ILLINOIS INSTITUTE OF TECHNOLOGY

10 W 35th St, Chicago, IL 60616

Multi Class Multi Label Classification

Project Members

Ajith Kumar Vakkalaganti Sunil Kumar (A20446704)

Manasdeep Deb (A20449643)

Shivani Rana (A20437697)

Under the guidance of

Professor Ming Long Lam

DEPARTMENT OF COMPUTER SCIENCE

Table of Contents

1 Project Name 3

1.1 Location of Document 3

2 Team Composition 3

2.1 Sponsor - CCC INFORMATION SERVICES 3

2.2 Illinois Tech 3

3 Project Background 4

3.1 Sponsor's Line of Business 4

3.2 Description of Sponsor's Problem 4

4 Project Objectives and/or Goals 4

4.1 Delivered Objectives 4

5 Project Deliverables 5

5.1 Description of Deliverables 5

5.2 Availability of Deliverables 5

6 Project Complications and Constraints 5

6.1 Complications 5

6.2 Constraint 5

7 Project Execution 6

- 7.1.1 Strategy - Open Set Classifier (Extreme Value Machine) 6
- 7.1.2 Agile Development- Open Set Classifier (Extreme Value Machine) 6
- 7.1.3 Findings and Results - Open Set Classifier (Extreme Value Machine) 14
- 7.2.1 Strategy - Ensemble Algorithm 19
- 7.2.2 Agile Development - Ensemble Algorithm 19
- 7.2.3 Findings and Results - Ensemble Algorithm 27
- 7.3.1 Strategy - Deep Learning Modelling 31
- 7.3.2 Agile Development - Deep Learning Modelling 31
- 7.3.3 Findings and Results - Deep Learning Modelling 47
- 7.4 Quality Assurance 53

8 Project Support 53

9 Project Retrospective 53

10 References or Other Supporting Documents 54

11 Document Revision History 54

Project Report Document

1 Project Name

Multi Class Multi Label Classification

1.1 Location of Document

https://drive.google.com/file/d/1cqzhepjkU8tequoH9EyyBVsJaLzIHM1w/view?usp=sharing

2 Team Composition

2.1 Sponsor - CCC INFORMATION SERVICES

- Person Of Contact 1: Manoochehr Assa (Email: MAssa@cccis.com)
- Person Of Contact 2: Husam Sweidan (Email: HSweidan@cccis.com)
- Person Of Contact 3: Mahboobeh Ghalehnovi (Email: MGhalehn@cccis.com)
- Advisor 1: Steven Penny (Email: SPenny@cccis.com)
- Advisor 2: Anneke Hidayat (Email: AHidayat@cccis.com)

2.2 Illinois Tech

- Advisor: Prof Ming Long Lam (Email: mlam5@iit.edu)
- Team Member 1: Ajith Kumar Vakkalaganti Sunil Kumar (Email: avakkalagantisunilku@hawk.iit.edu)
- Team Member 2: Manasdeep Deb (Email: mdeb@hawk.iit.edu)
- Team Member 3: Shivani Rana (Email: srana8@hawk.iit.edu)

3 Project Background

3.1 Sponsor's Line of Business

CCC Information Services, Inc. provides software products and services. The Company offers software solutions, analytical tools, and comprehensive data that focuses on integrated claims management, collision repair, and insurance estimating services. CCC Information Services serves clients in the United States. The specific department that this project will be a part of is the Research and Development unit.

3.2 Description of Sponsor's Problem

Achieving high precision on all classes(majority and minority classes), overcoming high class imbalance & being able to detect unknown classes from the dataset which is highly sparse in a Multi Label Multi Class classification is significantly challenging & hard to achieve. The CCC team wanted us to work with 2 Kaggle Datasets which closely represented their own data (in that they are sparse, multi-label classification problems). We were asked to work on two datasets namely Reuters News Multilabel Classification and Mechanism of Action (MOA): Pharmaceutical Drug Reactions Dataset which closely represented the company's data (highly sparse multi label multi class dataset).

4 Project Objectives and/or Goals

4.1 Delivered Objectives

We have delivered the following objectives

- Overcame class imbalance problem and achieved high precision score for each class
- Performed Open Set Classification with high precision
- Analysis of precision-recall values using deep learning for multi-label sparse target set prediction.

5 Project Deliverables

5.1 Description of Deliverables

The Deliverables comprise an algorithm or model that runs in the backend and supports a larger project while making predictions on classification. The Research and Development team does not require a User Interface as the final product but want a model that produces better precision and recall values.

5.2 Availability of Deliverables

All our code detailing the implementation of our approaches and algorithms is available to CCC in the form of a github repository

6 Project Complications and Constraints

6.1 Complications

- Propagation of class imbalance in label powerset transformation techniques
- RNN model predicting for only most popular class
- High time of execution and/or crashing od compiling platform

6.2 Constraint

- Minimal information is shared about the models that give out predictions.
- The models that give out predictions are not subject to any changes.
- Public datasets closely resembling actual data used so the results may not fully apply to actual data.
- No information about actual data shared with the team.

7 Project Execution

○ 7.1.1 Strategy - Open Set Classifier (Extreme Value Machine)

The EVM is derived from the statistical extreme value theory and is the first classifier that can perform kernel-free, nonlinear, variable bandwidth outlier detection combined with incremental learning. It has a compact probabilistic representation of each class's decision boundary, characterized in terms of its extreme vectors which provides a bound on open space risk. They bound class confidence to label regions with sufficient support from the training set belonging to that class.

○ 7.1.2 Agile Development- Open Set Classifier (Extreme Value Machine)

Here is my initiative towards Open Set Classification

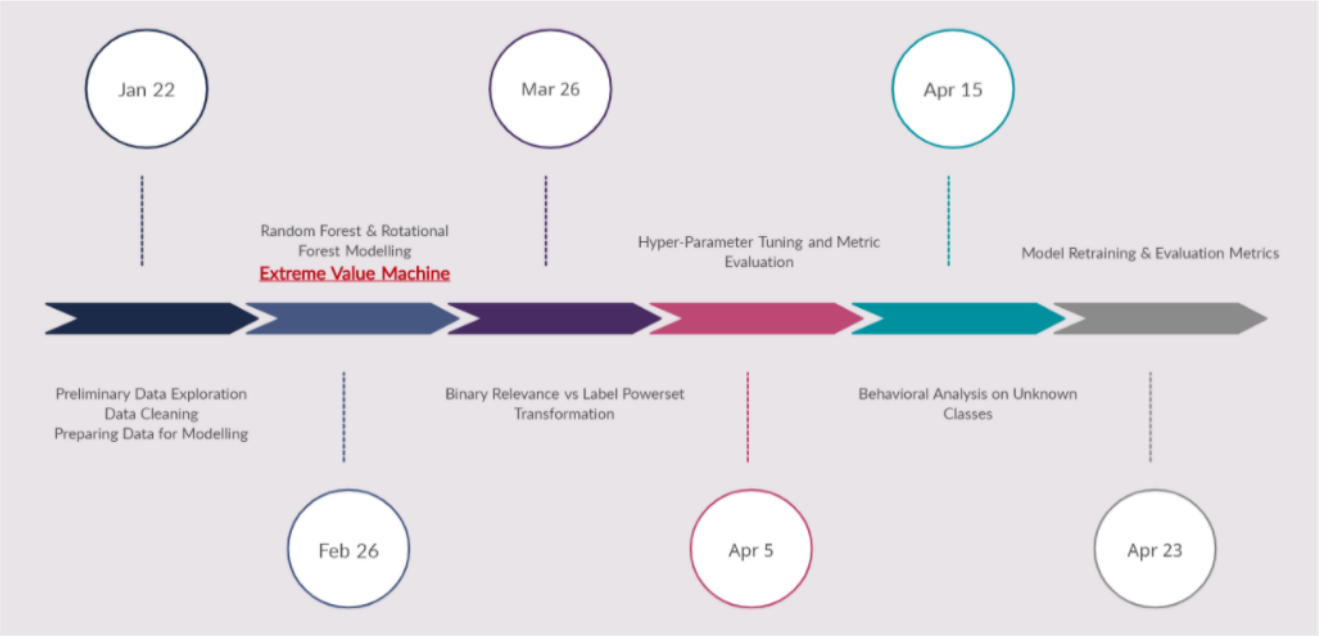


Fig: Open Set Classification Roadmap

Epic 1: Dataset Exploration & Data Extraction

Story 1: Targeting Sparse Datasets

- *Task 1: Open Source datasets*

We tried working with several datasets like Boston Housing, CIFAR-10, Fashion-MNIST and so on. Finally, the Reuters-21578 Text categorization collection dataset was finalized since it closely resembled the company's data. Reuters-21578 was sourced from UCI KDD Archive. The dataset consisted of 21 SGML format files with each file containing a maximum of 1000 documents from the Reuters financial newswire service. Here is an example of 1 document from the Reuters data

```
<!DOCTYPE lewis SYSTEM "lewis.dtd">
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="229" NEWID="12001">
<DATE> 1-APR-1987 11:04:07.01</DATE>
<TOPICS><D>earn</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>

&#5;&#5;&#5;F
&#22;&#22;&#1;f1288&#31;reute
s f BC-REPUBLIC-SAVINGS-AND   04-01 0039</UNKNOWN>
<TEXT>&#2;
<TITLE>REPUBLIC SAVINGS AND LOAN &lt;RSLA> SETS DIVIDEND</TITLE>
<DATELINE>    MILWAUKEE, Wis., April 1 -
    </DATELINE><BODY>Qtly div 30 cts vs 30 cts prior
    Pay April 27
    Record April 13
    NOTE: Company's full name is Republic Savings and Loan
Association of Wisconsin.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="230" NEWID="12002">
<DATE> 1-APR-1987 11:04:40.35</DATE>
<TOPICS><D>wheat</D><D>corn</D><D>soybean</D><D>grain</D><D>oilseed</D></TOPICS>
<PLACES><D>usa</D><D>ussr</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
```

Fig: SGML File Data

Story 2: Data Extraction

- Task 1: SGML Parsers & Data Transformation - The extracted dataset from SGML parsers had two columns, text and tags with 445 unique classes.

	text	tags
0	SANDOZ PLANS WEEDKILLER JOINT VENTURE IN USSR\...	[usa, ussr]
1	TAIWAN REJECTS TEXTILE MAKERS EXCHANGE RATE PL...	[usa, taiwan]
2	NATIONAL FSI INC <NFSI> 4TH QTR LOSS\n\nShr lo...	[earn, usa]
3	OCCIDENTAL <OXY> OFFICIAL RESIGNS\n\nMidCon Co...	[usa]
4	ITALY'S BNL TO ISSUE 120 MLN DLR CONVERTIBLE B...	[italy]
...	...	...
19711	ASHTON-TATE <TATE> TO OFFER COMMON SHARES\n\nA...	[usa]
19712	KEYCORP <KEY> REGISTERS SUBORDINATED NOTES\n\n...	[usa]
19713	<NATIONAL SEA PRODUCTS LTD> 4TH QTR NET\n\nOpe...	[earn, usa]
19714	U.K. MONEY MARKET SHORTAGE FORECAST REVISED DO...	[money-fx, interest, uk]
19715	NATIONAL AMUSEMENTS AGAIN UPS VIACOM <VIA> BID...	[acq, usa]

19716 rows × 2 columns

Fig: Extracted Dataset (2 columns)

- *Task 2: Dealing with Class Imbalance*

The dataset had a very high class imbalance due to which only the top 5 labels were considered in my analysis which were usa, earn, acq, uk and japan.

	unique_values	counts
0	usa	12542
1	earn	3987
2	acq	2448
3	uk	1489
4	japan	1138
5	canada	1104
6	money-fx	801
7	crude	634
8	grain	628
9	west-germany	567

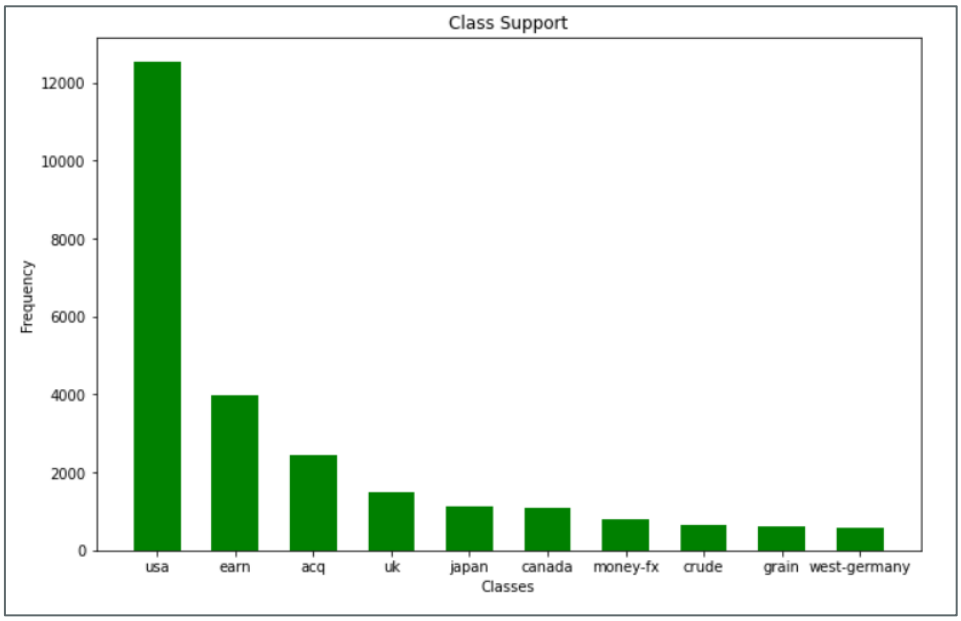
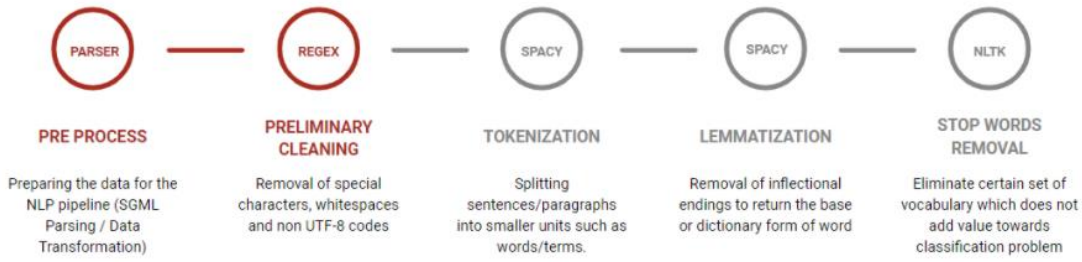


Fig: Class Imbalance

Epic 2: Data Cleaning & Natural Language Processing



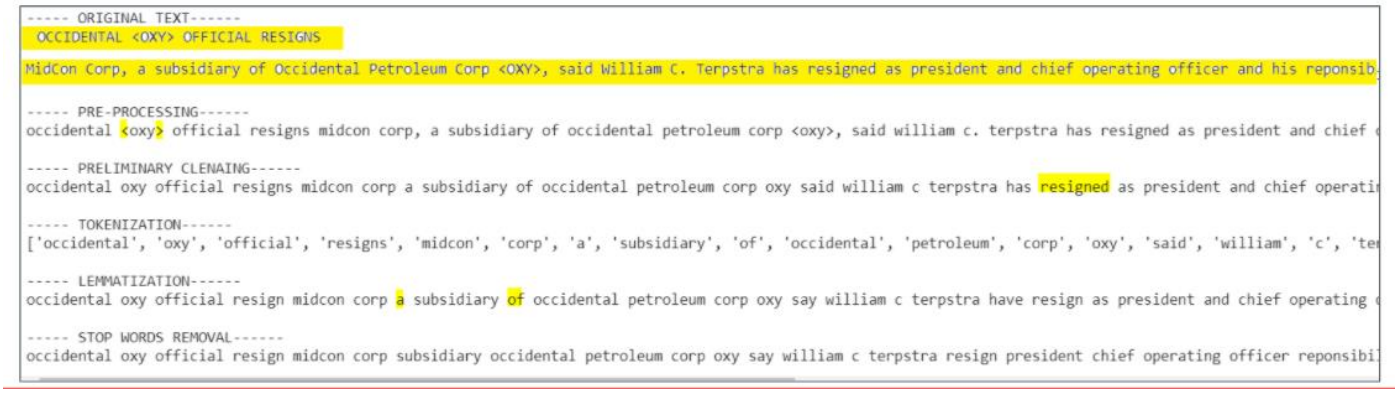


Fig: Data Cleaning & Natural Language Processing Pipeline

Story 1: Preliminary Data Analysis & PreProcessing

- Task 1: Fundamental data analysis

- Task 2: Data Cleaning -In the process of cleaning the data, all white spaces, non UTF-8 codes and special characters was removed

Story 2: Natural Language Processing

A highly efficient Natural Language Processing pipeline using NLTK and Spacy libraries was used to process the data for modelling.

- *Task 1: Tokenization - It basically refers to splitting of documents, paragraphs or sentences into smaller units called tokens*
- *Task 2: Lemmatization - It is a process of removing the inflectional endings of words to generate the base form or the dictionary form of the word*
- *Task 3: Stopwords Removal - Here, certain words called stop words which are not relevant towards classification are removed.*

Epic 3: Closed Set Classification vs Open Set Classification

A closed set classifier, SVM for instance works by building a decision boundary or hyperplane among the classes and in this case samples from an unknown class will be misclassified as a known class. For classifiers that assess confidence in terms of signed distance from a decision boundary, this misclassification will occur with high confidence if x — a result that is very misleading. An open set classifier on the other hand is trained to minimize both empirical risk with open space risk would bound class confidence to only label regions with sufficient support from the training set as belonging to a given class.

