# Solution overview

## How to use UI app?

We have created two modules in application.

1. Single paper classification  
   User can simply copy paste the title and abstract into the app and classify the whitepaper.  
   Graphical user interface, application

   Description automatically generated
2. Bulk paper classification  
   User can upload the comma separated (.csv) file and get the onscreen predictions and download the output as well.

Table

Description automatically generated

## Project Structure

.

│ requirements.txt

│

├───code

│ mainui.py

│ prediction\_module.py

│

├───documents

│ 1353271\_Mayur Kagathara\_UC6\_DS-ML Classifications.docx

│ UC6 – DS-ML Classifications.docx

│

├───input\_data

│ Testing\_Data.csv

│ Training\_Data.csv

│

├───notebooks

│ EDA\_Cleaning.ipynb : Cleaning, transforming raw data

│ EDA\_Cleaning.py

│ Prediction.ipynb : Prediction on test data

│ Prediction.py

│ Training\_validation.ipynb : Training and cross validation

│ Training\_validation.py

│

└───output\_files

logreg\_model.sav : Logistic regression model

mnb\_model.sav : Multinomial Naive bayes model

train\_tfidf\_6000f.csv : Cleaned and transformed Train data

predictions.csv : predictions submission file

voting\_model.sav : Voting classifier model

### How to run the app

Download the project and install libraries using requirements.txt. Run streamlit run /code/mainui.py

$pip install -r requirements.txt

$cd code

$streamlit run mainui.py

Text

Description automatically generated

# Data gathering

Data is provided as part of hackathon. Original data is available at below link.

<https://www.kaggle.com/vin1234/janatahack-independence-day-2020-ml-hackathon>

# EDA and Cleaning

# Exploratory Data Analysis

* Data is clean, without any missing values.
* We have all text features – Title and abstract
* There are 6 different topics/categories. This is multilabel classification problem.
* Dataset is imbalanced.

# Cleaning

* Words are converted to lower cases.
* All special characters and punctuations are removed.
* English Stop words are removed.
* Remaining words are stemmed using snowball stemmer.

# Transformation

## TFIDF

We have used TFIDF (Term Frequency Inverse Document Frequency) to convert text data into vectors so that we can use machine learning models.

We tried to build the model for below two cases:

1. Without stemming using top 800 features of title and 2000 features of abstract.
2. With stemming using top 1000 features of title and 5000 features of abstract.

# Training and Model Selection

## 1. Without stemming and using 800+2000 features from TFIDF,

**Title: 800 and abstract: 2000 (not good results)**

**a. In SVC we got,**

Topic\_1 accuracy= 0.7422611036339165

Topic\_2 accuracy= 0.8048452220726783

Topic\_3 accuracy= 0.8011440107671601

Topic\_4 accuracy= 0.7802826379542396

Topic\_5 accuracy= 0.9761103633916555

Topic\_6 accuracy= 0.9862045760430687

It took so much time in training and run time as well due to high dimensionality.

Train time: 40 min

Test time: 5 min

**b. In Logistic regression with C=1 we got,**

Topic\_1 accuracy= 0.7176985195154778

Topic\_2 accuracy= 0.7940780619111709

Topic\_3 accuracy= 0.7934051144010767

Topic\_4 accuracy= 0.7711978465679677

Topic\_5 accuracy= 0.9761103633916555

Topic\_6 accuracy= 0.9862045760430687

Train Time: 23 sec

Test time: 10 sec

Here SVC is working good but training time is too much due to its higher order of time complexity.

Logistic regression is working great for topic 5 and 6. But average for rest of the classes.

