0.00503	0.066648	0.020029	-0.00969	-0.01522	-0.07895	-0.0149	0
17	0.21801	-0.05549	0.050008	0.178255	-0.25644	-0.06665	0.015398	-0.0104	-0.05988	0.073992	0.083795	-0.04225	0.093733	-(
18	0.188131	-0.04401	0.056049	0.293934	-0.35335	-0.04236	0.034826	-0.02587	0.170932	0.011167	-0.02873	-0.03929	0.005414	0
19	0.083042	0.028843	0.128149	0.225173	-0.3665	0.009762	0.076322	-0.04554	0.085271	-0.04801	-0.08652	-0.02101	0.016489	
20	0.126563	0.066026	0.193766	0.174185	-0.35612	0.073096	-0.02557	0.020718	-0.00633	-0.05053	-0.02812	-0.02741	-0.00713	0
21	0.136265	0.05599	0.157464	0.224175	-0.37465	0.054596	0.004495	0.066286	-0.01569	-0.05423	-0.0676	-0.04481	-0.04492	0
22	0.149409	-0.006	0.126466	0.252958	-0.37298	0.044793	-0.01131	0.062228	0.037749	-0.0559	-0.03799	-0.08801	-0.01768	0
23	0.116653	0.004664	0.165336	0.174233	-0.30151	-0.01684	-0.02288	0.066386	-0.00064	-0.03231	-0.10296	-0.06356	-0.04988	0
24	-5.1E-05	-0.00219	0.058203	0.184433	-0.32368	-0.02958	0.145286	-0.12738	0.120884	-0.02923	-0.09069	0.003214	0.064531	0
25	0.012141	-0.03313	0.033757	0.181479	-0.30378	-0.06626	0.145484	-0.11738	0.145203	-0.021	-0.09709	-0.00945	0.058339	
26	0.233401	-0.10716	0.129028	0.198047	-0.26826	-0.03015	-0.02226	0.125546	0.164546	-0.01279	-0.05282	-0.10453	-0.0595	0
27	0.156849	-0.06442	0.106679	0.217292	-0.31585	-0.04769	0.03444	0.024514	0.150233	-0.01888	-0.08907	-0.0626	-0.02748	0
28	0.067802	0.009891	0.125558	0.171405	-0.32019	0.013597	-0.02308	0.058503	-0.04122	-0.07449	-0.09817	-0.07262	-0.0176	0
29	0.057324	-0.001	0.157078	0.146332	-0.27092	-0.02389	0.039101	-0.05203	0.04992	-0.00967	-0.12182	-0.04464	0.00989	0
30	0.123559	0.062239	0.183462	0.196592	-0.35604	0.062851	0.004432	0.073665	-0.0452	-0.05476	-0.0631	-0.04663	-0.05033	0
31	0.10034	0.102191	0.126999	0.219632	-0.36512	0.034944	0.002166	-0.00288	-0.04466	-0.06187	-0.01803	0.0064	0.024106	-(
32	0.020749	0.024759	0.078533	0.186685	-0.33828	-0.02632	0.129521	-0.08952	0.095877	-0.04093	-0.08266	0.008238	0.046762	0
33	0.088547	0.009564	0.093285	0.2253	-0.37139	-0.01767	0.057861	-0.0047	0.046057	-0.04078	-0.07831	-0.03398	-0.00268	0

Skip K Gram Embedding (SKG) :

It is one of the unsupervised techniques used to learn the most relevant words for a given word. It estimates the context word given the target word. It is, in a sense, the reverse of the CBOW algo.

The input and output are interchanged.

The functions shown below are the probability and the loss functions respectively in that order.

$$p(w_{c,j} = w_{O,c}|w_I) = \frac{\exp u_{c,j}}{\sum_{j'=1}^{V} \exp u_{j'}}$$

$$\mathcal{L} = -\log \mathbb{P}(w_{c,1}, w_{c,2}, \ldots, w_{c,C}|w_o) = -\log \prod_{c=1}^{C} \mathbb{P}(w_{c,i}|w_o)$$

$$= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j^*})}{\sum_{j=1}^{V} \exp(u_{c,j})} = -\sum_{c=1}^{C} u_{c,j^*} + \sum_{c=1}^{C} \log \sum_{j=1}^{V} \exp(u_{c,j})$$

Though both the techniques described above are similar in many ways, and just their inputs and outputs are interchanged, they differ significantly in many of the other parameters for choosing them, one of them being the training time taken. So, according to one's requirement, one should make the choice between the two techniques described.

On a more serious note, these two techniques are together used to get the word2vec representation that we see so much in use.