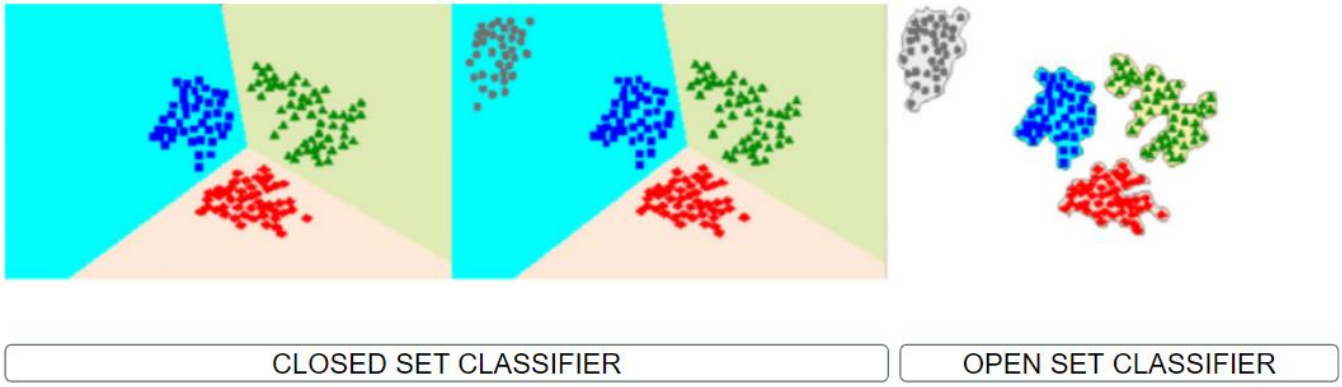


Fig: Closed Set Classification vs Open Set Classification

Story 1: Closed Set Classification Techniques

- *Task 1: Random Forest & Rotational Forest Modeling*

Story 2: Open Set Classification Techniques

- *Task 1: Extreme Value Machine (EVM) Modeling*

Epic 4: Extreme Value Machine (EVM) Modeling

The EVM is derived from the statistical extreme value theory and is the first classifier that can perform kernel-free, nonlinear, variable bandwidth outlier detection combined with incremental learning. It has compact probabilistic representation of each class's decision boundary, characterized in terms of its extreme vectors which provides a bound on open space risk

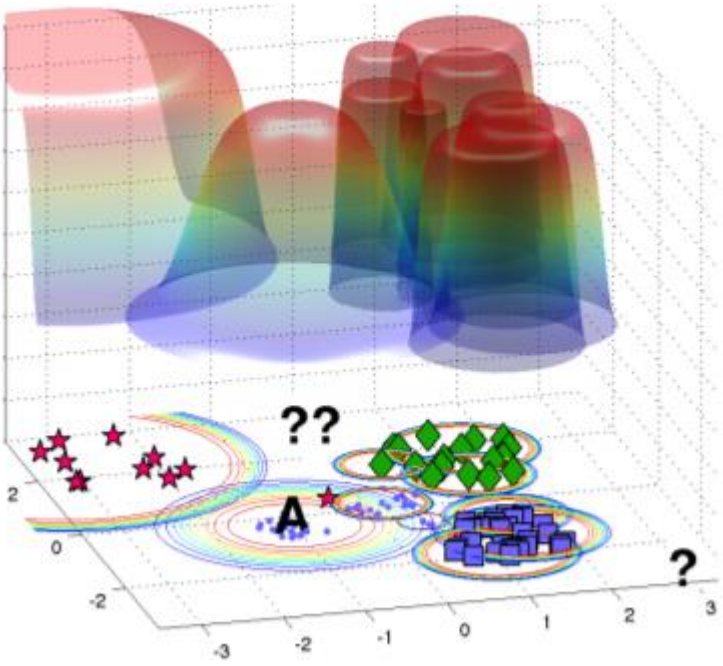


Fig: Probabilistic Representation Of Each Class in Extreme Value Machine (EVM)

<u>Story 1: Approaches</u>

- *Task 1: Label Powerset Transformation*

This approach transforms a multi-label problem to a multi-class problem with 1 multi-class classifier trained on all unique label combinations found in the training data.

```
unique_sets
```

```
{'acq': 6,
 'acqearn': 20,
 'acqearnusa': 18,
 'acqjapan': 19,
 'acqjapanuk': 22,
 'acqjapanukusa': 23,
 'acqjapanusa': 16,
 'acquk': 11,
 'acqukusa': 13,
 'acqusa': 3,
 'earn': 5,
 'earnjapan': 14,
 'earnjapanusa': 21,
 'earnuk': 12,
 'earnukusa': 17,
 'earnusa': 2,
 'japan': 4,
 'japanuk': 7,
 'japanukusa': 15,
 'japanusa': 9,
 'uk': 8,
 'ukusa': 10,
 'usa': 1}
```

Fig: Unique Label Combinations

- *Task 2: Binary Relevance*

This approach transforms a multi-label problem to a multi-class problem with 1 binary-class classifier trained on all labels found in the training data.

Story 2: Hyper parameter tuning

- *Task 1: Tuning model parameters*

Story 3: Optimum Threshold Evaluation

- *Task 1: Probability Thresholding*

Threshold: 0.001

	precision	recall	f1-score	support
0	0.41	0.48	0.44	491
1	0.00	0.00	0.00	247
2	0.69	0.83	0.75	778
3	0.31	0.52	0.39	250
4	0.00	0.00	0.00	164
5	0.40	0.64	0.49	298
6	0.78	0.61	0.69	2467
micro avg	0.62	0.58	0.60	4695
macro avg	0.37	0.44	0.39	4695
weighted avg	0.61	0.58	0.58	4695
samples avg	0.45	0.55	0.48	4695

****

Threshold: 0.005

	precision	recall	f1-score	support
0	0.49	0.38	0.43	491
1	0.00	0.00	0.00	247
2	0.80	0.78	0.79	778
3	0.37	0.38	0.37	250
4	0.00	0.00	0.00	164
5	0.52	0.51	0.52	298
6	0.81	0.50	0.62	2467
micro avg	0.71	0.49	0.58	4695
macro avg	0.43	0.37	0.39	4695
weighted avg	0.66	0.49	0.55	4695
samples avg	0.40	0.45	0.41	4695

Threshold: 0.01

	precision	recall	f1-score	support
0	0.54	0.35	0.43	491
1	0.00	0.00	0.00	247
2	0.83	0.76	0.79	778
3	0.42	0.33	0.37	250
4	0.00	0.00	0.00	164
5	0.56	0.46	0.50	298
6	0.81	0.46	0.59	2467
micro avg	0.74	0.45	0.56	4695
macro avg	0.45	0.34	0.38	4695
weighted avg	0.68	0.45	0.54	4695
samples avg	0.38	0.42	0.39	4695

****

Threshold: 0.05

	precision	recall	f1-score	support
0	0.68	0.27	0.39	491
1	0.00	0.00	0.00	247
2	0.93	0.70	0.80	778
3	0.49	0.22	0.31	250
4	0.00	0.00	0.00	164
5	0.70	0.34	0.46	298
6	0.85	0.35	0.50	2467
micro avg	0.83	0.36	0.50	4695
macro avg	0.52	0.27	0.35	4695
weighted avg	0.74	0.36	0.48	4695
samples avg	0.32	0.32	0.31	4695

Fig: Classification reports of all classes for threshold =0.001, 0.005, 0.01 and 0.05

Threshold: 0.1

	precision	recall	f1-score	support
0	0.73	0.24	0.36	491
1	0.00	0.00	0.00	247
2	0.95	0.66	0.78	778
3	0.56	0.18	0.27	250
4	0.00	0.00	0.00	164
5	0.73	0.30	0.42	298
6	0.86	0.30	0.45	2467
micro avg	0.85	0.32	0.47	4695
macro avg	0.55	0.24	0.33	4695
weighted avg	0.76	0.32	0.44	4695
samples avg	0.30	0.28	0.28	4695

****

Threshold: 0.15

	precision	recall	f1-score	support
0	0.74	0.22	0.33	491
1	0.00	0.00	0.00	247
2	0.97	0.62	0.76	778
3	0.63	0.16	0.26	250
4	0.00	0.00	0.00	164
5	0.78	0.29	0.42	298
6	0.87	0.27	0.41	2467
micro avg	0.87	0.30	0.44	4695
macro avg	0.57	0.22	0.31	4695
weighted avg	0.78	0.30	0.42	4695
samples avg	0.28	0.26	0.26	4695

Threshold: 0.2

	precision	recall	f1-score	support
0	0.76	0.20	0.32	491
1	0.00	0.00	0.00	247
2	0.97	0.61	0.75	778
3	0.69	0.15	0.24	250
4	0.00	0.00	0.00	164
5	0.82	0.26	0.39	298
6	0.88	0.25	0.39	2467
micro avg	0.89	0.28	0.42	4695
macro avg	0.59	0.21	0.30	4695
weighted avg	0.79	0.28	0.40	4695
samples avg	0.26	0.24	0.25	4695

****

Threshold: 0.45

	precision	recall	f1-score	support
0	0.85	0.16	0.27	491
1	0.00	0.00	0.00	247
2	0.99	0.53	0.69	778
3	0.72	0.11	0.19	250
4	0.00	0.00	0.00	164
5	0.83	0.18	0.30	298
6	0.88	0.17	0.28	2467
micro avg	0.91	0.21	0.34	4695
macro avg	0.61	0.16	0.25	4695
weighted avg	0.81	0.21	0.32	4695
samples avg	0.22	0.19	0.20	4695

Fig: Classification reports of all classes for threshold =0.1, 0.15, 0.2 and 0.45

Threshold: 0.5

	precision	recall	f1-score	support
0	0.85	0.15	0.26	491
1	0.00	0.00	0.00	247
2	0.99	0.51	0.67	778
3	0.74	0.11	0.19	250
4	0.00	0.00	0.00	164
5	0.85	0.17	0.29	298
6	0.88	0.16	0.27	2467
micro avg	0.91	0.20	0.33	4695
macro avg	0.62	0.16	0.24	4695
weighted avg	0.81	0.20	0.31	4695
samples avg	0.22	0.18	0.19	4695

****

Threshold: 0.55

	precision	recall	f1-score	support
0	0.89	0.15	0.25	491
1	0.00	0.00	0.00	247
2	0.99	0.50	0.67	778
3	0.77	0.11	0.19	250
4	0.00	0.00	0.00	164
5	0.87	0.17	0.29	298
6	0.89	0.15	0.25	2467
micro avg	0.92	0.19	0.32	4695
macro avg	0.63	0.15	0.24	4695
weighted avg	0.82	0.19	0.30	4695
samples avg	0.21	0.17	0.18	4695

Threshold: 0.7

	precision	recall	f1-score	support
0	0.89	0.13	0.23	491
1	0.00	0.00	0.00	247
2	0.99	0.47	0.64	778
3	0.92	0.10	0.17	250
4	0.00	0.00	0.00	164
5	0.86	0.14	0.25	298
6	0.90	0.12	0.21	2467
micro avg	0.94	0.17	0.28	4695
macro avg	0.65	0.14	0.21	4695
weighted avg	0.83	0.17	0.26	4695
samples avg	0.20	0.15	0.16	4695

Fig: Classification reports of all classes for threshold =0.5, 0.55 and 0.7

Story 4: Behavioral Analysis on Unknown Classes

- *Task 1: Building K Dimensional Probability Space*

The prediction probabilities of 5 binary models on unknown classes are used to build 5 dimensional probability space for clustering and retraining purposes

	probabilityc1	probabilityc2	probabilityc3	probabilityc4	probabilityc5
0	0.715453	0.000349	0.000162	0.001529	0.999999
1	0.546686	1.000000	0.001966	0.006069	0.007426
2	0.729032	0.000454	0.004563	0.125171	0.021777
3	0.217031	0.000349	0.002045	0.018291	0.002034
4	0.002766	0.000349	0.033438	0.996164	0.043321
...	...	...	...	...	...
1498	0.650708	0.000349	0.016585	0.005590	0.007622
1499	0.007772	0.036540	0.001848	0.001543	0.000556
1500	0.495708	0.000349	0.000004	0.329688	0.999851
1501	0.658106	0.002917	0.049345	0.000644	0.000384
1502	0.134988	0.002845	0.533178	0.002351	0.001816

1503 rows × 5 columns

- *Task 2: Clustering (K-Means & DBSCAN)*

Story 5: Model Retraining & Evaluation

- *Task 1: Retraining models on unknown classes*
- *Task 2: Model Evaluation*

7.1.3 Findings and Results - Open Set Classifier (Extreme Value Machine)

The above processing of data and experiments with the models have lead to certain conclusions that would be explained in detail in this section

- Minimum document frequency parameter of TF-IDF vectorizer helps in better feature selection. It tries to eliminate almost all noise and retain features which contribute towards classification

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectoriser=TfidfVectorizer(min_df=15)
vectoriser.fit_transform(data["text"])
print(vectoriser.vocabulary_.keys())
tokens=list(vectoriser.vocabulary_.keys())
tokens.sort()
print(tokens)
def retain_words(text):
  return " ".join(filter(lambda x:x in tokens, text.split()))

train["text"]=train["text"].apply(lambda x:retain_words(x))
```

Fig: TF-IDF Vectorization

- The classification threshold need not be 0.5 when there is class imbalance and for a threshold of 0.7 we are able to achieve 0.85 and above precision on all minority and majority classes

Threshold: 0.5

	precision	recall	f1-score	support
0	0.85	0.15	0.26	491
1	0.00	0.00	0.00	247
2	0.99	0.51	0.67	778
3	0.74	0.11	0.19	250
4	0.00	0.00	0.00	164
5	0.85	0.17	0.29	298
6	0.88	0.16	0.27	2467
micro avg	0.91	0.20	0.33	4695
macro avg	0.62	0.16	0.24	4695
weighted avg	0.81	0.20	0.31	4695
samples avg	0.22	0.18	0.19	4695

****

Threshold: 0.55

	precision	recall	f1-score	support
0	0.89	0.15	0.25	491
1	0.00	0.00	0.00	247
2	0.99	0.50	0.67	778
3	0.77	0.11	0.19	250
4	0.00	0.00	0.00	164
5	0.87	0.17	0.29	298
6	0.89	0.15	0.25	2467
micro avg	0.92	0.19	0.32	4695
macro avg	0.63	0.15	0.24	4695
weighted avg	0.82	0.19	0.30	4695
samples avg	0.21	0.17	0.18	4695

Threshold: 0.7

	precision	recall	f1-score	support
0	0.89	0.13	0.23	491
1	0.00	0.00	0.00	247
2	0.99	0.47	0.64	778
3	0.92	0.10	0.17	250
4	0.00	0.00	0.00	164
5	0.86	0.14	0.25	298
6	0.90	0.12	0.21	2467
micro avg	0.94	0.17	0.28	4695
macro avg	0.65	0.14	0.21	4695
weighted avg	0.83	0.17	0.26	4695
samples avg	0.20	0.15	0.16	4695

Fig: Classification reports of all classes for threshold =0.5, 0.55 and 0.7

- DBSCAN outperformed K-Means Clustering in performing clustering of N-Dimensional probability space. DBSCAN was able to cluster two major groups which was used for retraining purposes.

```
from sklearn.cluster import DBSCAN
import collections, numpy
db=DBSCAN(eps=0.1,min_samples=25, algorithm='ball_tree',
          metric='minkowski', leaf_size=90, p=2)
model=db.fit(df)
label=model.labels_
collections.Counter(label)
```

```
Counter({-1: 311,
         0: 605,
         1: 192,
         2: 42,
         3: 115,
         4: 37,
         5: 64,
         6: 25,
         7: 54,
         8: 48,
         9: 25})
```

Fig: DBSCAN Clustering Algorithm

```
tpcounter,fpcounter=0,0
for index, row in train_unknown_v2.iterrows():
  if(row["actual"]==0 and row["predicted"]==0):
    tpcounter=tpcounter+1
  if(row["actual"]!=0 and row["predicted"]==0):
    fpcounter=fpcounter+1

tpcounter/(tpcounter+fpcounter)
```

```
0.6049586776859505
```

```
[112] tpcounter,fpcounter=0,0
for index, row in train_unknown_v2.iterrows():
  if(row["actual"]==1 and row["predicted"]==1):
    tpcounter=tpcounter+1
  if(row["actual"]!=1 and row["predicted"]==1):
    fpcounter=fpcounter+1

tpcounter/(tpcounter+fpcounter)
```

```
0.5729166666666666
```

Fig: Precision on Train after performing DBSCAN

- Extreme Value Machine (EVM) performs really good when it comes to open set classification problems.

