DARPG: HACKATHON

Authors: Ankita Bhatia and Siddharth Singh

Contents

[Category: 1](#_Toc29074261)

[What is the problem we are targeting? 1](#_Toc29074262)

[Solution: 1](#_Toc29074263)

[Details of components: 2](#_Toc29074264)

[Exploratory Data analysis (EDA) 5](#_Toc29074265)

[Model building 7](#_Toc29074266)

[Models used 7](#_Toc29074267)

[Random forest (with Cross validation): 7](#_Toc29074268)

[SVM (with Cross Validation): 7](#_Toc29074269)

[Naïve Bayes: 7](#_Toc29074270)

[The metrics that we have chosen are: 8](#_Toc29074271)

[F1-Score: 8](#_Toc29074272)

[ROC-AUC score 9](#_Toc29074273)

[Challenges faced 11](#_Toc29074274)

# Category:

Making the redressal process more robust and data-driven to reduce the Grievance submission and resolution lifecycle. Technology such as AI and ML could be used.

# What is the problem we are targeting?

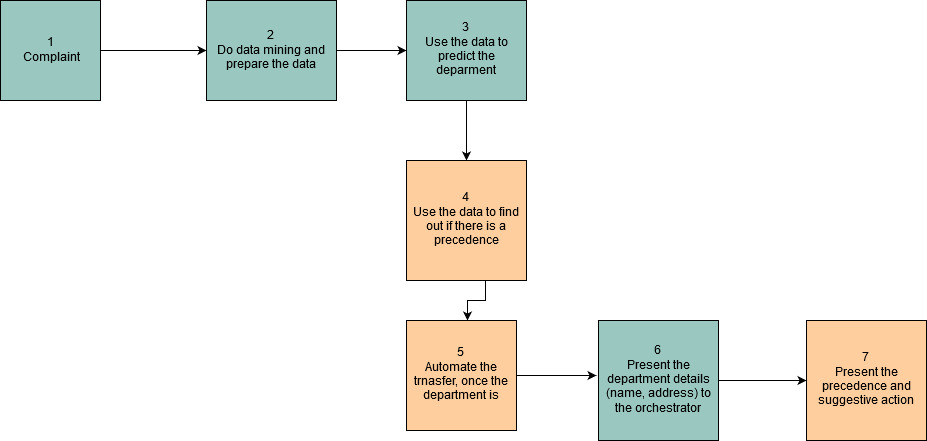
1. One of the issue that we see is the transfer of complaint between multiple departments before being addressed. This takes a lot of time for the complaint to reach the right department/nodal officer to address it.
2. Other issues we see are complaints sent back for more evidence or complaints disposed off, without taking any action.
3. We also think that if nodal officers have information of previous remedial actions taken on similar complaints readily available, she/he might be able to take faster and affirmative action..

We are focusing on problem 1 as the MVP. Problem 2 and 3 will be treated as a stretch goal.

# Solution:

For MVP: We will be using the complaint raised from the consumers of <https://pgportal.gov.in/> to do the data mining and understand how can we determine the probable rightful department which should handle this request. The ultimate goal will be to automate the transfer, without manual intervention.

The high-level workflow is depicted in below diagram:



**Figure 1: End to End workflow**

Legend:

: Stretch goal

: MVP

# Details of components:

1. This the raw record as filed by the customer. This is present in file: **“Public Grievance details in CPGRAMS along with feedback details”**. The basic structure of the record is given below:

{

"registration\_no": "AYUSH\/E\/2019\/00300 ",

"ministry\_department": "Ministry of Ayush",

"country\_name": "India",

"state\_name": "Delhi",

"distname": "South Delhi",

"subject\_content": "Sir The ESIC dispensary in Jangpura new delhi is not working properly. The doctors not treated to patients they always chat with other staff for hours and hours. Sir please do something so that we can save our time please.",

"diarydate": "03-05-2019 12:37",

"closing\_date": "06-06-2019",

"SourceName": "Local\/Internet",

"rating": "N",

"comments": "NA",

"ratingdate": "NA"

},

We will be using country\_name, state\_name and subject content feature from the above complaint record, to do the prediction.