Output snippet of SKG embedding:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0.140252	-0.02611	0.190669	0.196285	-0.37017	-0.04373	0.028434	0.072665	-0.03352	-0.05507	-0.13723	-0.08908	-0.12015
2	0.098773	0.002387	0.147185	0.244397	-0.40977	-0.07361	0.0567	0.024713	-0.02386	-0.0467	-0.16986	-0.09186	-0.11002
3	0.156062	0.089258	0.101921	0.046664	-0.24875	-0.0027	-0.09629	0.134507	0.09792	-0.0957	-0.09988	-0.0906	-0.02419
4	0.133227	-0.00442	0.129921	0.084953	-0.24387	-0.01033	0.034307	0.073407	0.05232	-0.05806	-0.13159	-0.04391	-0.16678
5	0.134166	0.005608	0.14751	0.184278	-0.36177	-0.04993	0.011492	0.093552	-0.0435	-0.02967	-0.13024	-0.16971	-0.07344
6	0.093092	-0.00397	0.150402	0.253335	-0.37478	-0.0385	0.033251	0.002662	-0.04017	-0.03465	-0.14067	-0.14525	-0.04113
7	0.106449	-0.02018	0.138186	0.210607	-0.40659	-0.13241	0.057533	0.089236	-0.03842	-0.03474	-0.19298	-0.09993	-0.14854
8	0.08744	-0.01075	0.137681	0.246064	-0.3882	-0.07055	0.039681	0.0595	-0.04185	-0.03548	-0.14213	-0.14882	-0.05513
9	0.185539	0.007893	0.160292	0.155215	-0.33645	-0.02985	0.022764	0.120503	-0.04103	-0.06906	-0.13523	-0.09927	-0.13071
10	0.115454	-0.01961	0.136506	0.232881	-0.40322	-0.12168	0.049982	0.07177	-0.04806	-0.04459	-0.17948	-0.13275	-0.11608
11	0.20784	0.00125	0.193386	0.164262	-0.26877	-0.05406	0.008895	0.059549	-0.10746	0.004029	-0.11703	-0.21351	-0.0537
12	0.192473	-0.0089	0.177927	0.180686	-0.3555	-0.02879	0.062449	0.138176	-0.05941	-0.05408	-0.13811	-0.1087	-0.13956
13	0.083638	-0.0204	0.127058	0.233067	-0.35635	-0.03217	0.015671	0.023806	-0.0057	-0.03961	-0.1452	-0.14641	-0.07388
14	0.118	0.009069	0.184917	0.224327	-0.39634	-0.05281	0.044562	0.096401	-0.0758	-0.03354	-0.1517	-0.17206	-0.09216
15	0.149367	0.038532	0.137784	0.200872	-0.37736	-0.04149	-0.01165	0.109027	-0.06943	-0.03009	-0.1439	-0.22194	-0.08376
16	0.090934	0.115061	0.076444	0.230021	-0.28115	-0.03487	0.045755	0.099957	-0.07953	0.015285	-0.07698	-0.1861	-0.01586
17	0.131487	-0.04977	0.00106	0.103331	-0.3321	-0.03927	-0.06393	0.095676	-0.14246	-0.01284	0.072665	-0.15114	0.076245
18	0.15679	0.011967	0.111609	0.224619	-0.4179	-0.051	0.068017	-0.00179	0.046121	-0.04201	-0.12034	-0.06629	-0.06625
19	0.122677	-0.02347	0.123054	0.231227	-0.38808	-0.06082	0.061137	0.059701	-0.03603	-0.05623	-0.14863	-0.11941	-0.09899
20	0.09799	-0.0434	0.114367	0.194765	-0.37021	-0.06549	0.004923	0.050322	-0.06327	-0.07439	-0.14438	-0.15461	-0.09654
21	0.0838	-0.01842	0.126353	0.246753	-0.38745	-0.04216	0.062914	0.044157	-0.03081	-0.05197	-0.11731	-0.10948	-0.06119
22	0.126228	-0.03574	0.117509	0.223493	-0.34193	-0.02539	0.09731	0.08548	-0.03678	-0.0384	-0.1297	-0.12617	-0.04116
23	0.128754	-0.01433	0.14606	0.190139	-0.33627	-0.07601	0.015648	0.102338	-0.0247	-0.07412	-0.10811	-0.11424	-0.13537
24	0.117221	0.003138	0.160577	0.225891	-0.42204	-0.0819	0.071287	0.053773	-0.06812	-0.03882	-0.18352	-0.09757	-0.11615
25	0.162001	-0.02516	0.122934	0.183197	-0.38577	-0.08897	0.037875	0.143333	-0.00698	-0.03707	-0.18552	-0.15948	-0.13845
26	0.215385	-0.00381	0.158782	0.159773	-0.32558	-0.03484	0.033579	0.157942	-0.00876	-0.05884	-0.11022	-0.10235	-0.12429
27	0.203203	-0.02807	0.144054	0.167037	-0.34755	-0.05559	0.035577	0.145639	0.001292	-0.05297	-0.12945	-0.11237	-0.14633
28	0.084114	0.000126	0.187754	0.214972	-0.38779	-0.07333	0.053285	0.087967	-0.09457	-0.07644	-0.1582	-0.17779	-0.06852
29	0.122796	0.016252	0.157374	0.110517	-0.25807	-0.08204	-0.03677	0.093003	-0.06451	-0.05651	-0.09857	-0.08312	-0.17419
30	0.094753	-0.01582	0.151565	0.232341	-0.35644	-0.02768	0.090243	0.068709	-0.07702	-0.05906	-0.14061	-0.10926	-0.08157
31	0.095007	0.020891	0.143459	0.2565	-0.40419	-0.06236	0.062496	0.020774	-0.05483	-0.08345	-0.13365	-0.09342	-0.02491
32	0.114	-0.01485	0.130313	0.220688	-0.3952	-0.0976	0.063975	0.06959	-0.03875	-0.05581	-0.18525	-0.1192	-0.13344

Global Vectors for Word Representation (GloVe)

This powerful technique is used to obtain vector representations for words by performing averaged word-to-word co-occurrence statistics on a Global Level. Unlike other techniques, this does not just consider the local context of the words in the corpus. In this Unsupervised Learning algorithm, similarity is measured in terms of the hidden factors between the words.

The basic outline of the model can be understood by noticing that ratios of word-word co-occurrence probabilities can be encoded to get some meaning in terms of vectors.

It is defined as a log-bilinear model with an objective of weighted least-squares.

The Cost function is defined as:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

$$\underbrace{w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})}$$

measures the similarity of the hidden factors between both words to predict co-occurrence count

where

X_{ij} tabulate the number of times word j occurs in the context of word i.

Term Frequency Inverse Document Frequency(TF-IDF):

This is another frequency based method, it takes the occurrence of a word in the single document and the whole corpus into account. Common words like ‘is’, ’the’, ’a’, etc. tend to appear quite frequently in comparison to the words which are more significant to a document. Ideally we would like to reduce the significance of these common words.

TF-IDF works by penalising these common words by assigning lower weights to them compared to other significant and important words.

Term Frequency:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Data Frequency:

$$idf(w) = log(\frac{N}{df_t})$$

TF-IDF:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

It can be applied in supervised as well as unsupervised way

Word2Vec:

Word2Vec algorithm uses a neural network model to learn similarities between words of the corpus. After training the model we can use this to find the words with similar contextual meaning. It converts distinct words of the corpus into a vector, these vectors are generated such that a simple mathematical function gives us the semantic similarity between 2 different words.