```
threshold=0.3
test_unknown_v2['predicted'] = [0 for _ in range(len(test_unknown_v2))]
for index, row in test_unknown_v2.iterrows():
    probabilitiesc6, indexesc6 =evm6.max_probabilities([tfidf.transform([row["text"]]).toarray()])
    if(probabilitiesc6[0]>threshold):
      test_unknown_v2.loc[index,"predicted"]=0
    else:
      test_unknown_v2.loc[index,"predicted"]=-1

tpcounter,fpcounter=0,0
for index, row in test_unknown_v2.iterrows():
  if(row["actual"]==0 and row["predicted"]==0):
    tpcounter=tpcounter+1
  if(row["actual"]!=0 and row["predicted"]==0):
    fpcounter=fpcounter+1

tpcounter/(tpcounter+fpcounter)
```

```
0.6363636363636364
```

Fig: Precision on “canada” unknown class

```
[145] threshold=0.05
      test_unknown_v2['predicted'] = [0 for _ in range(len(test_unknown_v2))]
      for index, row in test_unknown_v2.iterrows():
          probabilitiesc7, indexesc7 =evm7.max_probabilities([tfidf.transform([row["text"]]).toarray()])
          if(probabilitiesc7[0]>threshold):
            test_unknown_v2.loc[index,"predicted"]=1
          else:
            test_unknown_v2.loc[index,"predicted"]=-1

      tpcounter,fpcounter=0,0
      for index, row in test_unknown_v2.iterrows():
        if(row["actual"]==1 and row["predicted"]==1):
          tpcounter=tpcounter+1
        if(row["actual"]!=1 and row["predicted"]==1):
          fpcounter=fpcounter+1

      tpcounter/(tpcounter+fpcounter)

      0.45454545454545453
```

Fig: Precision on “money-fx” unknown class

- EVM is best suited for outlier detection & incremental learning problems

Model Statistics

TEXT PROCESSING	30m 56s
TRAINED CLASSES UNKNOWN CLASSES	["usa","earn","acq","uk","japan"] ["canada","money-fx"]
MODEL TRAINING	75m 31s - 10m
OPTIMAL THRESHOLD EVALUATION	66m
TEST SET EVALUATION	8m
RETRAINING TIME	8m - 5m

MODEL SIZE	413 Mb - 38 Mb
ENVIRONMENT	GOOGLE COLAB

○ 7.2.1 Strategy - Ensemble Algorithm

The strategy adopted was to use Reuters Dataset and experiment with various classification algorithms and choose the best among them in terms of performance metrics and speed to create a stacking ensemble that could improve the overall prediction for each label. The next step was to create an RNN model to process the same data and generate multi label predictions and compare the results in terms of support for class imbalance. Further hypertune both the models to check for any improvement in results

○ 7.2.2 Agile Development - Ensemble Algorithm

Epic:

This is the roadmap followed for Ensemble Algorithm

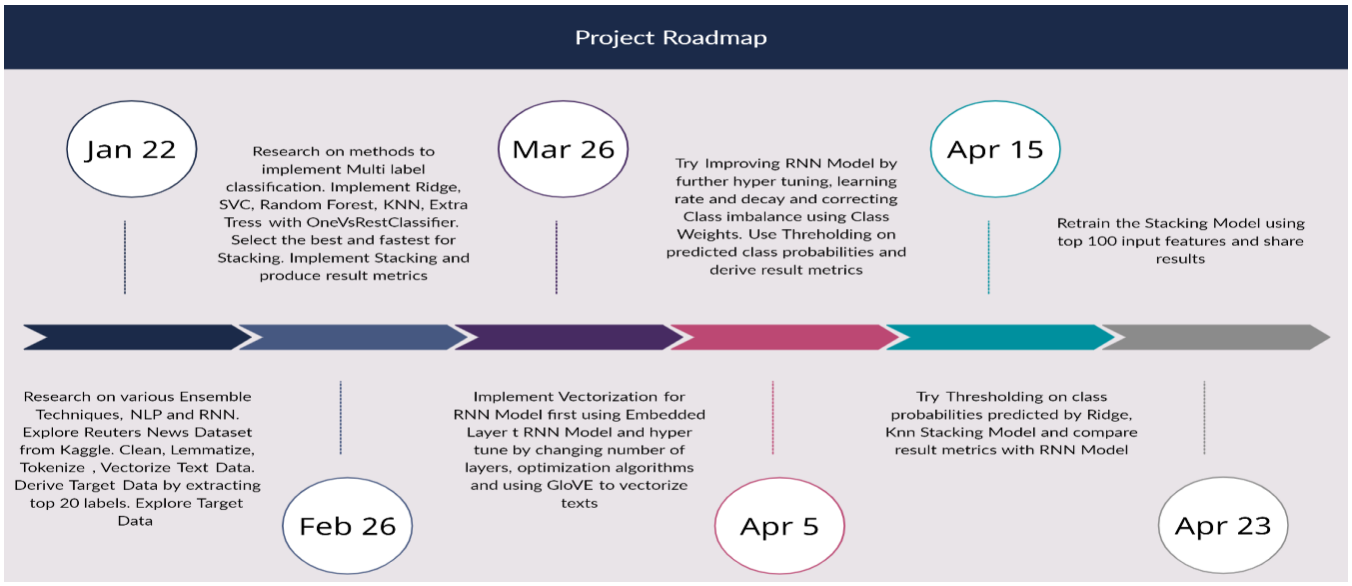


Fig: Roadmap for Ensemble Model Approach

Story 1: Research and Data Preparation

Researched on various Ensemble Techniques and best practices related to processing textual data and then implemented them to process data. Also, performed preliminary exploratory data analysis

- *Task 1: Reading*

Gained Preliminary Knowledge on various ensemble techniques like Stacking, Bagging, Boosting and Blending etc as the goal of this approach was to combine the results of multiple models rather than selecting the best out of them. Studied at length about RNN, NLP and regex techniques to get best results on textual data. The reading material used for this task were some technical articles and blogs on various platforms

- *Task 2: Data Acquisition and Preparation*

The Reuters News Dataset was acquired from Kaggle with the below columns

newID	oldID	title	author	data_split	isTopicsAvailable	document_date	dateline	people	places	orgs	exchanges	topic_body	topics
1	5544	BAHIA COCOA REVIEW	NaN	TRAIN	YES	26-FEB-1987 15:01:01.79	SALVADOR, Feb 26 -	[]	['el-salvador', 'usa', 'uruguay']	[]	[]	Showers continued throughout the week in\nthe ...	['cocoa']
2	5545	STANDARD OIL <SRD> TO FORM FINANCIAL UNIT	NaN	TRAIN	NO	26-FEB-1987 15:02:20.00	CLEVELAND, Feb 26 -	[]	['usa']	[]	[]	Standard Oil Co and BP North America\nInc said...	NaN
3	5546	TEXAS COMMERCE BANCSHARES <TCB> FILES PLAN	NaN	TRAIN	NO	26-FEB-1987 15:03:27.51	HOUSTON, Feb 26 -	[]	['usa']	[]	[]	Texas Commerce Bancshares Inc's Texas\nCommerc...	NaN
4	5547	TALKING POINT/BANKAMERICA <BAC> EQUITY OFFER	by Janie Gabbett, Reuters	TRAIN	NO	26-FEB-1987 15:07:13.72	LOS ANGELES, Feb 26 -	[]	['usa', 'brazil']	[]	[]	BankAmerica Corp is not under\npressure to act...	NaN
5	5548	NATIONAL AVERAGE PRICES FOR FARMER-OWNED RESERVE	NaN	TRAIN	YES	26-FEB-1987 15:10:44.60	WASHINGTON, Feb 26 -	[]	['usa']	[]	[]	The U.S. Agriculture Department\nreported the ...	['grain', 'wheat', 'corn', 'barley', 'oat', 's...

Fig: Reuters News Dataset (Kaggle)

The 5 columns that represent target categories are ‘people’ , ‘place’, ‘orgs’, ‘exchanges’ and ‘topics’. A new column ‘input’ was created by joining columns ‘title’ and ‘topic_body’. The top 20 labels were created by selecting top 4 labels from each of the 5 categories. These 20 labels were then added as 20 target label columns to the dataset and if the label was present in the column ‘input’ then that label column was set as 1 or else 0.

```
print(y_train)
```

```
       usa   uk  canada  japan  reagan  ...  trade  nyse  nasdaq  amex  tse
0      1.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
1      1.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
2      1.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
3      1.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
4      1.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
...    ...  ...     ...    ...     ...  ...    ...   ...     ...   ...  ...
14817  0.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
14818  0.0  0.0     0.0    1.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
14819  1.0  0.0     0.0    1.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
14822  0.0  0.0     0.0    1.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
17191  0.0  0.0     0.0    0.0     0.0  ...    0.0   0.0     0.0   0.0  0.0
```

Fig: 20 Target Label Columns

- *Task 3: Data Cleaning and Text Vectorization*

The ‘input’ column was cleaned using libraries ‘nltk’, ‘re’, ‘punkt’ and ‘wordnet’. The cleaning steps were as below

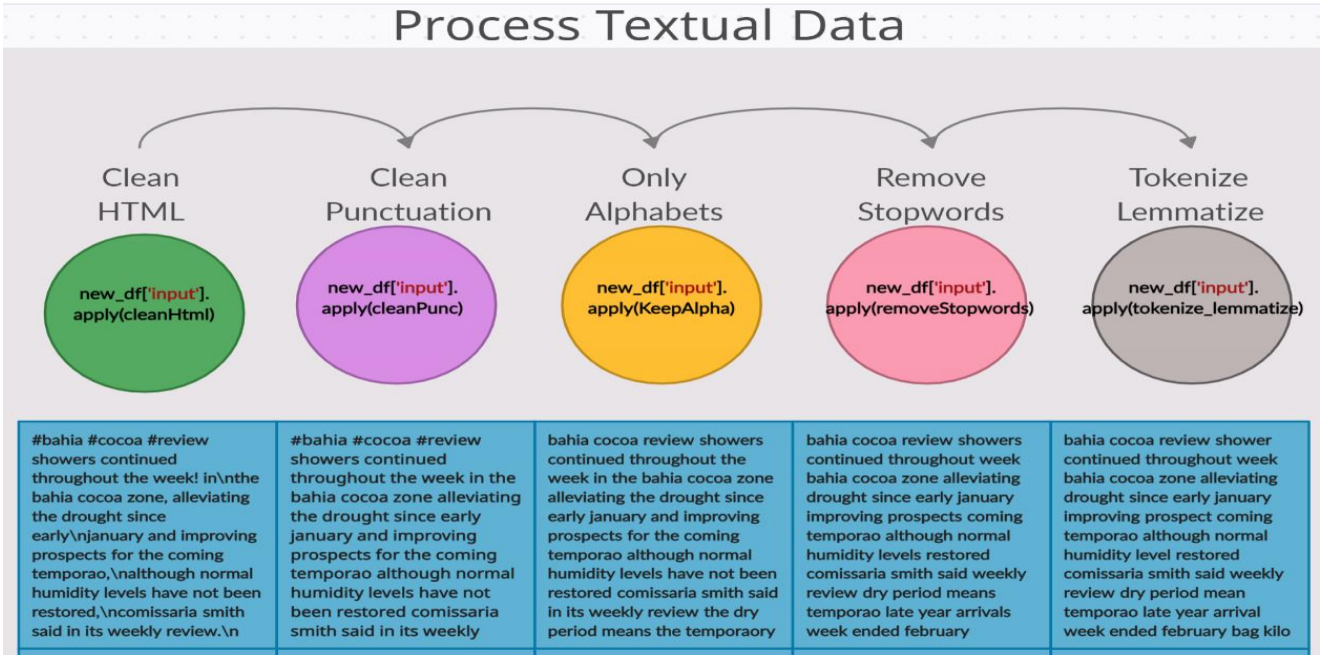


Fig: Data Cleaning Pipeline

The cleaning of textual data started with removing html tags, punctuation, special characters, stop words and then culminates with tokenization and lemmatization of the words. This cleaned data was then vectorized using tf-idf vectorization

- *Task 4: Exploratory Data Analysis*

The target data is further analysed to notice the trends in data. The below graph shows the number of labels for articles where it is observed about 9331 articles have only 1 label while 7 articles have 6 labels which is also the maximum number of labels present for any article

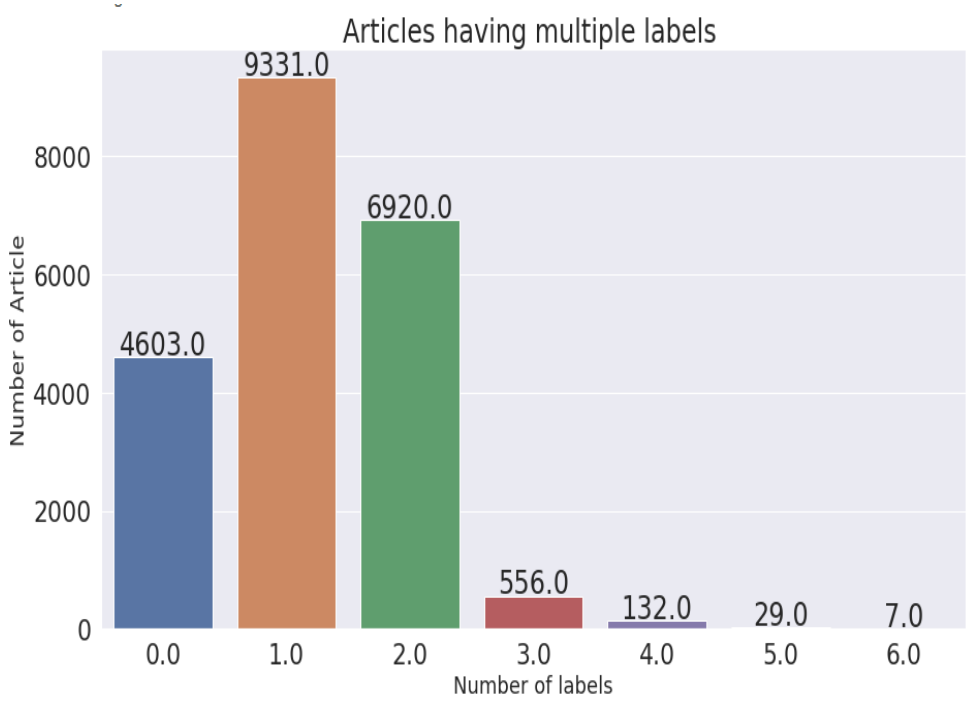


Fig: Number of Labels Vs Number of Articles

Next step was to check correlation between all the labels to make sure there is no high correlation present between any of the labels which may overfit the data. The correlation is very low with highest being 0.3869

	Row	Column	Value
149	opec	crude	0.386984
133	ec	opec	0.360513
1	usa	uk	0.300914
135	ec	icocoffee	0.277817
2	usa	canada	0.250215

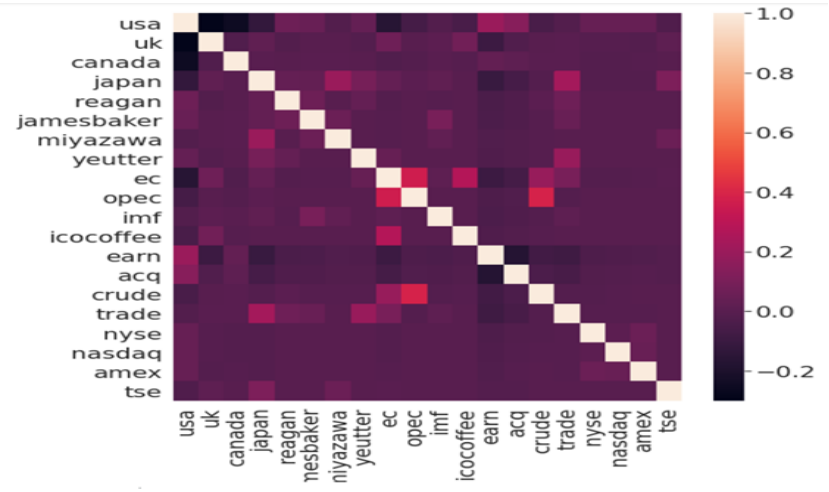


Fig: Correlation between Target Labels

Story 2: Stacking Model

The point of stacking is to explore a space of different models for the same problem. The idea is that you can attack a learning problem with different types of models which are capable of learning some part of the problem, but not the whole space of the problem. So, you can build multiple different learners and you use them to build an intermediate prediction, one prediction for each learned model. Then you add a new model which learns from the intermediate predictions the same target.The overall idea of stacking is to train several models, usually with different algorithm types (aka base-learners), on the train data, and then rather than picking the best model, all the models are aggregated/fronted using another model (meta learner), to make the final prediction. The inputs for the meta-learner are the prediction outputs of the base-learners.

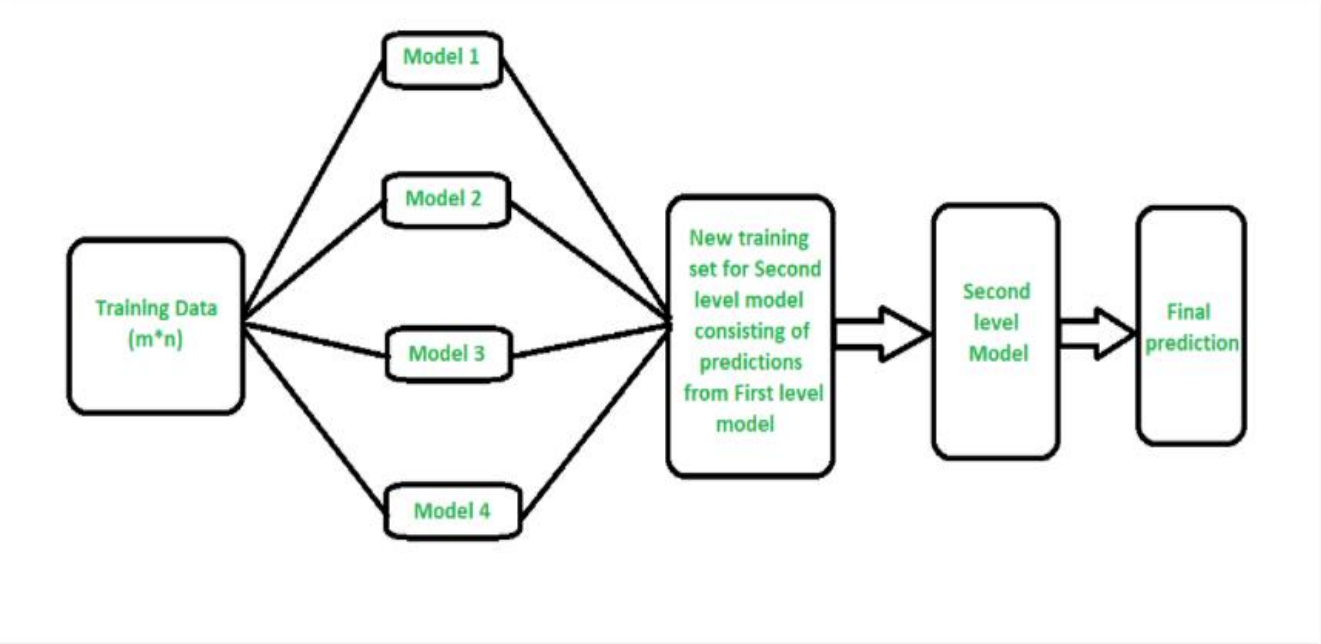


Fig: Stacking Model Workflow

- *Task 1: Select Algorithms for Stacking*

As Stacking combines the results of more than one heterogeneous models it is important to select the models that ensure an overall improvement in results rather than selecting poor performing models that do not really contribute to final results. For this reason, 6 different classification algorithms (Ridge, SVC, Logistic Regression, Extra Trees, Random Forest and KNN)were used to create models on the same data and compared based on speed of execution, tolerance to class imbalance and performance metrics like micro f1 score, precision, recall, roc_auc score etc. Since, these algorithms aon their own cannot predict multi-label results OneVsRestClassifier was used as a wrapper to perform multi-label classifications.

Classifier	Train Accuracy	Test Accuracy	roc_auc_score (Train)	roc_auc_score (Test)	Micro F1 Score (Test)	Micro Precision (Test)	Micro Recall (Test)
Ridge	0.98	0.73	0.97	0.77	0.87	0.89	0.86
SVC	0.97	0.7	0.94	0.69	0.85	0.89	0.81
Logistic	0.71	0.67	0.6	0.61	0.83	0.89	0.77
ExtraTrees	0.99	0.66	0.99	0.6	0.81	0.89	0.74
Random Forest	0.99	0.64	0.99	0.59	0.79	0.88	0.72
KNN	0.79	0.64	0.85	0.77	0.81	0.82	0.8

Fig: Comparison of Classifiers

Among all these models, KNN performed best in tolerating class imbalance closely followed by the Ridge Model. Ridge and KNN were also fastest in execution and had the highest roc_auc_score at 0.77. Ridge Model had the highest score in terms of micro f1 score, Precision and Recall. So, Ridge and KNN models were chosen for Stacking.

- *Task 2: Create Stacking Model*

The Stacking Model was created by using Ridge and KNN as base learners, i.e the models that predict the intermediate results. Logistic Regression was used as a meta learner to aggregate these predictions and perform final predictions on top of the two base learners. OneVsRestClassifier from sklearn.multiclass was again used to perform multi-label classification.

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.multiclass import OneVsRestClassifier

estimators_list = [('RidgeClassifier',RidgeClassifier()),('SVC', SVC())]

estimators_ensemble = StackingClassifier(estimators=estimators_list,final_estimator = LogisticRegression())

ovr_model = OneVsRestClassifier(estimators_ensemble)

ovr_model.fit(x_train, y_train)
#ovr_model.score(x_test, y_test)
train_pred = ovr_model.predict(x_train)
yhat = ovr_model.predict(x_test)
```