## 2. With stemming and using 1000+5000 features from TFIDF,

**Title: 1000 and Abstract: 5000**

### 2.1 Multinomial Naive bayes

**In Multinomial Naive bayes with alpha=0.001 we got, (using grid search CV)**

Topic\_1 accuracy= 0.8352777777777778

Topic\_2 accuracy= 0.9283333333333333

Topic\_3 accuracy= 0.8913888888888889

Topic\_4 accuracy= 0.8602777777777778

Topic\_5 accuracy= 0.9725

Topic\_6 accuracy= 0.9894444444444445

Train Time: 8.632 sec

F1-micro: 0.7944068166921564

Grid Search CV (5-fold)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alpha | split0 test\_score | split1 test\_score | split2 test\_score | split3 test\_score | split4 test\_score | Mean test\_score | Std test\_score | Rank test\_score |
| 1e-06 | 0.798613 | 0.799888 | 0.806720 | 0.799268 | 0.792926 | 0.799483 | 0.004390 | 7 |
| 1e-05 | 0.801216 | 0.800223 | 0.806648 | 0.800000 | 0.793444 | 0.800306 | 0.004201 | 5 |
| 0.0001 | 0.801376 | 0.802611 | 0.806969 | 0.803018 | 0.795628 | 0.801920 | 0.003662 | 4 |
| 0.001 | 0.803232 | 0.804101 | 0.807703 | 0.805343 | 0.797133 | 0.803502 | 0.003522 | 3 |
| 0.01 | 0.803978 | 0.803911 | 0.809517 | 0.806751 | 0.799122 | 0.804656 | 0.003454 | 2 |
| 0.1 | 0.805142 | 0.804160 | 0.810803 | 0.806748 | 0.801199 | 0.805611 | 0.003164 | 1 |
| 0 | 0.792121 | 0.797977 | 0.802856 | 0.794459 | 0.788440 | 0.795171 | 0.004941 | 8 |
| 1 | 0.799836 | 0.798780 | 0.806465 | 0.802266 | 0.792183 | 0.799906 | 0.004680 | 6 |
| 2 | 0.794563 | 0.793830 | 0.801458 | 0.792688 | 0.789985 | 0.794505 | 0.003809 | 9 |
| 10 | 0.697751 | 0.708646 | 0.706272 | 0.693624 | 0.698022 | 0.700863 | 0.005656 | 10 |

### 2.2 Logistic Regression

**In Logistic regression with C=2 we got, (Best we got from trial and error)**

Topic\_1 accuracy= 0.8461111111111111

Topic\_2 accuracy= 0.9319444444444445

Topic\_3 accuracy= 0.8922222222222222

Topic\_4 accuracy= 0.87

Topic\_5 accuracy= 0.9722222222222222

Topic\_6 accuracy= 0.9888888888888889

Train Time: 45.056 sec

F1-micro: 0.7932742139813429

(Grid search CV caused the multiple breakdown of machine)

### 2.3 Voting Classifier : FINAL MODEL

As we can see both Multinomial Naive bayes and logistic regression is doing good job. Now both are ensembled to get the best out of it.  
  
**f1 micro score**: **0.8017968664402322**

Topic\_1 accuracy= 0.8441666666666666

Topic\_2 accuracy= 0.9294444444444444

Topic\_3 accuracy= 0.8952777777777777

Topic\_4 accuracy= 0.8638888888888889

Topic\_5 accuracy= 0.9747222222222223

Topic\_6 accuracy= 0.99

Train time: 43.372 sec

**Params –**

OneVsRestClassifier(estimator=VotingClassifier(

estimators=

[('mnb',OneVsRestClassifier(estimator=MultinomialNB(alpha=0.1),n\_jobs=-1)),

('lr',OneVsRestClassifier(estimator=LogisticRegression(C=2), n\_jobs=-1))],

n\_jobs=-1, voting='soft', weights=[2, 1]), n\_jobs=-1)

### 2.4 Stacking Classifier

Stacking is same as voting but here we have meta classifier as logistic regression, which takes output of all stacked classifier and predict the final output. This is also working with good accuracy but takes more time than voting classifier.

**f1 micro score: 0.7992277992277993**

Topic\_1 accuracy= 0.8483333333333334

Topic\_2 accuracy= 0.9336111111111111

Topic\_3 accuracy= 0.8936111111111111

Topic\_4 accuracy= 0.8680555555555556

Topic\_5 accuracy= 0.975

Topic\_6 accuracy= 0.9902777777777778

Train time: 232.105 sec

## Conclusion

We choose voting classifier for less training time, high f1-micro score, high topic wise accuracy.

This model is saved in ***'../output\_files/voting\_model.sav’***

# Appendix

## Screenshots:

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated with medium confidence

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Score of voting classifier model