Output snippet of Word2Vec embedding:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	0.027957	0.00803	-0.09505	0.09495	-0.11842	0.015732	0.0262	-0.07888	0.129064	0.03894	0.056597	0.007094	0.105324	0.014783	-0
2	0.036371	-0.05271	0.008813	0.10929	-0.07081	0.030892	0.022241	-0.09557	0.135885	0.05273	-0.06876	-0.0118	-0.06476	-0.02295	-0
3	0.098118	0.041441	0.042051	-0.00968	-0.02925	-0.03093	0.077093	-0.07405	0.070389	0.115182	-0.03931	0.001838	-0.03641	-0.01447	0.(
4	-0.03862	0.035746	0.053836	0.077469	-0.09833	-0.05339	0.007913	-0.15389	0.03554	0.052119	0.005087	-0.08633	-0.06906	0.014906	-0
5	0.010282	-0.02591	-0.01754	0.000012	0.017684	0.022488	0.036548	-0.06012	0.09364	0.028688	0.029601	0.061256	0.104843	0.048884	-0
6	0.099428	0.026825	-0.01318	0.100693	-0.01521	-0.0208	0.018587	-0.10137	0.093933	0.095526	-0.03747	-0.02081	0.044231	0.025662	-0
7	0.051722	-0.06109	-0.02081	0.056425	-0.07662	0.090531	0.014327	0.043945	0.069605	-0.01202	-0.09676	0.038772	0.017064	-0.02138	-0
8	0.034478	-0.01838	0.020023	0.085138	-0.01069	-0.04024	-0.00396	-0.1461	0.130276	0.08326	0.045999	0.017213	-0.04796	-0.0148	-0
9	-0.00659	-0.05198	0.036597	0.043291	-0.01319	-0.05176	-0.07077	-0.11444	0.06784	0.103285	-0.02365	0.017834	-0.06205	0.005926	-0
10	-0.02113	-0.05608	0.012069	0.076794	-0.09768	0.044437	-0.02309	-0.01247	0.08193	0.09553	-0.08353	-0.00036	-0.01278	0.001179	0.(
11	-0.01392	-0.05829	-0.03898	0.070404	0.031075	-0.06428	-0.02442	-0.11675	0.170499	0.090067	-0.02654	0.034236	0.036303	0.018942	-0
12	0.000494	-0.07836	0.010312	0.041212	-0.06662	-0.03313	-0.05891	-0.10784	0.098694	0.079735	0.002583	0.067516	-0.00841	0.014101	-0
13	-0.02497	-0.08055	-0.04013	0.031079	-0.05935	-0.04753	-0.00882	-0.0796	0.112464	0.132882	-0.02407	-0.0271	-0.00698	0.008959	-0
14	0.009038	0.005814	0.030263	0.099081	-0.06374	-0.07296	-0.03161	-0.13337	0.085551	0.099585	-0.03491	0.003497	-0.02827	0.015057	-0
15	-0.03289	-0.09788	-0.01062	-0.00829	-0.03702	-0.04499	-0.07753	-0.13069	0.138957	0.07165	-0.00454	0.059416	-0.0156	-0.00844	-0
16	-0.00131	-0.05598	0.035807	0.006748	-0.04796	-0.07172	-0.04447	-0.12122	0.09772	0.104279	0.01434	0.026551	-0.03535	0.021994	-0
17	-0.00834	-0.06536	-0.00616	-0.00378	0.001733	-0.05425	0.006051	-0.09605	0.100535	0.112439	-0.06053	-0.00178	0.009955	0.031968	0.(
18	-0.00156	-0.02796	0.005138	0.114513	-0.1069	0.075717	0.122733	0.015231	0.090511	-0.03659	-0.06003	-0.02842	-0.03061	0.144562	-0
19	0.008422	-0.06577	0.019329	0.149223	-0.09152	-0.04752	-0.06	-0.04311	0.100921	0.058622	-0.0654	-0.02088	-0.0231	0.045943	-0
20	-0.02768	0.008241	0.053673	0.014444	-0.08479	-0.0425	0.004011	-0.07603	0.018628	0.012394	-0.01605	-0.11253	-0.04301	0.01513	-0
21	0.073602	-0.02793	0.02129	0.06972	0.044877	-0.03999	0.034792	-0.10864	0.109721	0.117231	0.026495	-0.13607	-0.05101	0.057111	-0
22	0.031234	-0.07119	-0.00186	0.060408	-0.03076	-0.03695	-0.048	-0.08864	0.13552	0.090974	0.025014	-0.00936	-0.03894	-0.02204	-
23	0.000173	0.025481	0.010836	0.061489	-0.05159	0.015654	0.017139	-0.10857	0.171807	0.106979	0.00986	-0.07534	-0.00519	-0.05761	-0
24	-0.00024	-0.0384	-0.02794	0.096829	-0.07698	0.061539	0.08936	0.027323	0.049183	0.042868	-0.10316	-0.00193	-0.02604	0.022371	-
25	0.029126	-0.08664	-0.01167	0.038708	-0.06368	0.016113	-0.0519	-0.0433	0.113234	0.057833	-0.04187	0.075793	0.003112	0.000317	-0
26	-0.01737	-0.09383	-0.01355	0.058594	-0.06014	-0.01477	0.00255	-0.12404	0.05053	0.147743	-0.00275	-0.01044	-0.03979	-0.00979	-0
27	0.002839	-0.07937	-0.00091	0.030872	-0.05154	-0.0441	-0.04842	-0.14371	0.112946	0.091701	-0.03468	0.037115	-0.01901	0.025079	-0
28	-0.01242	0.024855	0.030937	0.035845	0.023342	-0.0587	0.009986	-0.07386	0.113934	0.070946	0.036077	-0.02474	0.011536	-0.0548	-0
29	-0.00674	0.026963	-0.02845	0.072626	-0.0101	-0.03059	0.139945	0.004943	0.102143	0.057387	-0.01606	-0.01312	0.073916	0.01283	-0
30	0.072169	0.002771	-0.00908	0.121048	-0.02663	-0.01716	-0.03126	-0.07635	0.09673	0.080164	-0.0062	-0.06277	-0.02988	0.033894	-0
31	0.005101	0.050847	0.029794	0.05176	-0.00562	-0.04642	0.100803	-0.09942	0.022019	0.002035	-0.087	-0.03869	-0.03831	0.023672	-0
32	0.006807	-0.10082	-0.00012	0.057103	-0.09078	-0.00741	-0.03636	-0.06116	0.139624	0.073659	-0.04169	0.04605	-0.02624	-0.01264	-0
33	0.091464	0.04655	0.047046	0.076029	-0.08665	0.015363	-0.02115	-0.04114	0.079491	0.165548	-0.03544	-0.13064	-0.05845	0.053596	-0
34	0.02502	0.01277	0.01733	0.021527	0.02286	0.01423	0.013583	0.11853	0.124742	0.118495	0.01098	0.01854	0.06679	0.03674	0

The Challenge of Imbalanced Data - SMOTE technique:

On preprocessing the Sentiment analysis data, we observe that there is a huge problem of Class Imbalance. This is especially the case when this is a binary classification problem where the target is spam and not spam. **Synthetic Minority Oversampling Technique**, or **SMOTE** is used to solve this problem. Minority class data can be upsampled and may also be resampled to generate new data points belonging to the class with the lesser representation.

Samples are selected close to points in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

Original_CBOW output:

31959	0.110006	0.07548	0.186897	0.201265	-0.36997	0.042893	0.042652	0.006986	0.028534
31960	0.098697	0.043652	0.135627	0.214872	-0.33397	0.00859	-0.02507	0.040789	-0.03433
31961	0.090209	-0.04694	0.091596	0.22149	-0.31689	-0.06172	0.068371	0.012371	0.07512
31962	0.090348	-0.01795	0.084225	0.196141	-0.32439	-0.08305	0.112929	-0.08022	0.159666

SMOTE_CBOW output:

SMOTE works by selecting examples that are close in the feature space(k-nearest neighbours are selected from the dataset), drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Synthetic data is then made between the random data and randomly selected k-nearest neighbor.

59437	-0.07141	0.060949	0.086726	0.074537	0.062493	0.014785	0.064672	0.193515	-0.03971	0.065015	0.152881	0.079252	-0.03725	-0.01768	0.109623	0.002338
59438	0.114773	-0.04941	0.044879	0.029122	-0.00098	0.048685	0.151947	0.09055	0.030354	0.088724	-0.01759	-0.01555	-0.04136	0.038064	0.093486	0.029055
59439	0.1336	-0.05937	0.105743	0.047837	0.06795	0.004692	0.129879	0.043028	-0.00239	0.150302	-0.01961	-0.02755	0.035043	-0.04668	0.05369	0.014815
59440	0.098985	0.008042	-0.00495	0.023282	-0.03171	0.060037	0.145815	0.198019	0.056365	0.077592	0.05604	-0.00206	-0.07403	0.07812	0.100118	0.07178

Borderline-CBOW:

We select those instances of the minority class that are misclassified, such as with a k-nearest neighbor classification model and then oversample just those difficult instances, providing more resolution only where it may be required.

59437	0.108837	0.020791	0.12216	0.201104	-0.35399	0.033694	0.011534	0.089911	-0.0196	-0.05614
59438	0.110327	0.078911	0.180481	0.209573	-0.37326	0.01857	0.10397	-0.0467	0.109374	-0.01174
59439	0.111631	0.073896	0.16214	0.221145	-0.36182	0.04727	0.01626	0.062634	-0.01388	-0.05145
59440	0.066812	0.016816	0.101381	0.246339	-0.37374	0.006035	0.05067	-0.00681	0.018393	-0.03303

smoteen_CBOW:

This algorithm undersamples the majority instances and makes it equal to the majority class. Here, the majority class has been reduced to the total number of minority classes, so that both classes will have equal numbers of records.