1. Data mining and preparation step: There are 2 more files that we use here:
   1. **NodalOfficer\_Details.csv**: This catalog contains details of Nodal Public Grievance Officers of all the departments covered under CPGRAMS.
   2. **Public Grievance movement details across the organization in CPGRAMS** another file which contains the trace of the complaint movements across the departments till it is either addressed or disposed off. The records in that file look like this:

{

"registration\_no": "AYUSH\/E\/2019\/00300",

"action\_srno": "1",

"action\_name": "RECEIVED THE GRIEVANCE",

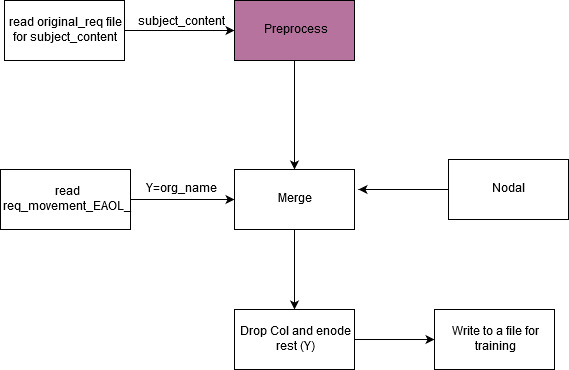
"date\_of\_action": "07-05-2019",

"org\_name": "COMPLAINANT",

"org\_name2": "Ministry of Ayush",

"remarks": "NA"

}

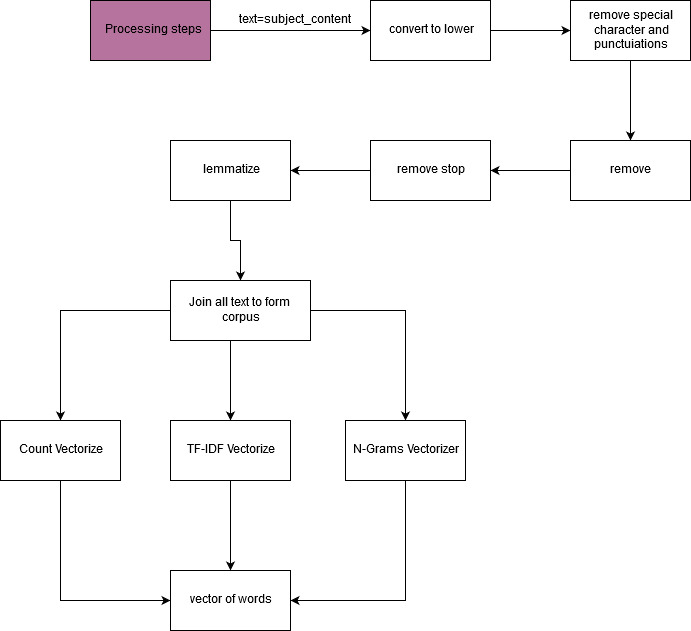


**Figure 2: Preparation of training data from different files**

The steps involved in data mining and preprocessing are:

1. From the complaints file (mentioned in 1) extract the “subject\_content” and apply following steps to the raw text to cleanse:
   1. Convert to lower case
   2. Remove special character, whitespaces, punctuations, numbers
   3. Remove stop-words
   4. Remove words (characters) which are less than 2 characters in length. This is done to remove errors in inputs.
   5. Choose unique words and convert them to string? What is this?
2. Vectorize the cleaned text. We are using following algorithms for vectorizing the data:
   1. **CountVectorizer**: Convert a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using scipy.sparse.csr\_matrix.
   2. **TF-IDF:** Term Frequency–Inverse Document Frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
   3. **n-Gams:** Type of probabilistic language model for predicting the next item in such a sequence in the form of a (n − 1)–order Markov model.
3. **Prediction step - Department:** Query the complaint filed data from the API and pass it to department prediction model (after preprocessing) for prediction. The department here means the **Parent Org Name** taken from the NodalOfficers.csv file. Please look into Exploratory Data Analysis section for details on why did we chose this.
4. **Prediction step - Suggestive action:** Query the complaint filed data from the API and pass it to action prediction model (after preprocessing) for prediction.
5. **Automate the transfer of complaint to department**: This is a stretch goal. For hackathon it is presented as a suggestion, to automate the transfer process without any human involvement.
6. **Present department details:** Once the predictions for departments are available, present the suggestion to the orchestrator. We use the raw text from the complaint to predict the department. We also use the city and state from the input data and present it along with the predicted department to give more details about the target department.
7. **Present suggestive action:** Once the predictions for possible actions are available, present the suggestion to the Nodal officer.