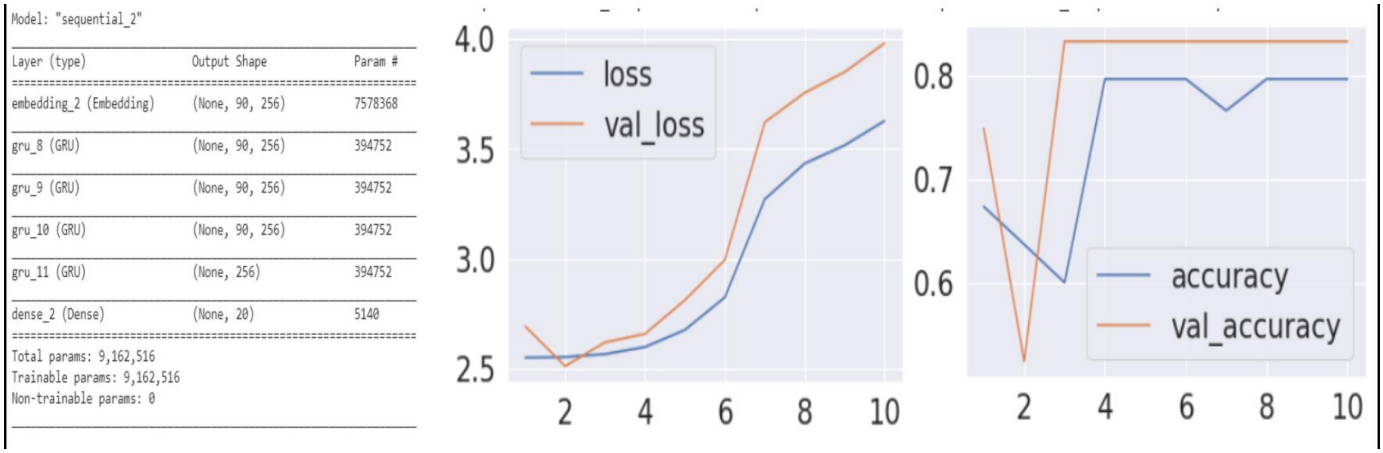
Fig: Ridge, KNN Stacking Model

Story 3: Recurrent Neural Network (RNN) Model

Recurrent Neural Networks are commonly used when we are dealing with sequential data. The reason is, the model uses layers that give the model a short-term memory. Using this memory, it can predict the next data more accurately. The time for which the information about the past data will be kept is not fixed, but it depends on the weights allotted to it. Thus, RNN is used in Sentiment Analysis, Sequence Labeling, Speech tagging, etc.

- *Task 1: RNN Model with Embedded Layers*

The first RNN model trained on text vectors using the Embedding layer in Keras and thus had 9,162,516 trainable parameters. Such a high number of trainable parameters combined with very small data to train on (about 21K rows) made it difficult for the model to learn and make good predictions. Thus the loss kept on increasing indefinitely and the accuracy shot to 80% at epoch 2 and stayed there proving inability of the model to reach an optimal solution and high susceptibility to class imbalance.



Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 90, 256)	7578368
gru_8 (GRU)	(None, 90, 256)	394752
gru_9 (GRU)	(None, 90, 256)	394752
gru_10 (GRU)	(None, 90, 256)	394752
gru_11 (GRU)	(None, 256)	394752
dense_2 (Dense)	(None, 20)	5140

Total params: 9,162,516
Trainable params: 9,162,516
Non-trainable params: 0

Fig: RNN Model with Embedded layer

- *Task 2: RNN Model with pre trained vectors (GloVe)*

In order to reduce the number of trainable parameters, the text vectors are pre trained on GloVe 50 dimension vectors and thus reduce the trainable parameters considerably and allowing the loss to decrease through each epoch

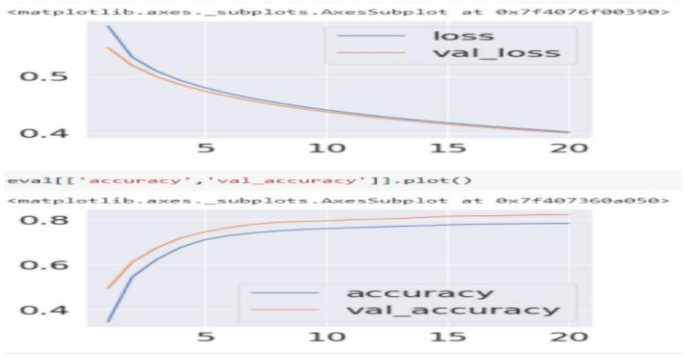


Fig: Loss and Accuracy after GloVe

- *Task 3: RNN Model with Class Weights*

The previous model shows a gradual decline in loss but the accuracy score more or less shows the same trend as in the previous model and the accuracy is still steeply rising to 80% and getting stuck there which shows a continued intolerance towards class imbalance. To rectify this shortcoming Class Weights were introduced to the RNN model. These Class Weights were based on inverse frequency of classes i.e class with highest frequency has lowest weight

Story 4: Fine tuning parameters

Some hyperparameter tuning was done through various iterations of RNN model with the hope of extracting better performance from the model. Probability thresholding was done to discover the threshold at which a certain performance metrics yields the highest value.

- *Task 1: Hypertune parameters in RNN Model*

After a lot of iterations the layers of RNN were limited to 2 layers. The learning rate was set to 0.001 and decay to 0.3 and epochs to 20. These changes were made in addition to pre-training text vectors using GloVe and adding class weights, activation function as sigmoid and loss function as binary cross entropy and optimizer as Adam.

```
model = Sequential()
model.add(Embedding(vocab_size+1,embedding_dim,input_length=maxLength, weights=[embeddings_matrix], trainable=False))
model.add(GRU(256,  return_sequences=True))
model.add(GRU(256))
model.add(Dense(20, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=keras.optimizers.Adam(lr=0.001,decay = 0.2), metrics=['accuracy'])
model.summary()
```

```
history = model.fit(totalX,
                    y_train,
                    batch_size=128,
                    epochs=20,
                    validation_data = (totalX_test,y_test),class_weight = weights)
```

Fig: RNN Model with Class Weights and hypertuned parameters

- *Task 2: Probability Thresholding*

RNN Model's predicted probabilities were passed through thresholds ranging from 0 to 0.95 with a 0.05 step. At each step of the threshold, the results were subjected to performance metrics like precision, recall, f1_score and auc_roc score. The thresholds at which each of these predictions reached highest values were noted and then used to make predictions which were further analyzed using classification reports. The same process was repeated for the Stacking Model to check for a probability threshold value that may result in a higher prediction score.

○ 7.2.3 Findings and Results - Ensemble Algorithm

The above processing of data and experiments with the models have lead to certain conclusions that would be explained in detail in this section

- *Stacking Model performs better than its base learners*

In the Stacking Model discussed here, Ridge and KNN are the base learners. It was observed that that the Ridge, KNN Stacking Model had a higher Micro F1 score across all 20 labels compared to both the base learners and also had very high roc_auc_scores

```
print("Test Accuracy = ",accuracy_score(y_test,yhat))
print("Train Accuracy = ",accuracy_score(y_train,train_pred))

Test Accuracy =  0.7409307438622206
Train Accuracy =  0.9787019043917606

from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,yhat,multi_class='ovr',average=None)

array([0.91400444, 0.86513211, 0.86827553, 0.93100956, 0.92940153,
       0.81688857, 0.79411765, 0.78562256, 0.88027276, 0.82096817,
       0.80244708, 0.99990827, 0.94810635, 0.94398337, 0.94333485,
       0.87850395, 0.77567787, 0.8        , 0.65909091, 0.5        ])

from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,yhat,multi_class='ovr',average='micro')

0.9453567912426941

from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,yhat,multi_class='ovr',average='macro')

0.8428372766483806

roc_auc_score(y_train,train_pred,multi_class='ovr')

0.9657189190806073
```

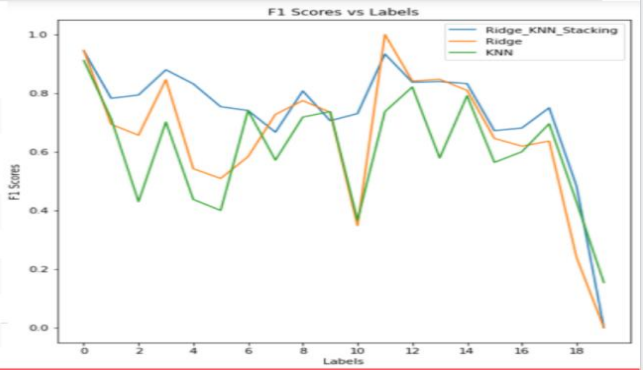


Fig: Stacking Model Results

A comparison between the Classification Reports for Ridge Model, KNN Model and Ridge, KNN Stacking Model shows a high tolerance for class imbalance in the stacking Model

Ridge Classifier

```
print(classification_report(y_test,yhatRidge))
```

	precision	recall	f1-score	support
0	0.95	0.94	0.95	3868
1	0.89	0.57	0.69	347
2	0.95	0.50	0.66	297
3	0.95	0.76	0.85	336
4	1.00	0.37	0.54	43
5	1.00	0.34	0.51	41
6	1.00	0.41	0.58	17
7	1.00	0.57	0.73	7
8	0.91	0.68	0.77	170
9	0.86	0.64	0.73	28
10	1.00	0.21	0.35	38
11	1.00	1.00	1.00	7
12	0.74	0.98	0.84	1045
13	0.87	0.83	0.85	643
14	0.88	0.75	0.81	161
15	0.66	0.63	0.65	112
16	1.00	0.45	0.62	29
17	1.00	0.47	0.64	15
18	1.00	0.14	0.24	22
19	0.00	0.00	0.00	12
micro avg	0.89	0.86	0.87	7238
macro avg	0.88	0.56	0.65	7238
weighted avg	0.90	0.86	0.87	7238
samples avg	0.77	0.77	0.75	7238

Knn Classifier

```
print(classification_report(y_test,yhatKnn))
```

	precision	recall	f1-score	support
0	0.90	0.92	0.91	3868
1	0.78	0.65	0.71	347
2	0.50	0.38	0.43	297
3	0.81	0.62	0.70	336
4	0.67	0.33	0.44	43
5	0.54	0.32	0.40	41
6	1.00	0.59	0.74	17
7	0.57	0.57	0.57	7
8	0.76	0.68	0.72	170
9	0.72	0.75	0.74	28
10	0.50	0.29	0.37	38
11	0.58	1.00	0.74	7
12	0.72	0.96	0.82	1045
13	0.75	0.47	0.58	643
14	0.82	0.76	0.79	161
15	0.56	0.57	0.56	112
16	0.71	0.52	0.60	29
17	1.00	0.53	0.70	15
18	1.00	0.27	0.43	22
19	1.00	0.08	0.15	12
micro avg	0.82	0.80	0.81	7238
macro avg	0.75	0.56	0.60	7238
weighted avg	0.82	0.80	0.80	7238
samples avg	0.73	0.73	0.71	7238

Ridge, Knn Stacking

```
print(classification_report(y_test,Ridge_Knn_Stack_test_pred))
```

	precision	recall	f1-score	support
0	0.96	0.93	0.94	3868
1	0.83	0.74	0.78	347
2	0.85	0.74	0.79	297
3	0.89	0.87	0.88	336
4	0.80	0.86	0.83	43
5	0.93	0.63	0.75	41
6	1.00	0.59	0.74	17
7	0.80	0.57	0.67	7
8	0.86	0.76	0.81	170
9	0.78	0.64	0.71	28
10	0.92	0.61	0.73	38
11	0.88	1.00	0.93	7
12	0.73	0.98	0.84	1045
13	0.77	0.93	0.84	643
14	0.78	0.89	0.83	161
15	0.60	0.77	0.67	112
16	0.89	0.55	0.68	29
17	1.00	0.60	0.75	15
18	1.00	0.32	0.48	22
19	0.00	0.00	0.00	12
micro avg	0.87	0.90	0.88	7238
macro avg	0.81	0.70	0.73	7238
weighted avg	0.88	0.90	0.88	7238
samples avg	0.77	0.80	0.77	7238

Fig: Classification Reports

Also, the micro average f1 score for Stacking is highest compared to its base learners at 88%. This score is very close to the 89.9% micro average f1 score achieved by the research paper titled '[MAGNET: Multi-Label Text Classification using Attention-based Graph Neural Network]' which explores multi-label text classification on the same Reuters News dataset. The speed and performance of the Stacking model is better than all the other models.