52585	0.152375	-0.0116	0.111104	0.238737	-0.35616	0.013915	-0.0185	0.087257	-0.01176
52586	0.18406	-0.06828	0.084133	0.218741	-0.25812	-0.05564	-0.04551	0.149818	0.063103
52587	0.135198	0.065101	0.162794	0.23627	-0.3857	0.050008	0.014419	0.00852	0.03961

smote_to_mek_cbow:

Tomek identifies pairs of nearest neighbors in a dataset that have different classes. Removing one or both of the examples in these pairs has the effect of making the decision boundary in the training dataset less noisy or ambiguous.

59416	0.152375	-0.0116	0.111104	0.238737	-0.35616	0.013915	-0.0185	0.087257	-0.01176	-0.02268	-0.05232
59417	0.18406	-0.06828	0.084133	0.218741	-0.25812	-0.05564	-0.04551	0.149818	0.063103	-0.02634	-0.11772
59418	0.135198	0.065101	0.162794	0.23627	-0.3857	0.050008	0.014419	0.00852	0.03961	-0.04446	-0.02938

svm_cbow:

An SVM is used to locate the decision boundary defined by the support vectors and examples in the minority class that close to the support vectors become the focus for generating synthetic examples.

59438	0.078989	-0.00299	0.097847	0.222764	-0.38026	-0.00995	0.097045	-0.0489	0.100933
59439	0.15516	-0.0067	0.114095	0.313851	-0.37295	0.004393	0.060464	-0.134	0.202722
59440	0.108482	0.067237	0.180171	0.217536	-0.35668	0.073492	0.017757	0.06535	-0.05265
59441									

Feature Selection Techniques and Their Uses:

CR	CS	CT	CU	CV
-0.06771	-0.16407	0.142055	-0.05453	0.027881
-0.03103	-0.1691	0.161782	-0.04067	0.032039
-0.04873	-0.0934	0.136547	-0.06676	0.02147
-0.038	-0.08261	0.173336	-0.07676	-0.02327
0.043244	-0.04702	0.06357	-0.03802	0.048994
-0.03374	-0.10243	0.093847	-0.03432	-0.02984
-0.03678	-0.19534	0.17527	-0.02737	0.063614
-0.02912	-0.07986	0.10545	-0.07715	-0.02448
-0.00419	-0.09135	0.142969	-0.05501	0.020615
-0.03583	-0.18335	0.164883	-0.02512	0.055976
0.057471	-0.00794	0.004463	-0.12282	0.073183
0.010962	-0.09849	0.127384	-0.06061	0.029269
-0.07982	-0.1255	0.147971	-0.03741	-0.03625

This image shows the last column in the data after the class imbalance was solved

When working with data, we often end up with a lot of the data that was collected turning out to not be of any importance to our objective. In that case, if those unused features are not filtered out, they may create unwanted disturbances making our work inconclusive. So we go ahead and use standard feature selection techniques. Without any feature selection techniques applied, we have 100 features in our dataset. It might so happen that there are some features which are redundant and not helping in predicting our target variable. To get rid of those we run some widely used and well known feature selection techniques.

CS	CT	CU
0.142055	-0.05453	0.027881
0.161782	-0.04067	0.032039
0.136547	-0.06676	0.02147
0.173336	-0.07676	-0.02327
0.06357	-0.03802	0.048994
0.093847	-0.03432	-0.02984
0.17527	-0.02737	0.063614

This image shows the column number after using one feature selection technique

After applying sigtet function, we got rid of one feature only, as can be seen by the last column. In this, we perform the anova test on the features to find which one's are actually significant to our model.

sigtet function:

```
def sigtetfinal(j):
    fname=fileloc+str(j)+'n.csv'
    df=np.genfromtxt(fname,delimiter=',')
    y=df[:,-1]
    in0=np.where(y==0)
    in1=np.where(y==1)
    #in2=np.where(y==3)
    #in3=np.where(y==4)
    #in4=np.where(y==5)
    #in5=np.where(y==6)
    p=np.zeros((df.shape[1]-1))
    for i in range(0,df.shape[1]-1):
        fv=df[:,i]
        #w,p[i]=stats.f_oneway(fv[in0],fv[in1],fv[in2],fv[in3],fv[in4],fv[in5])
        w,p[i]=stats.f_oneway(fv[in0],fv[in1])
    return p,df
```

AP	AQ	AR
0.079032	-0.05453	0.027881
0.091023	-0.040668	0.032039
0.034858	-0.066759	0.02147
0.020198	-0.076763	-0.023267

After applying coranaf function (user written), we have managed to come down to 44 features which are important to us. This function checks which features are uncorrelated to our problem of classification.

coranaf function:

```
[ ]  def coranaf(df,sigfn):
        corr=np.zeros((sigfn.shape[0],sigfn.shape[0]))
        for i in range(0,sigfn.shape[0]):
            for j in range(0,sigfn.shape[0]):
                a=np.corrcoef(df[:,sigfn[i]],df[:,sigfn[j]])
                corr[i,j]=a[0,1]
        sg=crosscorr(corr,sigfn.shape[0])
        return sg
```

H	I	J
0.061198	-0.022389	-0.029091
-0.038456	-0.010684	-0.009327
-0.088439	-0.084475	0.06622
0.046863	-0.014336	-0.013679
-0.006273	-0.052407	-0.046735

After applying principal component analysis, we end up with 10 (only 10) important features which are of significance to us which is a drastic drop. The principal component analysis is a dimensionality reduction method which combines large sets of data to reduce the number of dimensions while still retaining much of the information present in those large sets of data. PCA just reduces the number of variables in the given dataset while still retaining as much information as possible.

We are just showing the snippets for some files, but we have obtained similar outputs for others too.

Classification Machine Learning Algorithms:

After Feature Selection, all the outputs including the word embedding have been passed through 11 chosen standard binary classification models to find the optimal classifier for the Dataset.

The process included choosing 4000 random data points from the train data (random because we need proper representation of both the classes) and running all the outputs including the word embedding ones (The 4000 data points are chosen from these files, which give a general representation of data for analysis of best combination -- Computationally less expensive). Then, we end up doing a 70:30 Train-Test split, training the models on the Train Data, and validating over the Test Data, over and over again.

Multinomial Naive Bayes:

Multinomial Naive Bayes algorithm is a probabilistic learning algorithm used in Natural Language Processing. The basis of the algorithm is Bayes' Theorem where the posterior probability is calculated for different words. The assumption is that the features and conditionally independent of each other.

Bernoulli Naive Bayes:

This implements the NB classifier for training and classification algos for data that is distributed similar to the multivariate bernoulli distributions.

Gaussian Naive Bayes:

This is similar to the standard naive Bayes, except that the likelihood of the features is assumed to be Gaussian.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Source:

Logistic regression

It is a predictive analysis algorithm which is based on probability. This can be considered as a linear regression with a complex cost function. Sigmoid function is used to predict the probabilities of the required variables.

sigmoid function: $$f(x) = \frac{1}{1 + e^{-(x)}}$$

The cost function of logistic regression is as follows:

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Decision Tree Classifier

Here we generate a tree where the internal node represents a feature, the branch represents a decision rule and each leaf node represents the outcome. This classifies on the basis of attribute value.