The figure below depicts the preprocessing steps:



**Figure 3: Data pre-processing details**

# Exploratory Data analysis (EDA)

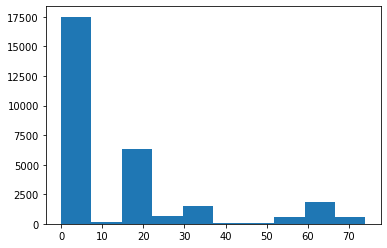
1. Total number of requests available in the dataset: from the **Public Grievance details in CPGRAMS along with feedback details,** we queried 2,60,000 records. From **Public Grievance movement details across the organization in CPGRAMS** we have queried 3,10,000 records.
2. For asserting the right department we assume that the department where the **action\_name** is equal to: *'INTERIM REPLY TO COMPLAINANT','CASE DISPOSED OF','EXAMINED AT OUR LEVEL'*, is the department where actual action was taken and we want that the complaint lands directly in this department. We are using this department as our label.

Next, we take the registration ID from the file, where action\_name is any of above and check what is org\_name.

We take this org\_name and search it in **NodalOfficers.csv** and map it to *'Apex Ministry/Dept/State'* and use it as labels. We could not use any further granular departments as we could not find any mapping *between org\_code, org\_name, Parent Org name* and *Apex Ministry/Dept/State.*

After analyzing all the above data, we find ~29000 records which can be used for analysis and model training.

1. We observed that the data obtained above is highly imbalanced. i.e. most of the requests (60%) were falling in same class (or department). Also there were many classes which had very little( <30 ), insignificant number of requests. To reduce the effect of this imbalance we did 2 things:
   1. Removed 75000 rows from the dominant class
   2. For all the classes having less than 30 requests, we have combined them as others.



**Figure 4: Class (or department distribution)**

1. Details of final training data set.

|  |  |
| --- | --- |
| **Data Set** | **No. of Records** |
| Unique registration IDs in original data | 222129 |
| Registration IDs available in movement file | 32890 |
| Unique registration IDs where selected action\_name available | 29365 |
| data size after removing 7500 rows from dominant class | 21865 |
| **Final training set after removing 3000 (random) rows for blind test** | **18865** |

1. **Complaint resolution velocity**: Velocity of complaint resolution is the time taken, in number of days. We calculate few variants of velocity

From **Dept\_stat\_receipt\_disposal\_010112019.csv** file we find the below data regarding the movement of requests. Please note total disposal may not mean all the reports were disposed after taking action, as we see that some reports are disposed without taking any action.

|  |  |  |
| --- | --- | --- |
| **Receipts** | **Sum** | **% of Total** |
| Total Receipts (01.01.2016 to 01.11.2019) | 4779587 |  |
| Total Disposal (01.01.2016 to 01.11.2019) | 3798755 | 79.48 |
| Total Pending as on 01.11.2019 | 980832 | 20.52 |
| Pending More Than 1 Year | 599572 | 12.54 |
| Pending Between 6 To 12 Months | 100096 | 2.09 |
| Pending Between 2 To 6 Months | 118237 | 2.47 |
| Pending Less Than 2 Months | 162927 | 3.41 |

# Model building

## Models used

Once the data is preprocessed as described in the section above, we use this data to train our models. We have used 3 different models her:

### Random forest (with Cross validation):

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

### SVM (with Cross Validation):

Support-Vector Machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

### Naïve Bayes:

Naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models. Naïve Bayes is a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

We train all these models with same data and then do the comparative analysis of the metrics generated.