- *Adding Class Weights to RNN Model did not improve it's tolerance to class imbalance*

The RNN Models with and without class weights were compared over different probability thresholds and below were the graphs obtained

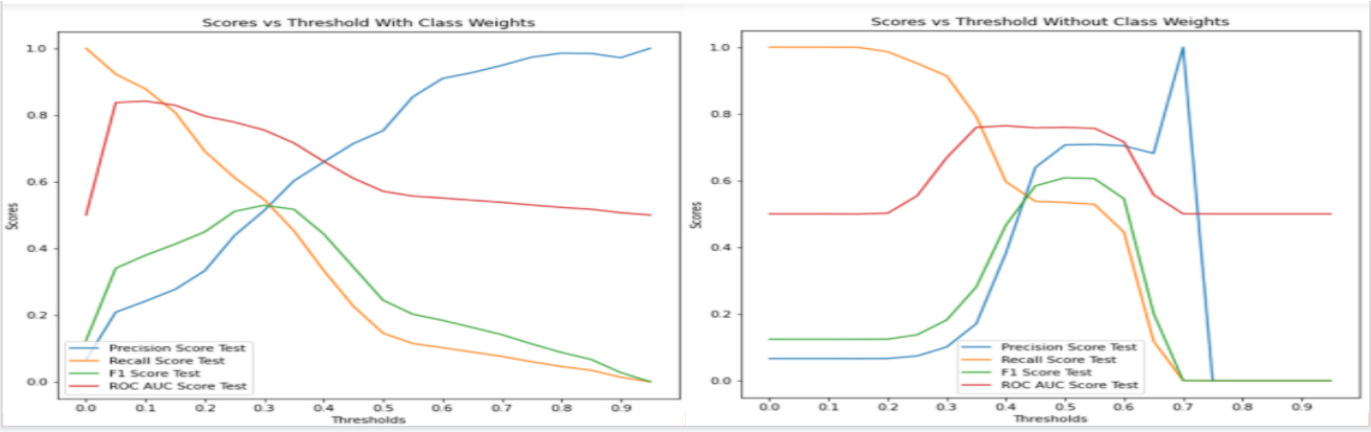


Fig: Probability Thresholding for RNN Model

The graph with class weights shows slightly higher values for performance metrics but it does not mean the class imbalance has been addressed which is obvious from the classification reports calculated for predictions at thresholds where Precision, Recall, F1 score or Roc_AUC scores were highest

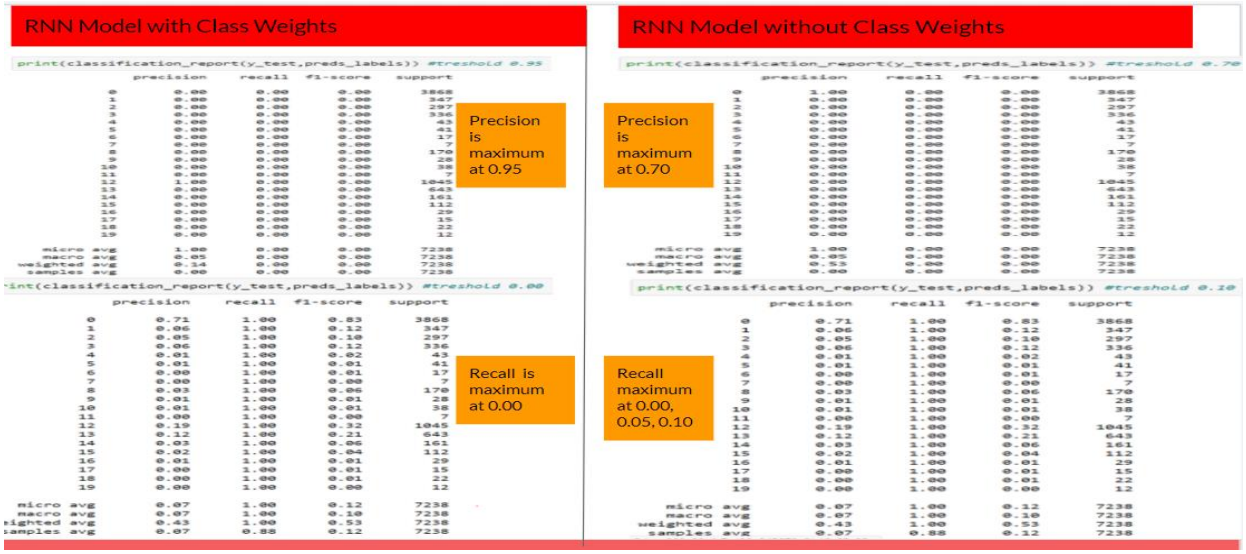


Fig: Classification Reports

There is no improvement in models at thresholds where highest Precision score was recorded, the model continued to predict for only 1 class albeit for a class with lower support even after adding class weights and no changes were improved at the threshold level with highest Recall Value

Fig: Classification Reports

For thresholds where F1 score or ROC_AUC scores are highest the model with class weights performs slightly better but the improvement is not significant enough to lead to any significant results. Thus adding class weight to the RNN model did not really improve its performance.

- *Stacking Model performs better than RNN for multi-label classification problem with high class imbalance*

The Stacking Model was put through probability thresholding in the same way as the RNN model and the results were as below

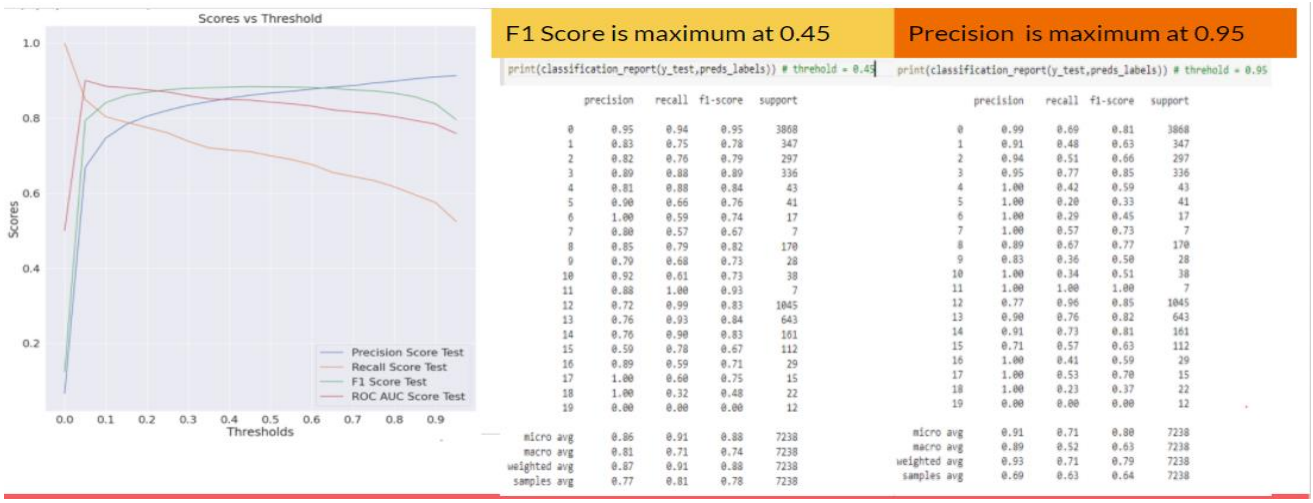


Fig: Probability Thresholding for Stacking Model

It is obvious from the classification report for Stacking that it deals with class imbalance in a much better way than the RNN model providing good predictions for 19 out of 20 labels. Also, as the threshold is increased to match the value where the highest precision score is achieved the precision still increases for all 19 classes as opposed to one class in RNN model. Also, the precision score in that case is 1 for 9 out of 20 classes and for most of the remaining classes it is at least 90% or above. Thus, clearly showing that Stacking outperformed RNN by a huge margin

Model Statistics

- Rows of Data - 21K
- Number of labels - 20
- Environment - Google Colab

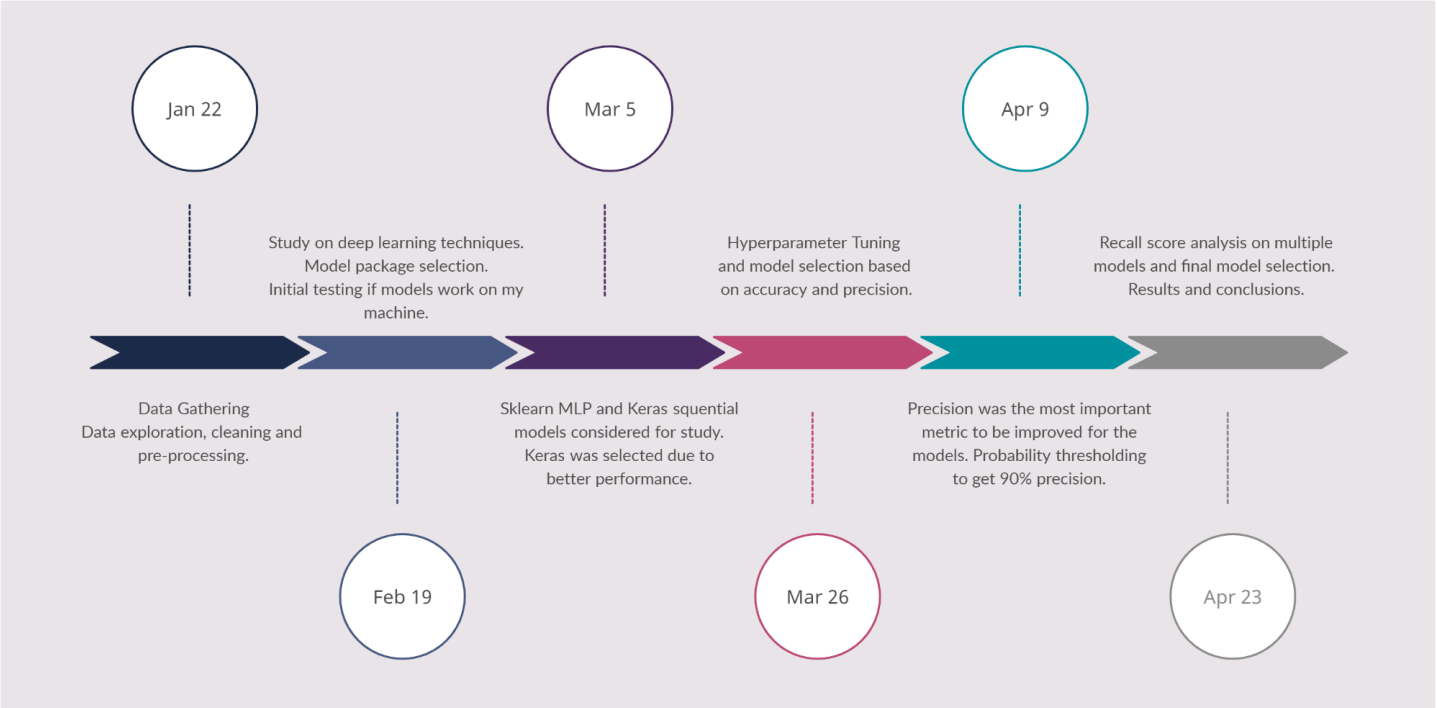
Model	Approximate Time for Training
Ridge	5 min
SVC	3 - 4 hours
Logistic	3 - 4 hours
Random Forest	5 - 7 hours
Extra Trees	5 - 7 hours
Knn	5 min
Ridge, KNN Stacking	15 min
RNN Model	30 min - 1 hour

7.3.1 Strategy - Deep Learning Modelling

This strategy was used to work on the Mechanism of Action dataset from Laboratory of Innovation Science Harvard which has a multi label binary target set. The Keras package in python was used to create Sequential models with fully connected Dense Layers. Hyperparameter tuning was done, multiple models were trained and tested using precision and recall scores. The overall scores and the individual scores on each label was studied. The goal was to achieve 90% precision and then study the recall values at that threshold.

7.3.2 Agile Development - Deep Learning Modelling

Project Roadmap:



Epic 1: Data Gathering and Exploration

Story 1: Data description

The **Mechanism of Action dataset** is from Laboratory of Innovation Science Harvard. The importance of working on this data is that scientists seek to identify a protein target (the target variables) associated with a disease and develop a molecule that can modulate that protein target. The dataset combines gene expression and cell viability data. The data is based on a new technology that measures simultaneously (within the same samples) human cells' responses to drugs in a pool of 100 different cell types (thus solving the problem of identifying ex-ante, which cell types are better suited for a given drug). This is a multi-label classification problem with binary labels.

Data points: 23814
Features: 872 (2 categorical, 770 Gene Expression, 100 Cell Viability)
Data types: Float/Integer
Targets/Labels: 206
Source: Kaggle[1]

We can see a sample of the feature vectors. There are two categorical ones (cp_time and cp_dose). The vectors starting with 'g-' represent the gene expression data and the ones starting with 'c-' represent the cell viability data.

	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6	g-7	...	c-90	c-91	c-92	c-93	c-94	c-95	c-96
0	24	D1	1.0620	0.5577	-0.2479	-0.6208	-0.1944	-1.0120	-1.0220	-0.0326	...	0.2862	0.2584	0.8076	0.5523	-0.1912	0.6584	-0.3981
1	72	D1	0.0743	0.4087	0.2991	0.0604	1.0190	0.5207	0.2341	0.3372	...	-0.4265	0.7543	0.4708	0.0230	0.2957	0.4899	0.1522
2	48	D1	0.6280	0.5817	1.5540	-0.0764	-0.0323	1.2390	0.1715	0.2155	...	-0.7250	-0.6297	0.6103	0.0223	-1.3240	-0.3174	-0.6417
3	48	D1	-0.5138	-0.2491	-0.2656	0.5288	4.0620	-0.8095	-1.9590	0.1792	...	-2.0990	-0.6441	-5.6300	-1.3780	-0.8632	-1.2880	-1.6210
4	72	D2	-0.3254	-0.4009	0.9700	0.6919	1.4180	-0.8244	-0.2800	-0.1498	...	0.0042	0.0048	0.6670	1.0690	0.5523	-0.3031	0.1094

5 rows × 874 columns

This is a sample of the targets which are protein substances.

	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylcholine_receptor_agonist	acetylcholine_receptor_antagonist	acetylcholinesterase_inhibitor
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0

5 rows × 206 columns

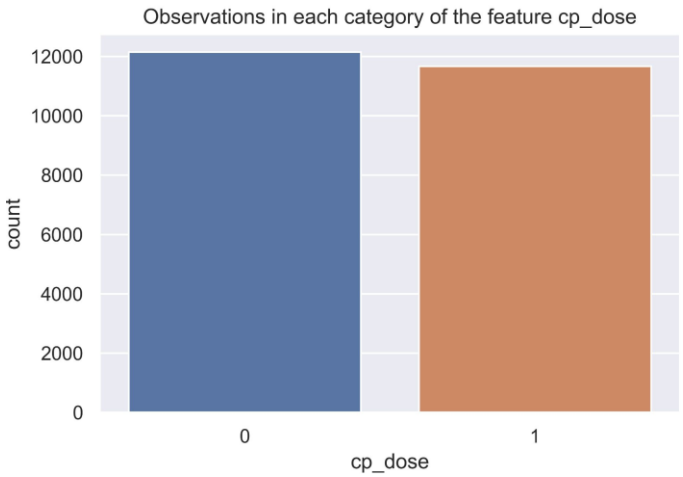
[1] https://www.kaggle.com/c/lish-moa/overview

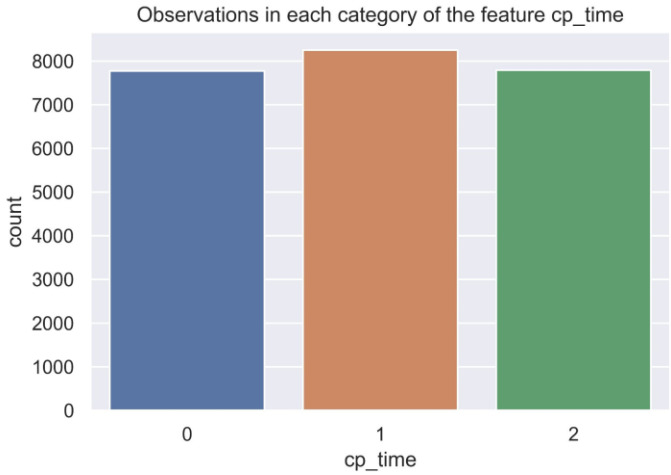
Story 2: Data Preprocessing

Initial inspection showed that the data was already clean and it did not have any missing values. The gene and cell data were in the form of simple float values. But before we move further two important transformations were carried out. Following are the few steps that were undertaken before we got into modelling:

- *Task 1: Encoding categorical data*

From the previous section we can see that the variable cp_time is in hours and cp_dose is in the form of strings. We encode them to be represented as class labels (0,1,2.. etc). Below displayed are the class distribution of the two variables.





Here is a sample of the data after the categorical variables were encoded:

	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6	g-7	...	c-90	c-91	c-92	c-93	c-94	c-95	c-96
0	0	0	1.0620	0.5577	-0.2479	-0.6208	-0.1944	-1.0120	-1.0220	-0.0326	...	0.2862	0.2584	0.8076	0.5523	-0.1912	0.6584	-0.3981
1	2	0	0.0743	0.4087	0.2991	0.0604	1.0190	0.5207	0.2341	0.3372	...	-0.4265	0.7543	0.4708	0.0230	0.2957	0.4899	0.1522
2	1	0	0.6280	0.5817	1.5540	-0.0764	-0.0323	1.2390	0.1715	0.2155	...	-0.7250	-0.6297	0.6103	0.0223	-1.3240	-0.3174	-0.6417
3	1	0	-0.5138	-0.2491	-0.2656	0.5288	4.0620	-0.8095	-1.9590	0.1792	...	-2.0990	-0.6441	-5.6300	-1.3780	-0.8632	-1.2880	-1.6210
4	2	1	-0.3254	-0.4009	0.9700	0.6919	1.4180	-0.8244	-0.2800	-0.1498	...	0.0042	0.0048	0.6670	1.0690	0.5523	-0.3031	0.1094

5 rows × 874 columns

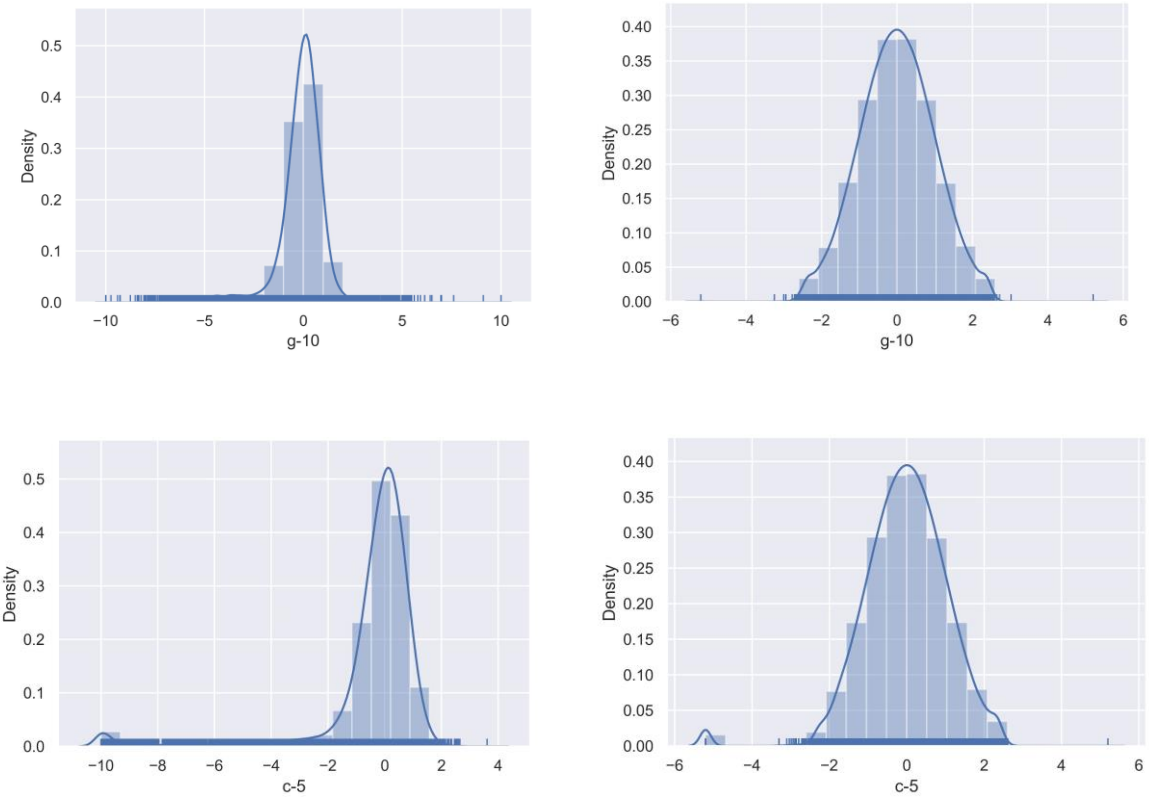
- *Task 2: Quantile transformation*[2]

Some input variables may have a highly skewed distribution, such as an exponential distribution where the most common observations are bunched together. Some input variables may have outliers that cause the distribution to be highly spread. These concerns and others, like non-standard distributions and multi-modal distributions, can make a dataset challenging to model with a range of machine learning models.