Support Vector Clustering

In SVC the data points are mapped to a high dimensional feature space using a kernel function. We have used 3 different kernel functions: Linear, polynomial and radial basis function.

On complex datasets, it is a strong and powerful classification model.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

General Loss function for SVC

Multilayer perceptron

This technique employs the power of Hierarchical Artificial Neural Networks, to map the predictions. In this Feedforward network, weights define the connections between the layers. Backpropagation is used to minimize the errors and update the weights across the multiple layers.

We use 3 different solvers: Limited-memory Broyden–Fletcher–Goldfarb–Shanno, Stochastic Gradient Descent, Adam Optimisation.

Results:

Accuracy Matrix:

We have a 63X11 matrix depicting the accuracies of the models corresponding to the data files on which the models were trained. We have the legend file documenting what dataset which number file represents which can be referred to. We show a snippet of the accuracy matrix here.

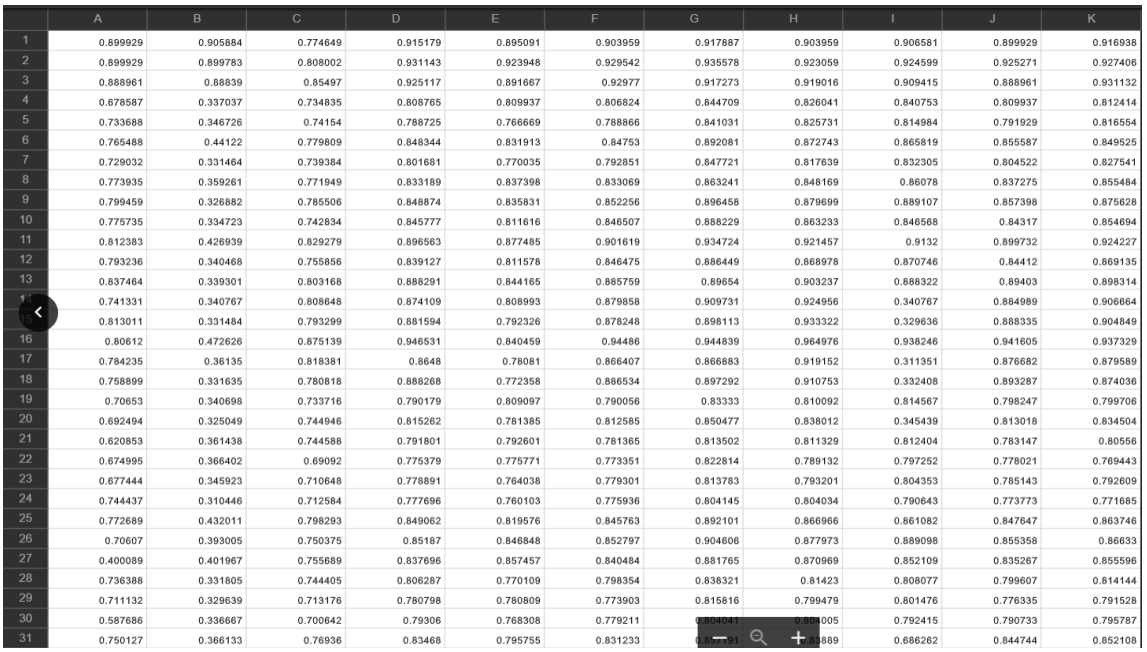
	A	B	C	D	E	F	G	H	I	J	K
1	0.9325	0.9325	0.7	0.939167	0.890833	0.934167	0.936667	0.934167	0.910833	0.9325	0.938333
2	0.9325	0.929167	0.745833	0.945	0.924167	0.945	0.94	0.9425	0.923333	0.941667	0.943333
3	0.925	0.920833	0.819167	0.935	0.891667	0.936667	0.913333	0.9375	0.900833	0.925	0.933333
4	0.679167	0.5025	0.735	0.809167	0.81	0.8075	0.845	0.826667	0.840833	0.81	0.813333
5	0.734167	0.494167	0.743333	0.789167	0.766667	0.79	0.841667	0.826667	0.815	0.7925	0.816667
6	0.7675	0.575833	0.78	0.849167	0.831667	0.848333	0.8925	0.873333	0.865833	0.855833	0.849167
7	0.729167	0.489167	0.740833	0.8025	0.77	0.794167	0.848333	0.818333	0.8325	0.805	0.8275
8	0.774167	0.515	0.7725	0.833333	0.8375	0.833333	0.863333	0.848333	0.860833	0.8375	0.855833
9	0.800833	0.4925	0.786667	0.849167	0.835833	0.8525	0.896667	0.88	0.889167	0.8575	0.875833
10	0.7775	0.495833	0.744167	0.845833	0.811667	0.846667	0.888333	0.863333	0.846667	0.843333	0.855
11	0.818333	0.575833	0.83	0.896667	0.8775	0.901667	0.935	0.921667	0.913333	0.9	0.924167
12	0.793333	0.49	0.758333	0.839167	0.811667	0.846667	0.886667	0.869167	0.870833	0.844167	0.869167
13	0.8375	0.495833	0.804167	0.888333	0.844167	0.885833	0.896667	0.903333	0.888333	0.894167	0.898333
14	0.741667	0.506667	0.809167	0.874167	0.809167	0.88	0.91	0.925	0.506667	0.885	0.906667
15	0.813333	0.4975	0.793333	0.881667	0.7925	0.878333	0.898333	0.933333	0.496667	0.888333	0.905
16	0.808333	0.6075	0.874167	0.946667	0.841667	0.945	0.945	0.965	0.938333	0.941667	0.9375
17	0.784167	0.5225	0.818333	0.865	0.780833	0.866667	0.8675	0.919167	0.48	0.876667	0.88
18	0.759167	0.494167	0.780833	0.888333	0.7725	0.886667	0.8975	0.910833	0.499167	0.893333	0.875
19	0.7075	0.503333	0.735833	0.790833	0.809167	0.790833	0.834167	0.810833	0.815	0.799167	0.8
20	0.6925	0.490833	0.745833	0.815833	0.781667	0.813333	0.850833	0.838333	0.510833	0.813333	0.835
21	0.644167	0.524167	0.746667	0.7925	0.7925	0.7825	0.814167	0.811667	0.8125	0.784167	0.805833
22	0.675	0.5125	0.691667	0.775833	0.775833	0.774167	0.823333	0.79	0.7975	0.778333	0.77
23	0.678333	0.504167	0.7125	0.78	0.764167	0.780833	0.815	0.794167	0.805	0.785833	0.794167
24	0.746667	0.479167	0.715	0.778333	0.76	0.776667	0.804167	0.804167	0.790833	0.774167	0.7725
25	0.776667	0.575833	0.8	0.85	0.82	0.846667	0.8925	0.8675	0.860833	0.848333	0.864167
26	0.718333	0.543333	0.751667	0.8525	0.8475	0.853333	0.905	0.878333	0.889167	0.855833	0.866667
27	0.558333	0.559167	0.758333	0.839167	0.8575	0.841667	0.8825	0.871667	0.8525	0.836667	0.856667
28	0.736667	0.485833	0.746667	0.806667	0.77	0.799167	0.839167	0.815	0.808333	0.8	0.814167
29	0.7125	0.495	0.714167	0.781667	0.780833	0.775	0.816667	0.800833	0.801667	0.7775	0.791667
30	0.6325	0.5	0.703333	0.793333	0.768333	0.78	0.804167	0.804167	0.7925	0.790833	0.795833
31	0.750833	0.515833	0.770833	0.835	0.795833	0.831667	0.8575	0.839167	0.705	0.845	0.8525

F-Score Matrix:

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

As with the accuracy matrix, the F-Score matrix has the exact same dimensions. F-score is the measure of a model's accuracy on a dataset. This is just another

important metric included for pinpointing the models which are consistent even with different metrics.