## The metrics that we have chosen are:

### F1-Score:

**F1-Score** is a metric to evaluate predictors performance using the formula.

**F1 = 2 \* (precision \* recall) / (precision + recall)**

When you have a multiclass setting, the *average* parameter in the f1\_score [function](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html) needs to be one of these:

* ***'weighted'***
* ***'micro'***
* ***'macro'***

1. The first one, **'weighted'** calculates de F1 score for each class independently but when it adds them together uses a weight that depends on the number of true labels of each class:

**F1class1∗W1+F1class2∗W2+⋅⋅⋅+F1classN∗WNF1class1∗W1+F1class2∗W2+⋅⋅⋅+F1classN∗WN**

therefore, favouring the majority class.

1. **'micro'** uses the global number of TP, FN, FP and calculates the F1 directly:

**F1class1+class2+class3F1class1+class2+class3**

not favouring any class in particular.

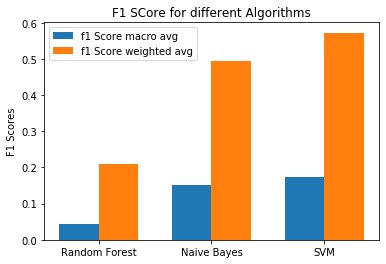
1. Finally, **'macro'** calculates the F1 separated by class but not using weights for the aggregation:

**F1class1+F1class2+⋅⋅⋅+F1classNF1class1+F1class2+⋅⋅⋅+F1classN**

which results in a bigger penalization when your model does not perform well with the minority classes.

#### For our model comparison we use:

1. F1 score (macro average):
2. F1 Score (weighted average):



**Figure 5: F1 Scores comparison**

### ROC-AUC score

**ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

* True Positive Rate
* False Positive Rate

**True Positive Rate** (**TPR**) is a synonym for recall and is therefore defined as follows:

TPR=TP/(TP+FN)

**False Positive Rate** (**FPR**) is defined as follows:

FPR=FP/(FP+TN)

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

**AUC** stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve.

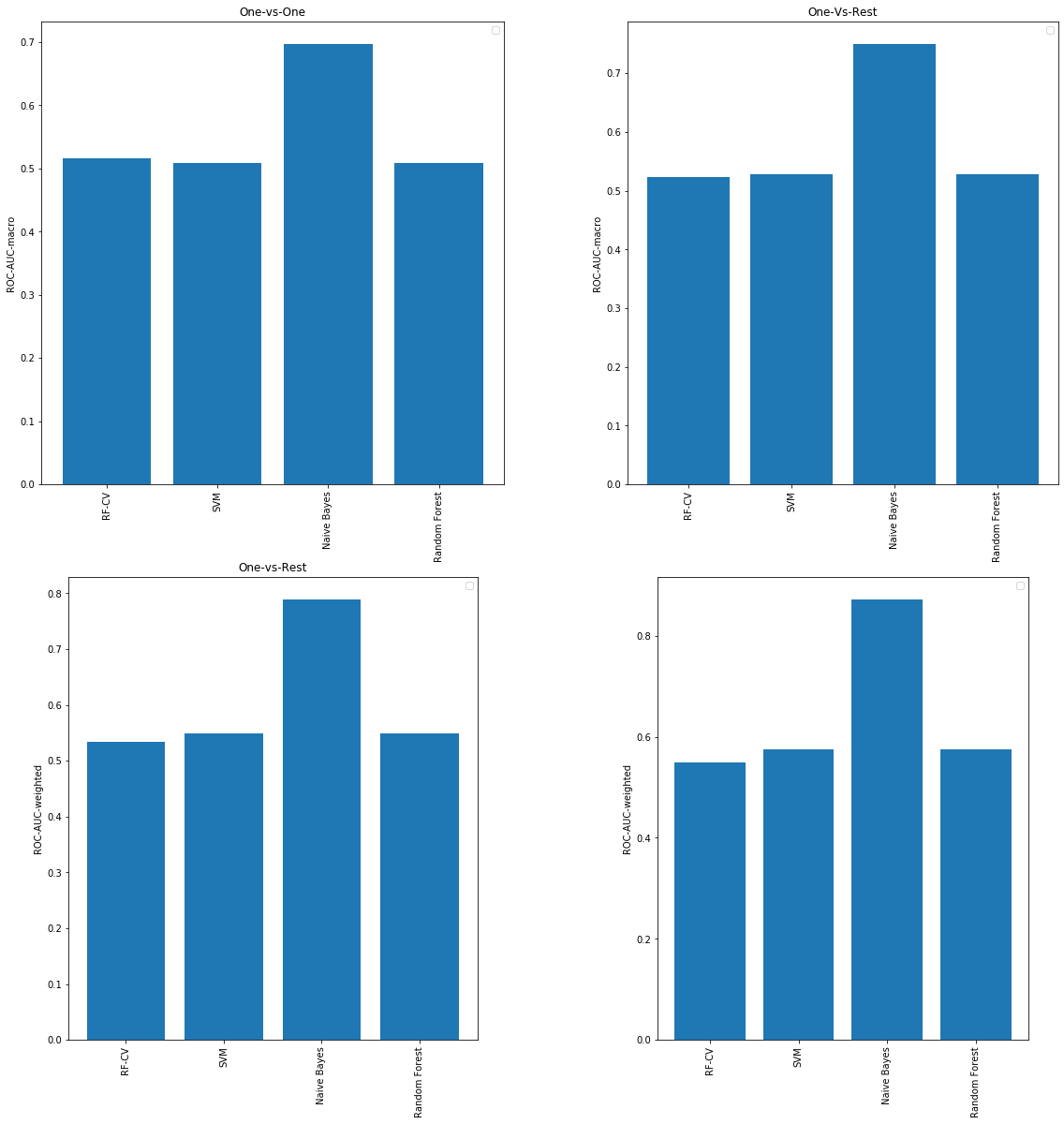
AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