We know that most machine learning models tend to perform better when the input variables are in the form of Gaussian distribution. As such, it is often desirable to **transform each input variable to have a standard probability distribution, such as a Gaussian (normal) distribution or a uniform distribution.** A quantile transform will map a variable's probability distribution to another probability distribution.

The quantile function ranks or smooths out the relationship between observations and can be mapped onto other distributions, such as the uniform or normal distribution. The transformation can be applied to each numeric input variable in the training dataset and then provided as input to a machine learning model to learn a predictive modeling task.

Following are the examples of one gene and one cell vector before and after transformation.



[2] https://machinelearningmastery.com/quantile-transforms-for-machine-learning/

Story 3: Principal component analysis:

PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The principal component can be taken as a direction orthogonal to the first principal components that maximizes the variance of the projected data.

It can be shown that the principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix.

Why PCA for our data?

The number of features **(872)** seemed too high for the amount of data we had **(~23000)**. This might cause a problem of overfitting due to the curse of dimensionality. The feature space is broadly divided into two sections. Gene expression data and Cell viability data. Since they convey different information, I decided to do the PCA in two parts. The 770 dimensional Gene expression data was reduced to 50 and the 100 dimensional Cell data was reduced to 15. So, the feature space was reduced to 67 (65 numerical, 2 categorical).

Amount of variance explained by 50-dim Gene data = 70%

Amount of variance explained by 15-dim Cell data = 75%

Below is a sample of the data after the dimension was reduced to 67:

	cp_time	cp_dose	0	1	2	3	4	5	6	7	...	5	6	7	8
0	0	0	-5.525087	3.181043	9.515787	-8.024189	4.916372	1.281874	3.575350	1.399371	...	1.077778	-0.409324	-0.161712	-0.406047
1	2	0	-4.875594	4.867819	-10.794629	6.022212	0.812267	0.333364	1.042722	-0.477125	...	-0.776515	-0.855593	-0.042043	-1.153853
2	1	0	1.155351	-6.702401	-6.146352	-0.862000	0.750856	3.522865	-2.058024	2.791020	...	0.579971	0.811467	-0.304513	-0.659633
3	1	0	11.348175	-8.101228	-5.074172	-5.971018	-6.821274	-3.175198	-2.411201	6.821145	...	1.255841	-0.270368	-0.215754	-0.977991
4	2	1	-6.604919	-1.092044	-10.821327	-4.096393	-7.743386	-8.699825	-4.207239	-3.014785	...	-0.322702	0.259244	-0.668553	0.327941

5 rows × 67 columns

Epic 2: Model Package Selection

For the purpose of solving this problem we decided to use deep learning methods. Multi-label classification often requires deep neural models to show good performance. Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

We decided to do a preliminary study on **Sklearn MLP and Keras Sequential** for the purpose of selecting the package that we will be using for model building. Experiments were done using some basic hyperparameter tuning: learning rate, layer sizes and optimizers.

Below displayed are the best results and the hyperparameters after some initial experimentation of the models from two packages. Accuracy was used as the metric:

Story 1:Model - Keras Sequential

Training Set best performance:

Layers Used - 5 Dense Layers

Input and Hidden layer activation: ReLU

Output layer activation: Sigmoid

Optimizer: NAdam | Loss: Binary Cross-entropy | Learning Rate: 0.001

Epochs: 300 | Batch Size: 32

Performance: Accuracy 99.2% (Corresponding test set accuracy = 40%)

Test Set best performance:

Layers Used - 4 Dense Layers

Input and Hidden layer activation: ReLU

Output layer activation: Sigmoid

Optimizer: NAdam | Loss: Binary Cross-entropy | Learning Rate: 0.001

Epochs: 200 | Batch Size: 32

Performance: Accuracy 50.45% (Corresponding train set accuracy = 58%)

Story 2: Model - Sklearn MLP

Training Set best performance:

Layers Used - 5 (Hidden sizes: 64, 128, 256)

Solver: Adam | Learning Rate: Adaptive

Iterations: 840 | Batch Size: 32

Error Tolerance: 1e-4

Performance: Accuracy 98.6% (Corresponding test set accuracy = 38%)

Test Set best performance:

Layers Used - 4 (Hidden sizes: 32,64)

Solver: Adam | Learning Rate: Adaptive

Iterations: 320 | Batch Size: 32

Error Tolerance: 1e-4

Performance: Accuracy 48% (Corresponding test set accuracy = 56.45%)

Inference: Keras Sequential was outperforming Sklearn MLP by some margin during most of the experimentation. We decided to move forward with Keras package for further model analysis, tuning and selection.

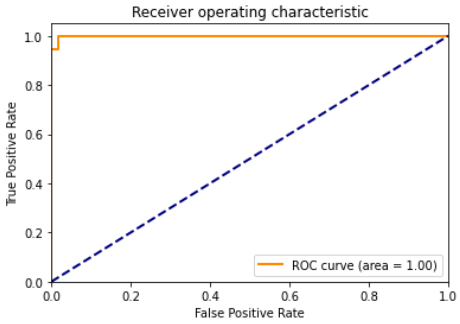
Story 3: ROC curve Analysis:

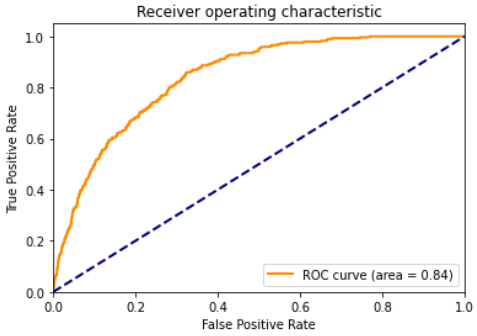
Roc curves were used to check the performance on some individual labels. This step was necessary to check whether the models are making any predictions or not. There were 206 labels, so checking for all of them wasn't feasible. For most of the labels analysed, it seemed like the performance on the training set was really good and on the test it was average.

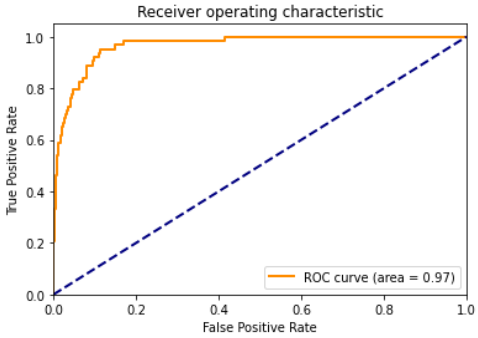
(These results are not to be interpreted as the final ones because they were just used for preliminary study to determine whether the model is learning the data at all.)

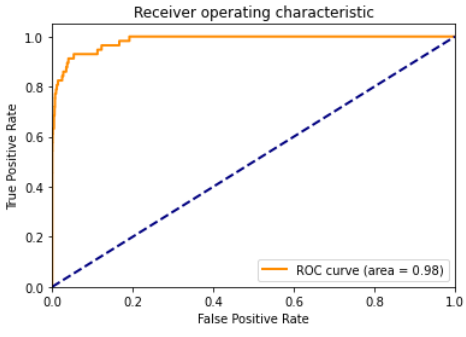
`

Training Set

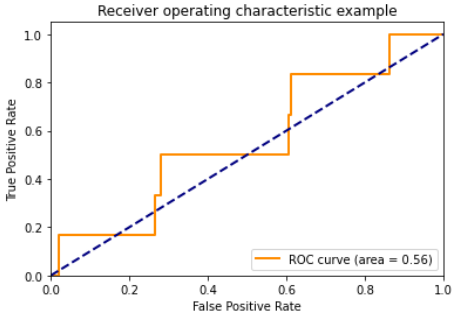


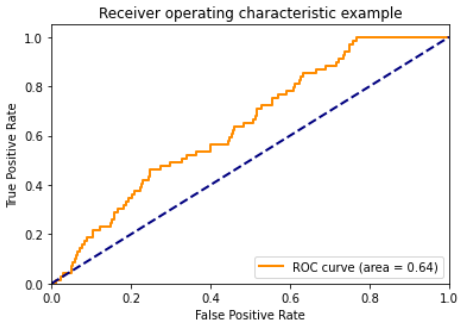


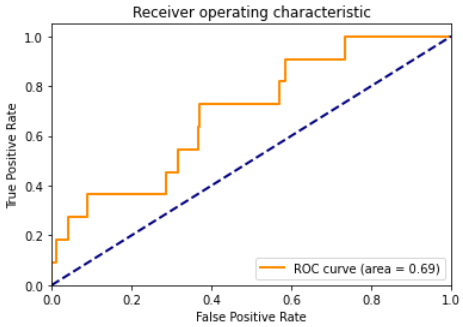




Test Set







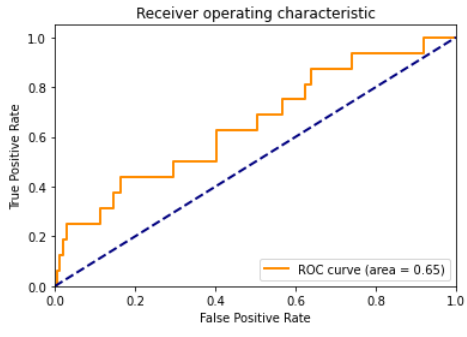


Fig: ROC curve analysis of training data and test data

Epic 3: Model Selection

Story 1:

This is the section in machine learning studies where various models with different hyperparameters are tested on the basis of some metric score and the best models are selected. The Keras package offers various tools to create a deep learning neural network. It used to create and train various Sequential models with Dense layer settings. The parameters were varied and the precision scores recorded for each model. We as a precaution have already done PCA to prevent overfitting of the data. To facilitate this even furthur we used the following:

- *Task 1: Regularization*

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term. Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will reduce overfitting quite a bit. We ended up using only L2 because it was giving us better results on the same model than L1.

- *Task 2: Dropout*

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "*dropped out*." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer.

Below displayed are the tested models. The highlighted models are the ones having the highest precision(micro-average) for the training and the test set at 0.5 probability threshold.

Regularizer - L2	Layer Sizes	Dropout	Batch/epochs	Train Acc	Test Acc	Train Prec	Test Prec
0.01	1024(3)	0.5 , 0.5 ,0.4	1024/1000	76.35	50.1	96.5	76.2
0.01	1024(3)	0.5(3)	1024/2000	64.6	50.17	98.4	76.7
0.001	1024, 256, 128, 64	0.25(3)	32/200	49.01	48.75	91.6	87
0.001	1024, 256, 128, 64	0.2(3)	256/400	54.5	50.4	94.1	80.8
0.001	1024, 256, 128, 64	0.2(3)	256/1000	56.9	51.45	94.2	79.5
0.001	1024, 256, 128, 64	0.3, 0.25, 0.2	256/2000	55.8	50	94.17	80.86
0.001	1024, 256, 128, 64	0.3, 0.25, 0.2	500/1000	58.67	50.07	94.64	81.7
0.001	1024, 256, 128, 64	0.5,0.4,0.3	1024/2000	99.7	49.8	96.5	79.7
0.0001	32, 64, 128, 256	0.25(1)	32/500	54.24	48.93	91.3	79
0.0001	1024, 256, 128, 64	0.25(3)	32/100	50.35	49.5	89.5	79.7
0.0001	1024, 256, 128, 64	0.25(3)	256/200	51.28	50.01	95.2	78.8
0.0001	1024, 256, 128, 64	0.2(3)	256/300	52.97	49.3	96.8	68.5
0.0001	1024, 256, 128, 64	0.2(3)	1024/300	74.93	49	95.9	71.4
0.0001	1024, 256, 128, 64	0.3(3)	500/1000	84.3	49	97.6	71.8
0.0001	1024, 256, 128, 64	0.3(3)	1024/1000	95.5	48.6	98.6	67.3
0.0001	1024, 256, 128, 64	0.3, 0.25, 0.2	1024/1500	94.5	49.25	99.16	65.5
0.0001	1024, 256, 128, 64	0.3, 0.25, 0.2	512/1500	66.7	50.34	98.4	67.4
0.0001	1024, 512, 256, 128, 64	0.3(3)	1024/2000	99.6	49.5	99.5	70.35
0.0001	1024, 1024, 256	0.3(2)	1024/2000	99.8	49	99.79	65.14
0.0001	1024, 1024, 256	0.7(2)	1024/1000	67	50.45	97	81.8
0.0001	1024, 1024, 256	0.7(2)	1024/1200	69.39	50.53	97.1	81.43

Story 2:

The model having the highest precision for the test set was tuned further and a better precision was achieved. The primary change was made to increase the number of epochs from 200 to 1000 for better learning of the data.

Regularizer - L2	Layer Sizes	Dropout	Batch/epochs	Train Prec	Test Prec
0.001	1024, 256, 128, 64	0.25(3)	32/500	92.21	87.14
0.001	1024, 256, 128, 64	0.25(3)	32/1000	92.42	87.94
0.001	1024, 256, 128, 64	0.25(3)	32/1200	90.2	86.6
0.001	1024, 256, 128, 64	0.2(3)	32/1000	90.8	85.7
0.0001	1024, 256, 128, 64	0.25(3)	32/1000	93.01	84.5

Story 3:

Finally, two models were selected for further analysis. The model having the second highest precision for the test set and a better precision score for the training set was also selected along with the one having highest precision for the test set. These models have very different parameters but both were working well in terms of micro-average precision at 0.5 probability threshold. So we decided to study both of them. Below displayed are the complete details about the selected models.

Model 1: Batch Size - 32, Epochs - 1000

```
# Keras Sequential
def NN_Model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(1024, input_dim = n_inputs, kernel_initializer='he_uniform',
                    kernel_regularizer=l2(0.001), activation = 'relu'))
    model.add(Dropout(0.25))
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.25))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(n_outputs, activation = 'sigmoid'))
    opt = keras_opt.Nadam(lr = 0.001)
    model.compile(loss = 'binary_crossentropy', optimizer = opt, metrics=['accuracy'])
    return model
```

Model 2: Batch Size - 1024, Epochs - 1000

```
# Keras Sequential
def NN_Model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(1024, input_dim = n_inputs, kernel_initializer='he_uniform',
                    kernel_regularizer=l2(0.0001), activation = 'relu'))
    model.add(Dropout(0.7))
    model.add(Dense(1024, activation = 'relu'))
    model.add(Dropout(0.7))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(n_outputs, activation = 'sigmoid'))
    opt = keras_opt.Nadam(lr = 0.001)
    model.compile(loss = 'binary_crossentropy', optimizer = opt, metrics=['accuracy'])
    return model
```

Epic 4: Probability thresholding

Story 1:

We have been using a fixed probability threshold of 0.5 to convert the output probabilities to labels. This is a method of analysis used in classification to get higher precision values by changing the probability threshold for class labelling from the default value of 0.5 to something else.