	A	B	C	D	E	F	G	H	I	J	K
1	0.899929	0.905884	0.774649	0.915179	0.895091	0.903959	0.917887	0.903959	0.906581	0.899929	0.916938
2	0.899929	0.899783	0.808002	0.931143	0.923948	0.929542	0.935578	0.923059	0.924599	0.925271	0.927406
3	0.888961	0.88839	0.85497	0.925117	0.891667	0.92977	0.917273	0.919016	0.909415	0.888961	0.931132
4	0.678587	0.337037	0.734835	0.808765	0.809937	0.806824	0.844709	0.826041	0.840753	0.809937	0.812414
5	0.733688	0.346726	0.74154	0.788725	0.766669	0.788866	0.841031	0.825731	0.814984	0.791929	0.816554
6	0.765488	0.44122	0.779809	0.848344	0.831913	0.84753	0.892081	0.872743	0.865819	0.855587	0.849525
7	0.729032	0.331464	0.739384	0.801681	0.770035	0.792851	0.847721	0.817639	0.832305	0.804522	0.827541
8	0.773935	0.359261	0.771949	0.833189	0.837398	0.833069	0.863241	0.848169	0.86078	0.837275	0.855484
9	0.799459	0.326882	0.785506	0.848874	0.835831	0.852256	0.896458	0.879699	0.889107	0.857398	0.875628
10	0.775735	0.334723	0.742834	0.845777	0.811616	0.846507	0.888229	0.863233	0.846568	0.84317	0.854694
11	0.812383	0.426939	0.829279	0.896563	0.877485	0.901619	0.934724	0.921457	0.9132	0.899732	0.924227
12	0.793236	0.340468	0.755856	0.839127	0.811578	0.846475	0.886449	0.868978	0.870746	0.84412	0.869135
13	0.837464	0.339301	0.803168	0.888291	0.844165	0.885759	0.89654	0.903237	0.888322	0.89403	0.898314
14	0.741331	0.340767	0.808648	0.874109	0.808993	0.879858	0.909731	0.924956	0.340767	0.884989	0.906664
15	0.813011	0.331484	0.793299	0.881594	0.792326	0.878248	0.898113	0.933322	0.329636	0.888335	0.904849
16	0.80612	0.472626	0.875139	0.946531	0.840459	0.94486	0.944839	0.964976	0.938246	0.941605	0.937329
17	0.784235	0.36135	0.818381	0.8648	0.78081	0.866407	0.866883	0.919152	0.311351	0.876682	0.879589
18	0.758899	0.331635	0.780818	0.888268	0.772358	0.886534	0.897292	0.910753	0.332408	0.893287	0.874036
19	0.70653	0.340698	0.733716	0.790179	0.809097	0.790056	0.83333	0.810092	0.814567	0.798247	0.799706
20	0.692494	0.325049	0.744946	0.815262	0.781385	0.812585	0.850477	0.838012	0.345439	0.813018	0.834504
21	0.620853	0.361438	0.744588	0.791801	0.792601	0.781365	0.813502	0.811329	0.812404	0.783147	0.80556
22	0.674995	0.366402	0.69092	0.775379	0.775771	0.773351	0.822814	0.789132	0.797252	0.778021	0.769443
23	0.677444	0.345923	0.710648	0.778891	0.764038	0.779301	0.813783	0.793201	0.804353	0.785143	0.792609
24	0.744437	0.310446	0.712584	0.777696	0.760103	0.775936	0.804145	0.804034	0.790643	0.773773	0.771685
25	0.772689	0.432011	0.798293	0.849062	0.819576	0.845763	0.892101	0.866966	0.861082	0.847647	0.863746
26	0.70607	0.393005	0.750375	0.85187	0.846848	0.852797	0.904606	0.877973	0.889098	0.855358	0.86633
27	0.400089	0.401967	0.755689	0.837696	0.857457	0.840484	0.881765	0.870969	0.852109	0.835267	0.855596
28	0.736388	0.331805	0.744405	0.806287	0.770109	0.798354	0.838321	0.81423	0.808077	0.799607	0.814144
29	0.711132	0.329639	0.713176	0.780798	0.780809	0.773903	0.815816	0.799479	0.801476	0.776335	0.791528
30	0.587686	0.336667	0.700642	0.79306	0.768308	0.779211	0.8604041	0.801005	0.792415	0.790733	0.795787
31	0.750127	0.366133	0.76936	0.83468	0.795755	0.831233	0.85[illegible]01	0.[illegible]889	0.686262	0.844744	0.852108

Precision Matrix:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Source: Accuracy, Precision, Recall or F1? | by Koo Ping Shung | Towards Data Science