#### One-verses-rest (ovr) and one verses-one (ovo):

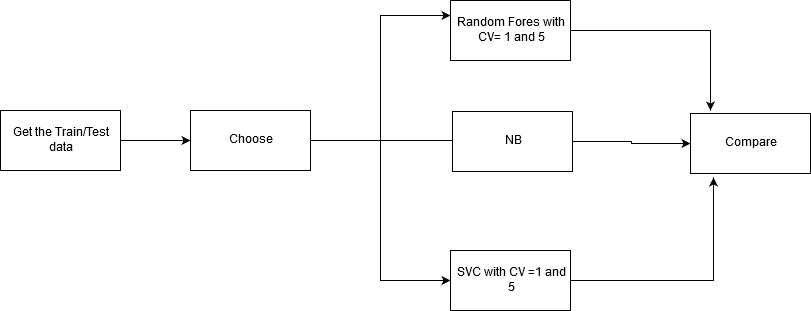
* **ovr**: Computes the AUC of each class against the rest. This treats the multiclass case in the same way as the multilabel case. Sensitive to class imbalance even when average == 'macro', because class imbalance affects the composition of each of the ‘rest’ groupings.
* **ovo:** Computes the average AUC of all possible pairwise combinations of classes. Insensitive to class imbalance when average == 'macro'.

#### For our model comparison we use:

1. ROC\_AUC (ovo, macro):
2. ROC\_AUC (ovo, weighted):
3. ROC\_AUC (ovr, macro):
4. ROC\_AUC (ovr, weighted):



**Figure 6: AUC-ROC comparison for different models**



**Figure 7: Model training**

# Challenges faced

1. **Subject\_content in too many languages:** The free text entered by the user is multiple languages. People have also typed Indic languages in English. This makes the text vectorization very difficult and the vectors obtained are extremely sparse.
2. Text contains spelling mistakes and many different abbreviations used. Though we had implemented a spellcheck module for identifying the spelling errors, but because of too many mistakes and abbreviations, it was making the free text worse. This was happening as the spellcheck was trying to map the abbreviations to English words. Also, we wanted to retain abbreviations for departments ( e.g. aiims) and use it as a feature. Hence, we decided to drop it.
3. To many departments and need information on hierarchy of departments for accurate predictions. While analyzing the data, we were thinking of using org\_code from NodalOFficers.csv file. But org\_code, org\_name and Parent of Organisation, all had too many values, making the number of classes to predict, too large. Also, We could not establish a hierarchy of departments.
4. The data is heavily imbalanced. We observed that the data obtained above is highly imbalanced. i.e. most of the requests (60%) were falling in same class (or department). Also there were many classes which had very little ( <30 ), insignificant number of requests.