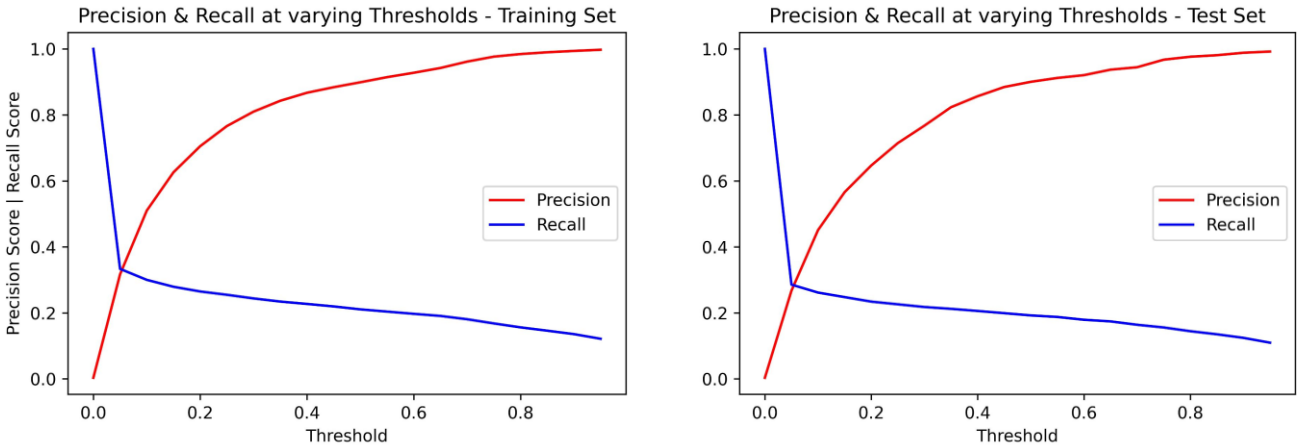
The next step of the analysis was to vary this threshold and determine the probability when the micro-average precision score reached 90%. Below displayed are the resulting threshold values.

- **Model 1: Training set = 0.55 | Test set = 0.6**
- **Model 2: Training set = 0.20 | Test set = 0.7**

Important Note: The full classification report of 206 labels were studied to check for the individual precision score on each label, especially for the test set. We found that Model 2 had some value of precision score for 62 labels(out of 206) while Model 1 had for 30(out of 206). Reason could be the target set is very sparse and the labels are highly unbalanced in terms of support. So, the majority support labels were recognised more by the models. This also means that Model 2 is performing better for more labels with lower support.

Story 2: Effect of varying threshold on Precision - Recall scores

Below displayed are the results:



Inference: We can see a very usual trend in classification, where the recall decreases as the precision increases. When we increase the threshold, the number or results of interest returned decreases but most of them are correctly labelled compared to the original label set. Therefore, we see a decrease in the recall score. If we decrease the threshold, the number of results returned increases but there's also an increase in the number of false positives returned. Therefore, the precision decreases and we see an inverse relationship.

Epic 5: Recall Analysis

After we were successful in achieving 90% precision for both training and test sets we wanted to study the individual recall scores of the labels. There were labels having very high support as well as very low ones. It is a highly sparse and imbalanced label set. To see weather the models are performing well for both cases we decided to study the recall further. At first the support ratio was calculated for all the labels. Then the labels were sorted according to the support ratio and their individual recall values were recorded.

$$Support\ ratio\ of\ \ a\ label = \frac{No.\ of\ \ observations\ denoted\ as\ class\ 1}{Total\ no.\ of\ \ observations}$$

Since there are 206 labels and it's not feasible to display all of them, I will be displaying the recall results of the **top 100 labels sorted by support ratio.**

Story 1: Model 1 - Training set: Precision reached 90% at 0.55

Total labels having recall > 0 = 45.
Labels in top 100 having recall > 0 = 33 (about 73.33% of labels having some recall were concentrated in top 100)

Label	Support Ratio	Recall Score
136	0.0349374	0.87902
163	0.0304863	0.993092
71	0.0182666	0
79	0.0178047	0
177	0.0169648	0
77	0.0168808	0.0157729
99	0.0154111	0
10	0.0151172	0
63	0.0142773	0.65614
80	0.0141093	0.660377
199	0.0132695	0.731518
4	0.0126396	0
149	0.0124717	0.340164
109	0.0118838	0.515419
54	0.0117998	0
89	0.0117158	0.39823
119	0.0114638	0.38009
9	0.0113379	0
182	0.0112119	0
96	0.0111699	0.905473
151	0.0110859	0
105	0.0101201	0
176	0.00991014	0
169	0.00936424	0.857143
43	0.00806248	0
3	0.0079785	0

Label	Support Ratio	Recall Score
202	0.00713866	0.0428571
94	0.0069287	0
83	0.00663475	0
153	0.00634081	0.0887097
133	0.00545897	0.125
194	0.005333	0.257732
78	0.00508104	0
159	0.00499706	0
45	0.00482909	0
157	0.00482909	0
93	0.00445116	0
103	0.00445116	0.58427
72	0.00436718	0
166	0.00432519	0.0769231
61	0.00428319	0
156	0.00411523	0
36	0.00407323	0.0365854
7	0.00403124	0
38	0.00403124	0.210526
144	0.00403124	0
110	0.00390527	0.719512
118	0.00386327	0.123288
17	0.0037373	0
44	0.0037373	0
131	0.00356933	0
162	0.00352734	0

Label	Support Ratio	Recall Score
18	0.00335937	0
41	0.00335937	0
98	0.00310742	0
128	0.00310742	0
5	0.00306542	0
21	0.00306542	0
28	0.00306542	0
114	0.00306542	0
200	0.00306542	0.0847458
102	0.00302343	0
108	0.00302343	0
127	0.00302343	0.5
184	0.00298144	0.0909091
51	0.00285546	0.0677966
49	0.00281347	0
171	0.00281347	0
11	0.00277148	0.211538
122	0.00260351	0
146	0.00260351	0.102041
124	0.00256152	0
143	0.00256152	0
148	0.00256152	0.0416667
40	0.00251953	0
101	0.00251953	0.212766
187	0.00251953	0
104	0.00247753	0

Label	Support Ratio	Recall Score
40	0.00251953	0
101	0.00251953	0.212766
187	0.00251953	0
104	0.00247753	0
95	0.00235156	0
168	0.00235156	0
56	0.00230957	0
155	0.00230957	0
6	0.00226757	0
64	0.00226757	0
68	0.00226757	0
116	0.0021416	0
88	0.00209961	0.025641
31	0.00205761	0
16	0.00201562	0
67	0.00201562	0
76	0.00201562	0
84	0.00201562	0
134	0.00201562	0
164	0.00201562	0
198	0.00201562	0
32	0.00197363	0
178	0.00184765	0.0540541
23	0.00180566	0
13	0.00176367	0.0606061
55	0.00176367	0

Story 2: Model 1 - Test set: Precision reached 90% at 0.6

Total labels having recall > 0 = 30.
Labels in top 100 having recall > 0 = 26 (about 92.85% of labels having some recall were concentrated in top 100)

Label	Support Ratio	Recall Score
136	0.0349374	0.832402
163	0.0304863	1
71	0.0182666	0
79	0.0178047	0
177	0.0169648	0
77	0.0168808	0
99	0.0154111	0
10	0.0151172	0
63	0.0142773	0.709091
80	0.0141093	0.732394
199	0.0132695	0.762712
4	0.0126396	0
149	0.0124717	0.339623
109	0.0118838	0.410714
54	0.0117998	0
89	0.0117158	0.358491
119	0.0114638	0.326923
9	0.0113379	0
182	0.0112119	0
96	0.0111699	0.846154
151	0.0110859	0
105	0.0101201	0
176	0.00991014	0
169	0.00936424	0.882353
43	0.00806248	0
3	0.0079785	0

Label	Support Ratio	Recall Score
202	0.00713866	0
94	0.0069287	0
83	0.00663475	0
153	0.00634081	0.037037
133	0.00545897	0.384615
194	0.005333	0.366667
78	0.00508104	0
159	0.00499706	0
45	0.00482909	0
157	0.00482909	0
93	0.00445116	0
103	0.00445116	0.352941
72	0.00436718	0
166	0.00432519	0.04
61	0.00428319	0
156	0.00411523	0
36	0.00407323	0.0666667
7	0.00403124	0
38	0.00403124	0.15
144	0.00403124	0
110	0.00390527	0.818182
118	0.00386327	0
17	0.0037373	0
44	0.0037373	0
131	0.00356933	0
162	0.00352734	0

Label	Support Ratio	Recall Score
18	0.00335937	0
41	0.00335937	0
98	0.00310742	0
128	0.00310742	0
5	0.00306542	0
21	0.00306542	0
28	0.00306542	0
114	0.00306542	0
200	0.00306542	0
102	0.00302343	0
108	0.00302343	0
127	0.00302343	0.571429
184	0.00298144	0.1875
51	0.00285546	0.333333
49	0.00281347	0
171	0.00281347	0
11	0.00277148	0.0714286
122	0.00260351	0
146	0.00260351	0.0769231
124	0.00256152	0
143	0.00256152	0
148	0.00256152	0
40	0.00251953	0
101	0.00251953	0.307692
187	0.00251953	0
104	0.00247753	0

Label	Support Ratio	Recall Score
40	0.00251953	0
101	0.00251953	0.307692
187	0.00251953	0
104	0.00247753	0
95	0.00235156	0.1
168	0.00235156	0
56	0.00230957	0
155	0.00230957	0
6	0.00226757	0
64	0.00226757	0
68	0.00226757	0
116	0.0021416	0
88	0.00209961	0
31	0.00205761	0
16	0.00201562	0
67	0.00201562	0
76	0.00201562	0
84	0.00201562	0
134	0.00201562	0
164	0.00201562	0
198	0.00201562	0
32	0.00197363	0
178	0.00184765	0
23	0.00180566	0
13	0.00176367	0
55	0.00176367	0

Story 3: Model 2 - Training set: Precision reached 90% at 0.2

Total labels having recall > 0 = 206.
Labels in top 100 having recall > 0 = 100 (This model has recall values for every label in the training set)

Label	Support Ratio	Recall Score
136	0.0349374	0.926493
163	0.0304863	1
71	0.0182666	0.154079
79	0.0178047	0.285294
177	0.0169648	0.219512
77	0.0168808	0.369085
99	0.0154111	0.126667
10	0.0151172	0.151408
63	0.0142773	0.964912
80	0.0141093	0.950943
199	0.0132695	0.968872
4	0.0126396	0.093617
149	0.0124717	0.963115
109	0.0118838	0.955947
54	0.0117998	0.274419
89	0.0117158	0.973451
119	0.0114638	0.99095
9	0.0113379	0.251163
182	0.0112119	0.239437
96	0.0111699	0.955224
151	0.0110859	0.25
105	0.0101201	0.0854271
176	0.00991014	0.261364
169	0.00936424	0.989418
43	0.00806248	0.202797
3	0.0079785	0.154321

Label	Support Ratio	Recall Score
202	0.00713866	0.914286
94	0.0069287	0.279412
83	0.00663475	0.382812
153	0.00634081	0.677419
133	0.00545897	0.798077
194	0.005333	0.958763
78	0.00508104	0.333333
159	0.00499706	0.598039
45	0.00482909	0.386364
157	0.00482909	0.606383
93	0.00445116	0.430233
103	0.00445116	0.94382
72	0.00436718	0.329412
166	0.00432519	0.589744
61	0.00428319	0.337209
156	0.00411523	0.361446
36	0.00407323	0.646341
7	0.00403124	0.447368
38	0.00403124	0.973684
144	0.00403124	0.291139
110	0.00390527	0.926829
118	0.00386327	0.958904
17	0.0037373	0.407895
44	0.0037373	0.295775
131	0.00356933	0.338235
162	0.00352734	0.42623

Label	Support Ratio	Recall Score
18	0.00335937	0.353846
41	0.00335937	0.555556
98	0.00310742	0.190476
128	0.00310742	0.344828
5	0.00306542	0.288462
21	0.00306542	0.686275
28	0.00306542	0.240741
114	0.00306542	0.534483
200	0.00306542	0.779661
102	0.00302343	0.457627
108	0.00302343	0.542373
127	0.00302343	0.948276
184	0.00298144	0.927273
51	0.00285546	0.762712
49	0.00281347	0.654545
171	0.00281347	0.897959
11	0.00277148	0.903846
122	0.00260351	0.38
146	0.00260351	0.877551
124	0.00256152	0.52
143	0.00256152	0.313725
148	0.00256152	0.854167
40	0.00251953	0.468085
101	0.00251953	0.87234
187	0.00251953	0.466667
104	0.00247753	0.404255

Label	Support Ratio	Recall Score
40	0.00251953	0.468085
101	0.00251953	0.87234
187	0.00251953	0.466667
104	0.00247753	0.404255
95	0.00235156	0.869565
168	0.00235156	0.276596
56	0.00230957	0.627907
155	0.00230957	0.55102
6	0.00226757	0.7
64	0.00226757	0.621622
68	0.00226757	0.477273
116	0.0021416	0.904762
88	0.00209961	1
31	0.00205761	0.828571
16	0.00201562	0.44186
67	0.00201562	0.513514
76	0.00201562	0.735294
84	0.00201562	0.756098
134	0.00201562	0.615385
164	0.00201562	0.631579
198	0.00201562	0.472222
32	0.00197363	0.459459
178	0.00184765	0.648649
23	0.00180566	0.857143
13	0.00176367	0.909091
55	0.00176367	0.588235

Story 4: Model 2 - Test set: Precision reached 90% at 0.7

Total labels having recall > 0 = 60.
Labels in top 100 having recall > 0 = 45 (about 75% of labels having some recall values were concentrated in top 100)

Label	Support Ratio	Recall Score
136	0.0349374	0.821229
163	0.0304863	0.986395
71	0.0182666	0
79	0.0178047	0
177	0.0169648	0
77	0.0168808	0.0352941
99	0.0154111	0
10	0.0151172	0
63	0.0142773	0.763636
80	0.0141093	0.84507
199	0.0132695	0.745763
4	0.0126396	0
149	0.0124717	0.528302
109	0.0118838	0.607143
54	0.0117998	0
89	0.0117158	0.509434
119	0.0114638	0.519231
9	0.0113379	0
182	0.0112119	0
96	0.0111699	0.415385
151	0.0110859	0
105	0.0101201	0
176	0.00991014	0
169	0.00936424	0.911765
43	0.00806248	0
3	0.0079785	0

Label	Support Ratio	Recall Score
202	0.00713866	0.0666667
94	0.0069287	0
83	0.00663475	0
153	0.00634081	0.0740741
133	0.00545897	0.0769231
194	0.005333	0.566667
78	0.00508104	0
159	0.00499706	0
45	0.00482909	0
157	0.00482909	0
93	0.00445116	0.05
103	0.00445116	0.529412
72	0.00436718	0
166	0.00432519	0.2
61	0.00428319	0
156	0.00411523	0
36	0.00407323	0.133333
7	0.00403124	0
38	0.00403124	0.45
144	0.00403124	0
110	0.00390527	0.909091
118	0.00386327	0.210526
17	0.0037373	0
44	0.0037373	0
131	0.00356933	0
162	0.00352734	0

Label	Support Ratio	Recall Score
18	0.00335937	0
41	0.00335937	0
98	0.00310742	0
128	0.00310742	0
5	0.00306542	0
21	0.00306542	0
28	0.00306542	0
114	0.00306542	0.0666667
200	0.00306542	0.0714286
102	0.00302343	0
108	0.00302343	0
127	0.00302343	0.642857
184	0.00298144	0.1875
51	0.00285546	0.333333
49	0.00281347	0
171	0.00281347	0
11	0.00277148	0.142857
122	0.00260351	0.0833333
146	0.00260351	0.230769
124	0.00256152	0.0909091
143	0.00256152	0
148	0.00256152	0
40	0.00251953	0
101	0.00251953	0.461538
187	0.00251953	0
104	0.00247753	0

Label	Support Ratio	Recall Score
40	0.00251953	0
101	0.00251953	0.461538
187	0.00251953	0
104	0.00247753	0
95	0.00235156	0.5
168	0.00235156	0
56	0.00230957	0.0833333
155	0.00230957	0
6	0.00226757	0
64	0.00226757	0
68	0.00226757	0
116	0.0021416	0
88	0.00209961	0.181818
31	0.00205761	0
16	0.00201562	0
67	0.00201562	0
76	0.00201562	0.0714286
84	0.00201562	0
134	0.00201562	0
164	0.00201562	0
198	0.00201562	0
32	0.00197363	0
178	0.00184765	0
23	0.00180566	0
13	0.00176367	0
55	0.00176367	0

Inference: We can see that Model 2 is performing better than Model 1 both for the training set and the test set. Model 2 has recall values for all the labels in the training set. That means it's learning the data really well. For the test set it has double the # of labels that have some recall value compared to Model 1.

○ 7.3.3 Findings and Results - Deep Learning Modelling

- *Aggregate recall analysis - Model 2:*

We wanted to check how the model was performing as a whole. So we decided to examine Global, Majority and Minority recall values.

Global Recall: This is the sum of the recall values of all the labels weighted by their support ratio.

$$Global\ recall = \sum_{for\ each\ label} (support\ ratio \times recall\ score)$$

Majority Recall: This is the total recall value of the top 100 labels weighted by the scaled support ratio. The support ratio was scaled so as to mask the effect of the remaining labels, as if the model only had the majority support labels.

$$scaled\ support\ ratio = for\ each\ label: \frac{support\ ratio}{\sum_{label\ 1}^{label\ 100} (support\ ratio)}$$

$$Majority\ recall = \sum_{label\ 1}^{label\ 100} (scaled\ support\ ratio \times recall\ score)$$

Minority Recall: This is the total recall value of the labels 51-100 (considered minor labels) weighted by the scaled support ratio. The support ratio was scaled so as to mask the effect of the remaining labels, as if the model only had the minor labels

$$scaled\ support\ ratio = for\ each\ label: \frac{support\ ratio}{\sum_{label\ 51}^{label\ 100} (support\ ratio)}$$

$$Minority\ recall = \sum_{label\ 51}^{label\ 100} (scaled\ support\ ratio \times recall\ score)$$

.

Below displayed are the corresponding values for Model 2:

	Training Set	Test Set
Global Recall	0.43	0.16
Majority Recall	0.58	0.25
Minority Recall	0.6	0.07

- *Aggregate recall analysis - 10 Models:*

Once we've established the method for determining the aggregate recall scores, this time 10 different models were selected from the model selection phase and trained. We wanted to see if any other model performs better than model 2 in terms of recall scores. The threshold for 90% precision was determined for each model and then used for calculating individual recall values of each label.