	A	B	C	D	E	F	G	H	I	J	K
1	0.869556	0.906013	0.924365	0.942892	0.899682	0.938508	0.92065	0.938508	0.902794	0.869556	0.928646
2	0.869556	0.881271	0.928441	0.938089	0.923732	0.940529	0.932987	0.941713	0.925956	0.932518	0.936524
3	0.855625	0.866708	0.917303	0.923387	0.891667	0.927404	0.922085	0.932797	0.92138	0.855625	0.929384
4	0.68067	0.750213	0.735472	0.811543	0.810324	0.81155	0.847371	0.83111	0.841404	0.810324	0.819216
5	0.735474	0.471633	0.749459	0.791153	0.766675	0.795609	0.846538	0.833045	0.81504	0.795222	0.81721
6	0.767295	0.691192	0.779687	0.849557	0.832456	0.848663	0.892884	0.873767	0.865807	0.855612	0.850879
7	0.73197	0.665596	0.751331	0.812411	0.771456	0.807523	0.85908	0.828024	0.837329	0.812011	0.827721
8	0.77651	0.662384	0.776853	0.83572	0.839452	0.836997	0.865477	0.851204	0.862366	0.840849	0.861182
9	0.812118	0.750486	0.795475	0.8536	0.836442	0.856446	0.901717	0.885796	0.891128	0.859714	0.879978
10	0.789145	0.678457	0.751389	0.847162	0.812722	0.849299	0.890978	0.865461	0.84855	0.845947	0.859601
11	0.832097	0.756806	0.829621	0.896541	0.877472	0.901591	0.935656	0.921773	0.913254	0.899977	0.924351
12	0.793579	0.449161	0.767496	0.839285	0.81194	0.847813	0.888855	0.870636	0.871416	0.844337	0.869307
13	0.838407	0.309331	0.812427	0.889678	0.844586	0.887757	0.899796	0.906042	0.889022	0.8974	0.899273
14	0.743792	0.256711	0.813849	0.874489	0.809768	0.881135	0.913761	0.925536	0.256711	0.885023	0.906667
15	0.816002	0.750217	0.793705	0.88294	0.79383	0.879748	0.90242	0.93383	0.246678	0.888356	0.908117
16	0.807358	0.692414	0.880947	0.946665	0.841021	0.944989	0.945029	0.964974	0.938246	0.941593	0.937464
17	0.784487	0.751078	0.818543	0.865547	0.780796	0.86755	0.870826	0.919169	0.2304	0.876716	0.882273
18	0.76041	0.310512	0.780934	0.889316	0.773249	0.888584	0.900883	0.91241	0.249167	0.894081	0.887112
19	0.710281	0.750836	0.74358	0.794506	0.809622	0.795203	0.841011	0.815767	0.817971	0.80472	0.801774
20	0.692489	0.750517	0.751631	0.82221	0.784773	0.821118	0.856476	0.843116	0.260951	0.817359	0.841513
21	0.672688	0.750773	0.763439	0.802945	0.79421	0.795935	0.825492	0.819402	0.817394	0.796802	0.812684
22	0.675004	0.648997	0.693654	0.778205	0.776185	0.778331	0.827325	0.795029	0.79906	0.780007	0.772765
23	0.680203	0.601096	0.717889	0.785548	0.764685	0.788635	0.82323	0.799583	0.808933	0.789404	0.803043
24	0.763165	0.229601	0.729623	0.786723	0.760716	0.785701	0.807268	0.80864	0.795708	0.780489	0.781969
25	0.778803	0.664847	0.799937	0.850274	0.819528	0.846799	0.89263	0.867472	0.861679	0.8482	0.863976
26	0.739566	0.646481	0.75195	0.85368	0.848622	0.854188	0.906693	0.879043	0.889129	0.856533	0.86709
27	0.311736	0.753663	0.75869	0.841027	0.857425	0.842821	0.883824	0.872468	0.852355	0.8382	0.858112
28	0.736519	0.660248	0.750378	0.807062	0.770565	0.800701	0.842187	0.816808	0.80847	0.800359	0.814131
29	0.718076	0.750451	0.718441	0.787692	0.781358	0.782064	0.824301	0.811155	0.80365	0.785086	0.792023
30	0.726591	0.253769	0.712387	0.795555	0.768709	0.785091	0.805448	0.805726	0.79339	0.79183	0.796433
31	0.75308	0.697943	0.77693	0.83705	0.796077	0.834454	0.8[illegible]04	[illegible]40987	0.765473	0.846772	0.855598

Recall Matrix:

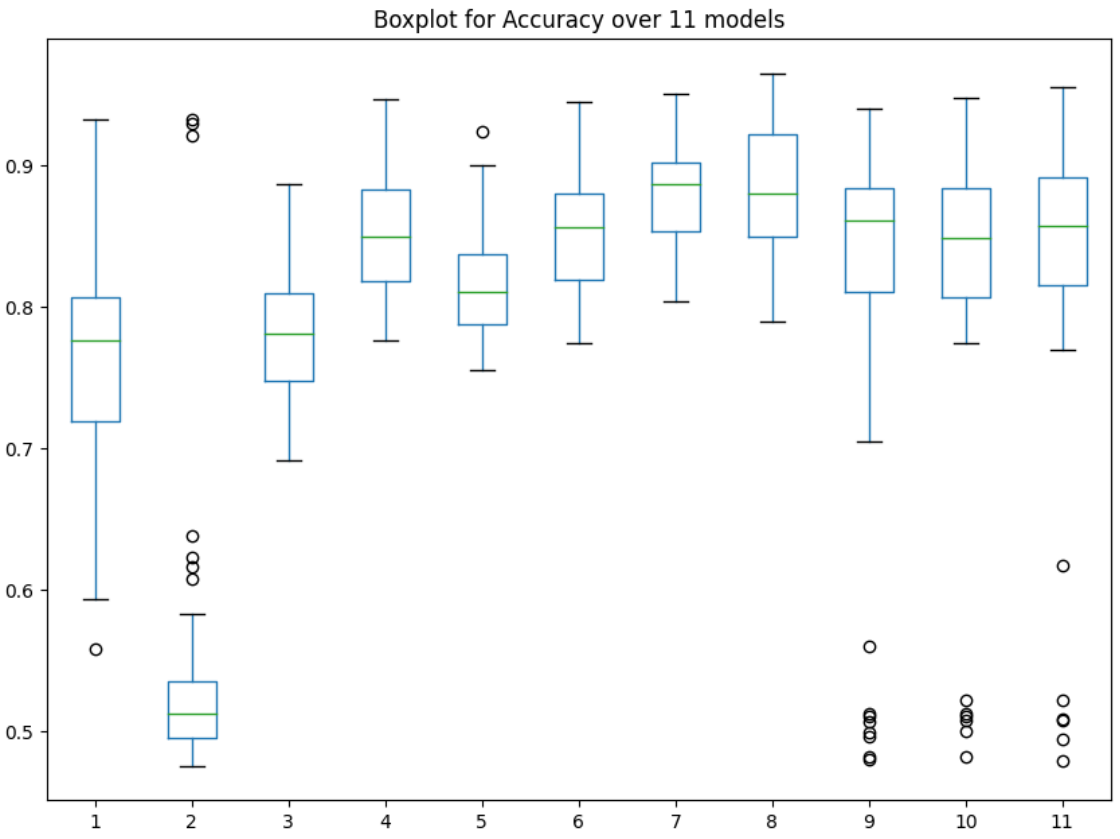
$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Source:[Accuracy, Precision, Recall or F1? | by Koo Ping Shung | Towards Data Science]

	A	B	C	D	E	F	G	H	I	J	K
1	0.9325	0.9325	0.7	0.939167	0.890833	0.934167	0.936667	0.934167	0.910833	0.9325	0.938333
2	0.9325	0.929167	0.745833	0.945	0.924167	0.945	0.94	0.9425	0.923333	0.941667	0.943333
3	0.925	0.920833	0.819167	0.935	0.891667	0.936667	0.913333	0.9375	0.900833	0.925	0.933333
4	0.679167	0.5025	0.735	0.809167	0.81	0.8075	0.845	0.826667	0.840833	0.81	0.813333
5	0.734167	0.494167	0.743333	0.789167	0.766667	0.79	0.841667	0.826667	0.815	0.7925	0.816667
6	0.7675	0.575833	0.78	0.849167	0.831667	0.848333	0.8925	0.873333	0.865833	0.855833	0.849167
7	0.729167	0.489167	0.740833	0.8025	0.77	0.794167	0.848333	0.818333	0.8325	0.805	0.8275
8	0.774167	0.515	0.7725	0.833333	0.8375	0.833333	0.863333	0.848333	0.860833	0.8375	0.855833
9	0.800833	0.4925	0.786667	0.849167	0.835833	0.8525	0.896667	0.88	0.889167	0.8575	0.875833
10	0.7775	0.495833	0.744167	0.845833	0.811667	0.846667	0.888333	0.863333	0.846667	0.843333	0.855
11	0.818333	0.575833	0.83	0.896667	0.8775	0.901667	0.935	0.921667	0.913333	0.9	0.924167
12	0.793333	0.49	0.758333	0.839167	0.811667	0.846667	0.886667	0.869167	0.870833	0.844167	0.869167
13	0.8375	0.495833	0.804167	0.888333	0.844167	0.885833	0.896667	0.903333	0.888333	0.894167	0.898333
14	0.741667	0.506667	0.809167	0.874167	0.809167	0.88	0.91	0.925	0.506667	0.885	0.906667
15	0.813333	0.4975	0.793333	0.881667	0.7925	0.878333	0.898333	0.933333	0.496667	0.888333	0.905
16	0.808333	0.6075	0.874167	0.946667	0.841667	0.945	0.945	0.965	0.938333	0.941667	0.9375
17	0.784167	0.5225	0.818333	0.865	0.780833	0.866667	0.8675	0.919167	0.48	0.876667	0.88
18	0.759167	0.494167	0.780833	0.888333	0.7725	0.886667	0.8975	0.910833	0.499167	0.893333	0.875
19	0.7075	0.503333	0.735833	0.790833	0.809167	0.790833	0.834167	0.810833	0.815	0.799167	0.8
20	0.6925	0.490833	0.745833	0.815833	0.781667	0.813333	0.850833	0.838333	0.510833	0.813333	0.835
21	0.644167	0.524167	0.746667	0.7925	0.7925	0.7825	0.814167	0.811667	0.8125	0.784167	0.805833
22	0.675	0.5125	0.691667	0.775833	0.775833	0.774167	0.823333	0.79	0.7975	0.778333	0.77
23	0.678333	0.504167	0.7125	0.78	0.764167	0.780833	0.815	0.794167	0.805	0.785833	0.794167
24	0.746667	0.479167	0.715	0.778333	0.76	0.776667	0.804167	0.804167	0.790833	0.774167	0.7725
25	0.776667	0.575833	0.8	0.85	0.82	0.846667	0.8925	0.8675	0.860833	0.848333	0.864167
26	0.718333	0.543333	0.751667	0.8525	0.8475	0.853333	0.905	0.878333	0.889167	0.855833	0.866667
27	0.558333	0.559167	0.758333	0.839167	0.8575	0.841667	0.8825	0.871667	0.8525	0.836667	0.856667
28	0.736667	0.485833	0.746667	0.806667	0.77	0.799167	0.839167	0.815	0.808333	0.8	0.814167
29	0.7125	0.495	0.714167	0.781667	0.780833	0.775	0.816667	0.800833	0.801667	0.7775	0.791667
30	0.6325	0.5	0.703333	0.793333	0.768333	0.78	0.804167	0.804167	0.7925	0.790833	0.795833
31	0.750833	0.515833	0.770833	0.835	0.795833	0.831667	0.8675	0.839167	0.705	0.845	0.8525

Box Plot



Legend:

1. MultinomialNB()
2. BernoulliNB()
3. GaussianNB()
4. LogisticRegression(solver='lbfgs')
5. DecisionTreeClassifier()
6. SVC(kernel='linear')
7. SVC(kernel='poly')
8. SVC(kernel='rbf')
9. MLPClassifier(solver='lbfgs')
10. MLPClassifier(solver='sgd')
11. MLPClassifier(solver='adam')

Note :

SVC(kernel='rbf') -- Best Classification Model based on box plots and other metrics (F-score, Accuracy, Precision, Recall) on the validation data, over all combinations of Word Embedding, Data Balancing, and Feature selection techniques.

1. Applying the above Classifier on the Coranaf Feature selection of the Smoteen Output of Word2Vec Word Embedding gave the following metrics:

```
F-score =  0.9921380754872522
Accuracy =  0.9921444321940464
Precision =  0.9921546737856787
Recall =  0.9921444321940464
AUC =  0.999632217395486
```

2. Applying the above Classifier on the Sigtet Feature selection of the Smoteen Output of Word2Vec Word Embedding gave the following metrics:

```
F-score =  0.9925544282708048
Accuracy =  0.9925578831312017
Precision =  0.9925587698433921
Recall =  0.9925578831312017
AUC =  0.9994324780151914
```

3. After applying 3-Fold Cross Validation, the results are shown below:

```
for Fold  1
F-score =  0.9792522053640035
Accuracy =  0.9791614983874969
Precision =  0.9801130959199587
Recall =  0.9791614983874969
AUC =  0.9994516184944816


for Fold  2
F-score =  0.9790026416008181
Accuracy =  0.978913420987348
Precision =  0.9799275404437243
Recall =  0.978913420987348
AUC =  0.9994838134851772


for Fold  3
F-score =  0.991385054183105
Accuracy =  0.9913787756620976
Precision =  0.9914105973922333
Recall =  0.9913787756620976
AUC =  0.999498766598436


Average Accuracy =  0.9831512316789809
Average AUC =  0.9994780661926983
```

Conclusions:

We have learnt and implemented the different word embedding techniques widely used to our tweets dataset. We came across some issues while coding and running the code, but we were able to overcome them. For some techniques, we got very large outputs and we could not proceed with them in our next steps. For example, the output of the tf_idf technique was a 11 GB file. Due to constraint of computing resources, we could not go forward with that file. Few other techniques also were not implementable due to the original constraint mentioned above. We understood the impact of having proper computational resources and could grasp that the existing techniques are computationally intensive and one should try to come up with less intensive algorithms.

We also learnt about the very important problem of class imbalance and how it affects our results and thereby the interpretation of them. We went ahead and used the standard techniques that are available. Similarly for feature selection.

From our analysis over the different combinations of Word embedding, Data balancing, Feature selection and Classification techniques, we have observed that Support Vector Classifier (with the kernel ‘rbf’) works best with the Word2Vec word embedding, after the optimal features are selected, along with the Smoteen Data balancing technique.

Our work is in no way comprehensive, but it is a start to understand the techniques actually used and not just the math and logic behind them. Having said that, we did not do too badly. We narrowed down the combination of techniques to achieve 98.31% accuracy. There might be better techniques to bring it even closer to 1, which we hope to cover in the future.

References:

1. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
2. Word2Vec | TensorFlow Core
3. NLP 101: Word2Vec — Skip-gram and CBOW | by Ria Kulshrestha | Towards Data Science
4. NLP — Word Embedding & GloVe. BERT is a major milestone in creating… | by Jonathan Hui | Medium
5. Too powerful NLP model (GPT-2). What is Generative Pre-Training | by Edward Ma | Towards Data Science
6. Kaggle Twitter Sentiment Analysis: NLP & Text Analytics
7. Overview of Classification Methods in Python with Scikit-Learn
8. Feature selection using principal component analysis
9. An Introduction to Feature Selection | By Jason Brownlee
10. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
11. https://machinelearningmastery.com/combine-oversampling-and-undersampling-for-imbalanced-classification/
12. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
13. https://towardsdatascience.com/understanding-decision-tree-classifier-7366224e033b
14.

Training Dataset

1. https://www.kaggle.com/arkhoshghalb/twitter-sentiment-analysis-hatred-speech