Below displayed are the models selected for analysis of recall scores and the probability thresholds when precision reached 90%.

Regularizer - L2	Layer Sizes	Dropout	Batch/epochs	90% Precision Threshold	
				Training Set	Test Set
0.001	1024, 256, 128, 64	0.25(3)	32/200	0.45	0.5
0.0001	1024, 1024, 256	0.7(2)	1024/1000	0.3	0.75
0.01	1024(3)	0.5 , 0.5 ,0.4	1024/1000	0.3	0.85
0.001	1024, 256, 128, 64	0.2(3)	256/400	0.35	0.6
0.001	1024, 256, 128, 64	0.3, 0.25, 0.2	256/2000	0.35	0.7
0.001	1024, 256, 128, 64	0.5,0.4,0.3	1024/2000	0.4	0.7
0.0001	1024, 256, 128, 64	0.25(3)	256/200	0.35	0.85
0.0001	1024, 256, 128, 64	0.3(3)	1024/1000	0.2	0.9
0.0001	1024, 256, 128, 64	0.3(3)	512/1000	0.25	0.85
0.0001	1024, 512, 256, 128, 6	0.3(4)	1024/2000	0.1	0.95

Results:

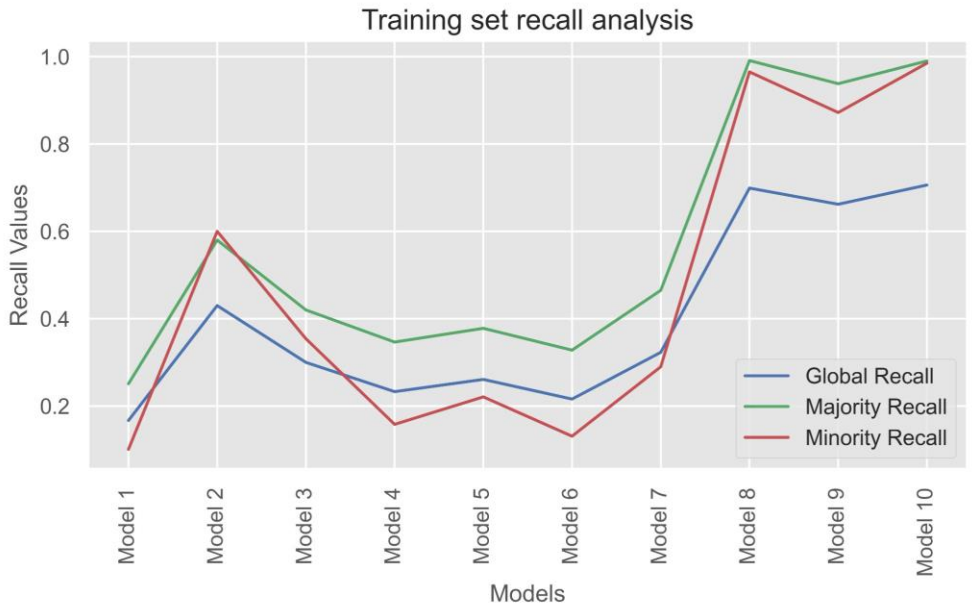
Training Set

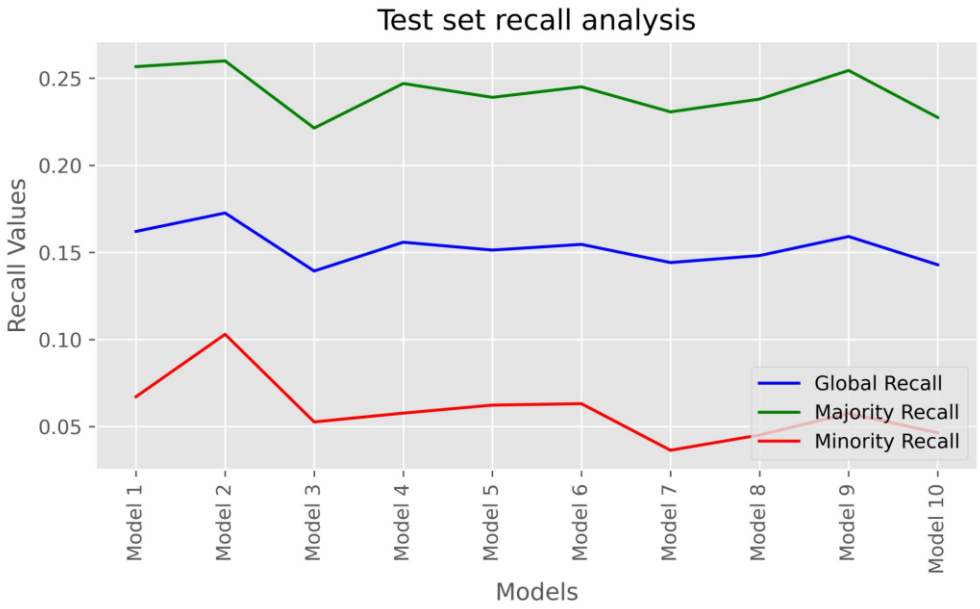
Index	Global Recall	Majority Recall	Minority Recall
Model 1	0.42848	0.556972	0.689341
Model 2	0.16761	0.251604	0.101391
Model 3	0.525519	0.706691	0.87228
Model 4	0.233383	0.346566	0.158127
Model 5	0.261707	0.378648	0.221895
Model 6	0.216822	0.328525	0.131355
Model 7	0.323088	0.465914	0.290899
Model 8	0.699882	0.991142	0.965576
Model 9	0.662942	0.938298	0.872574
Model 10	0.706684	0.999141	0.985493

Test Set

Index	Global Recall	Majority Recall	Minority Recall
Model 1	0.162048	0.256652	0.0671502
Model 2	0.172626	0.259951	0.102968
Model 3	0.139325	0.221387	0.0526835
Model 4	0.155841	0.246967	0.0577237
Model 5	0.151336	0.239009	0.0623314
Model 6	0.154601	0.245051	0.0631527
Model 7	0.144161	0.230667	0.0364069
Model 8	0.148175	0.238015	0.0451607
Model 9	0.159115	0.254444	0.0575561
Model 10	0.142906	0.227465	0.0464342

Results - Visualized





Inference: We can see that Model 8,9,10 are performing really well for the training set but for the test set Model 2 has the higher recall values. It also has a decent recall for the training set. This might be due to the fact that Model 2 does not overfit the data and generalize it more than the other ones.

- *PCA - Capturing more variance:*

Previously we captured a variance of 72% for the Gene vectors and 75% for the Cell vectors. We wanted to see if capturing more variance would improve the performance of the models. This time we decided to capture **95% variance** for both parts of the data.

- Number of principal components needed for Gene vectors = 600
- Number of principal components needed for Cell vectors = 82

Total feature space is now **684** (2 Categorical, 682 Numerical). This might improve the performance of the model but also result in the training time to go up significantly. We will examine that at the end of the project.

Below displayed is sample of the data:

	cp_time	cp_dose	0	1	2	3	4	5	6	7	...	73	74	75	76
0	0	0	-5.587642	3.924553	9.156055	-8.027759	5.067363	1.477832	3.439147	1.455533	...	-0.253762	0.217414	0.008822	-0.851564
1	2	0	-4.950738	3.906345	-11.198661	5.965759	0.709618	0.333061	1.084945	-0.618108	...	-0.209962	0.791759	-0.175913	0.636835
2	1	0	1.138424	-7.224107	-5.560808	-0.770526	0.685674	3.628206	-2.251991	2.773414	...	0.571611	1.133288	-0.219006	-1.292794
3	1	0	11.261830	-8.715983	-4.319024	-5.899394	-6.887881	-3.134697	-2.174687	6.905652	...	0.336012	0.154716	0.149124	0.044932
4	2	1	-6.685881	-2.359479	-10.702104	-4.134541	-7.535818	-8.971010	-3.763539	-2.875785	...	0.336353	-0.477286	-1.597157	0.388531

5 rows × 684 columns

Results:

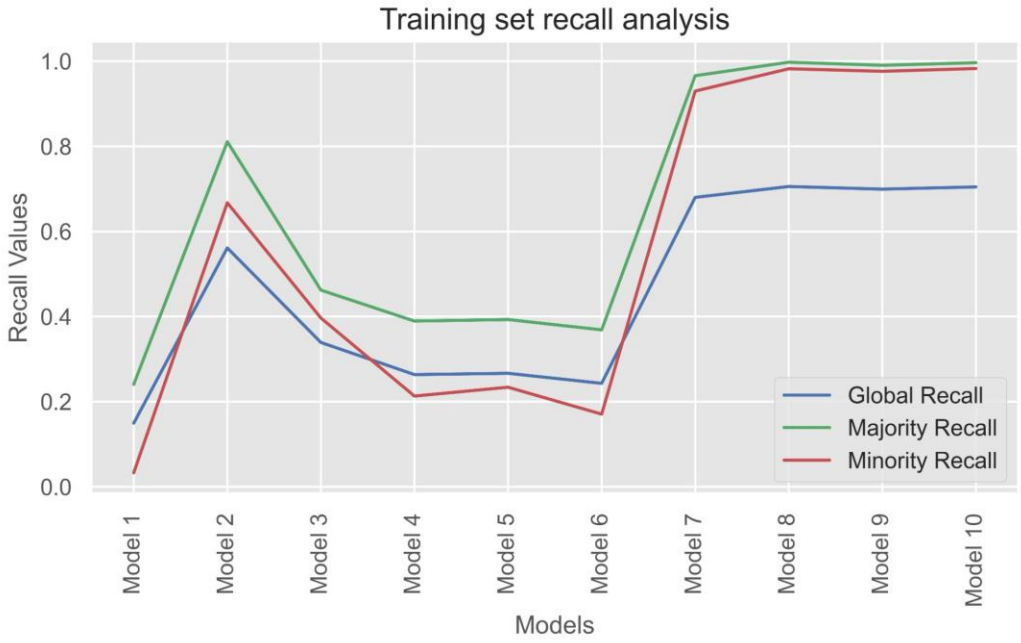
Training Set

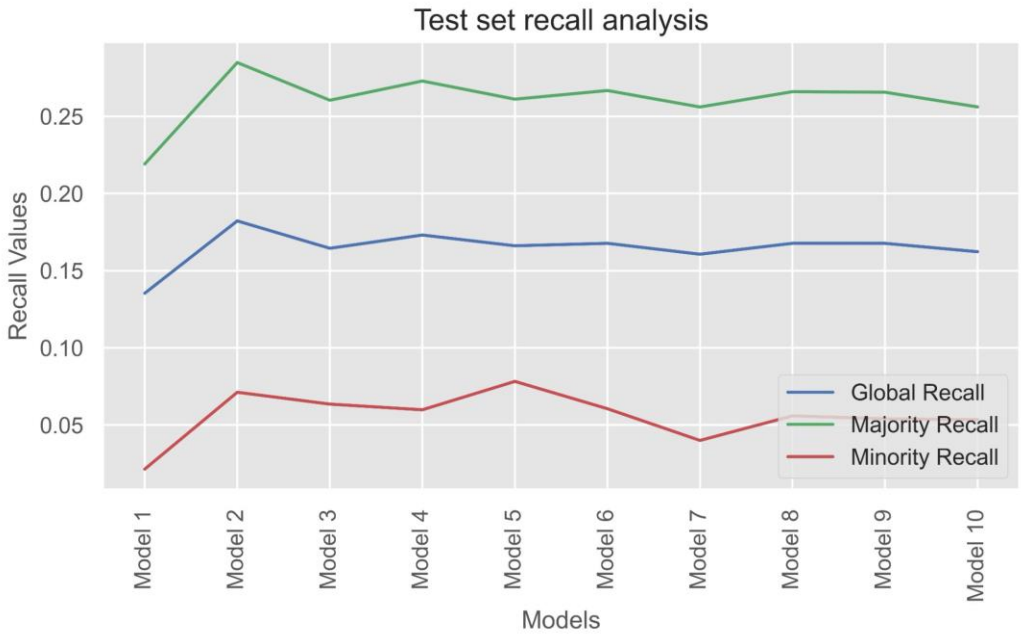
Index	Global Recall	Majority Recall	Minority Recall
Model 1	0.14946	0.240642	0.0330292
Model 2	0.561271	0.810792	0.667309
Model 3	0.339177	0.462168	0.396611
Model 4	0.263639	0.38938	0.213154
Model 5	0.266681	0.393019	0.233976
Model 6	0.243099	0.368763	0.170755
Model 7	0.680009	0.965926	0.929747
Model 8	0.705574	0.997586	0.982165
Model 9	0.699361	0.990354	0.976239
Model 10	0.704585	0.996438	0.982743

Test Set

Index	Global Recall	Majority Recall	Minority Recall
Model 1	0.135329	0.219095	0.0213468
Model 2	0.182216	0.284846	0.0711579
Model 3	0.164464	0.260415	0.0634892
Model 4	0.17307	0.272878	0.0598007
Model 5	0.166092	0.261104	0.0782112
Model 6	0.167696	0.266717	0.0604991
Model 7	0.160621	0.256041	0.0399459
Model 8	0.167698	0.265965	0.0558862
Model 9	0.167675	0.265613	0.0539378
Model 10	0.16224	0.25609	0.053481

Results – Visualized





Inference: The recall scores of Model 2 actually improved significantly for the training set. For the test we can see that Model 2 is still the best in terms of Global and Majority recall but for minority recall Model 5 is at par with Model 2. From the training set analysis we can see that Model 5 is not learning the data very well, so this might be just a case of random guessing / prediction being done by the model. Overall Model 2 still seems to generalize data

- *Conclusions and Future Scope*
    - It is possible to achieve 90% precision for almost all the models both for training and test sets.
    - After further analysis, Model 2 came out to be the best overall in terms of generalizing the data better and the recall scores.
    - There were multiple models that could learn the data harder and perform better on the training set but the test set performance was best for Model 2. It had rivaling performance on the training set too.
    - The model performed really well for the labels having high support and average for the lower support labels.
    - For future work, there might be ways to get better individual performance for the minority labels. Retraining the models emphasising only on the minority labels might be of help. More study and research on that is required.

Model Execution Statistics

We can clearly see that the training time went up by quite some margin when the feature space was increased to capture more variance.

	Feature space: 67		Feature space: 682	
Models	Training Time (min)	Prediction Time (sec) (Traning Set + Test Set)	Training Time (min)	Prediction Time (sec) (Traning Set + Test Set)
Model 1	10	1.569	31.3	2.02
Model 2	20.5	0.681	30.4	2.53
Model 3	31	2.427	43.5	3.47
Model 4	5.5	0.656	16.14	1.82
Model 5	26.8	0.652	63.09	1.82
Model 6	18	0.677	37.13	1.95
Model 7	3	0.678	6.23	1.81
Model 8	9	0.684	18.18	1.87
Model 9	10.2	1.025	23.43	2.002
Model 10	30	1.263	50.4	2.88

Project execution environment: SpyderExecution engine: Tensorflow-GPU

Machine Specifications:

- CPU - Intel Core i5-9300H @2.4GHz
- RAM - 16GB-DDR4
- Graphics - NVIDIA GeForce GTX

7.4 Quality Assurance

For Quality Assurance, we had weekly meetings with the CCC team where we would each detail our progress and findings in the form of powerpoint presentations complete with model specifications, graphs, charts, performance metrics etc. They would then send us an email at the end of each meeting detailing the next steps for each team member and the techniques, improvements and recommendations they wanted us to implement for the next week. This way we made sure we were in sync with the requirements and vision of the CCC team and delivered results that they expected from us. We also had a general goal of improving precision score above 90% and had some benchmark models like the f1 score of MAGNET research paper mentioned earlier that was very close to our score.

8 Project Support

We are wholeheartedly willing to continue supporting CCC if given the opportunity to work further on this project. For that reason, we have already submitted each of our weekly presentations as well as the final presentation to them and will also share this project document and all our code in the form of a github repository that they can refer to if they want more information on our project. We have also provided them with our contact details so they can reach out to us if they have any questions.

9 Project Retrospective

Altogether, we had a very great learning experience with respect to the research aspects of Data Science and Machine Learning. We will be very grateful to the team of CCC and Illinois Tech for providing us with this opportunity. One area of improvement we thought could be implemented was code reviews which would have helped us to maintain industry level code standards and fix bugs.

10 References or Other Supporting Documents

[1] E. M. Rudd, L. P. Jain, W. J. Scheirer and T. E. Boult, "The Extreme Value Machine," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 762-768, 1 March 2018, doi: 10.1109/TPAMI.2017.2707495.

[2] EVM - https://github.com/EMRResearch/ExtremeValueMachine/blob/master/evm.py

[3] Keras API Reference - https://keras.io/api/

[4] Sklearn MLP - https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[5] Stacking - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

[6] OneVsRestClassifier - https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html

11 Document Revision History

- 01/22/2021 - Preliminary Discussion on Safety Prediction System Project with CCC Information Services Technical Team.
- 01/29/2021 - The Research & Development Team of CCC Information Services came up with an update on the project we had to work on. The new project was based on Ensemble of two AI models for predicting damaged parts of a vehicle. Several research papers related to ensemble modelling techniques were shared by the team.
- 02/05/2021 - A brief discussion on the research pacers and Ensemble methods studied so far. Each team member was assigned two ensemble methods and they are required to choose any public dataset and build models on it to compare the results from those two algorithms. The results will be presented to the team in the next session. The contents of the Project Scope document were also discussed and approved.
- 05/08/2021 - Added in detail description of project execution strategy, findings and results

.