DARPG: HACKATHON

Authors: Ankita Bhatia and Siddharth Singh

Contents

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 right department which should handle this request, and present it to the orchestrator. The final goal will be to automate the transfer, without manual intervention.

Problem Discovery

We did following research to understand the issues of the user and arrive at the solution proposed

1. **Customer discovery:** We went through a lot of customer reviews/Quora discussion threads, for the pgportal site (and for the app also).
   The top problems which surfaced were:
   a. Issues with logging in to the site and app
   b. Time taken for the redressal.
   c. Notification/updates to the user.
2. **Personal experience:** We also consulted the friends and family to understand what was their experience of using the site. It was consistent with the above observations
3. **Data Analysis**: Details of the analysis done on the data provided by the DARPG team, is given further in the document.

The high-level workflow is depicted in below diagram:

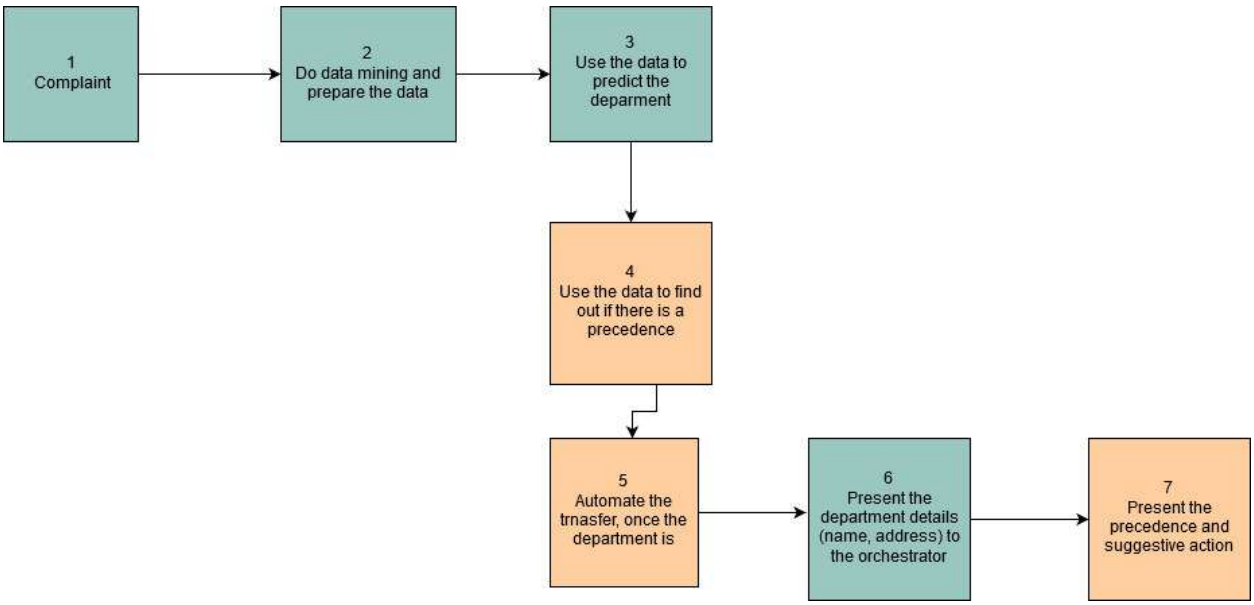

Figure 1: End to End workflow

Legend:

: Stretch goal

: MVP

Details of components:

1. The data for the complaint filed by the user is obtained by one of the APIs provided to us: **"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 features from the above complaint record, to do the prediction.

2. Data mining and preparation step: There are 2 files that we use here:
    a. **NodalOfficer_Details.csv**: This file contains a catalog of Nodal Public Grievance Officers of all the departments covered under CPGRAMS.
    b. **Public Grievance movement details across the organization in CPGRAMS**: API which gives us the data which contains the trace of the complaint movements across the departments till it is either addressed or disposed of. The records present here 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"
 }
```


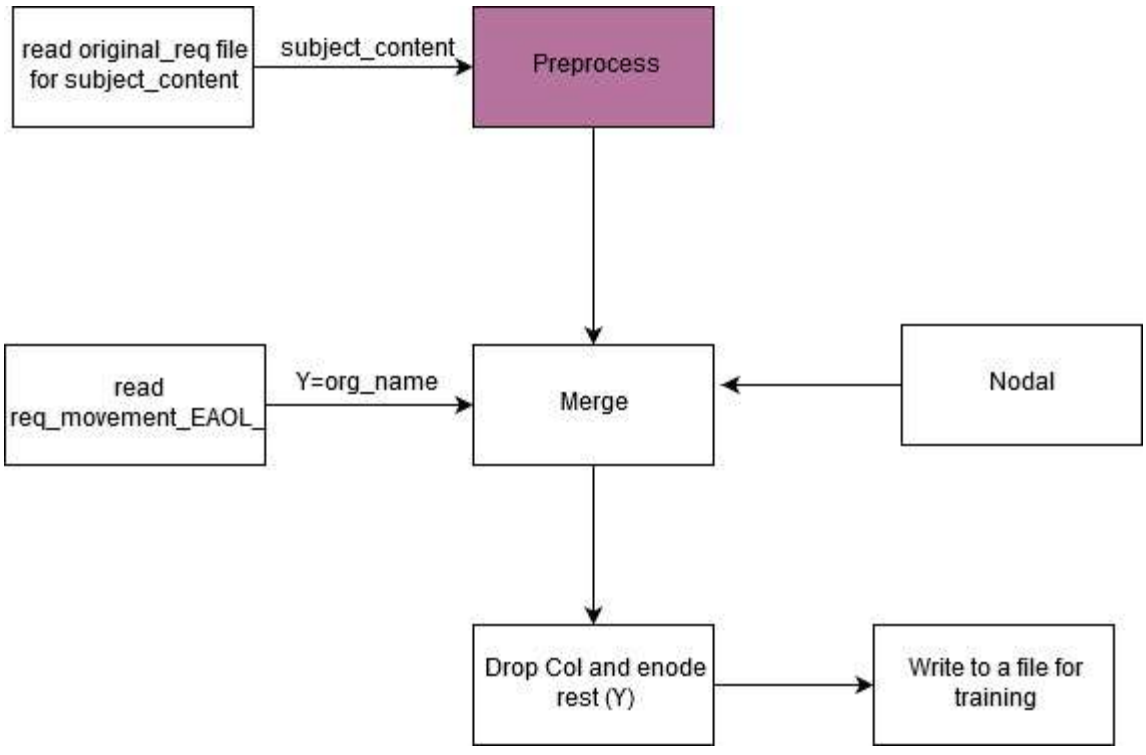

Figure 2: Preparation of training data from different files

The steps involved in data mining and preprocessing are:

I. From the complaints file (mentioned in 1) extract the "`subject_content`" and apply following steps to the raw text to cleanse:
    a. Convert to lower case
    b. Remove special character, whitespaces, punctuations, numbers
    c. Remove stop-words
    d. Remove words (characters) which are less than 2 characters in length. This is done to remove errors in inputs.

II. Vectorize the cleaned text. We are using following algorithms for vectorizing the data:
    a. **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.
    b. **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.
    c. **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 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. The Exploratory Data Analysis section covers the details on why did we chose this.
4. **Prediction step - Suggestive action:** Query the complaint data from the API and pass it to action prediction model (after preprocessing) for prediction. The model (in future) will predict the suggestive action to be taken on the complaint, based on historical data.
5. **Automate the transfer of complaint to department**: This is a stretch goal. For hackathon it is presented as a suggestion. The goal is 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.
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:

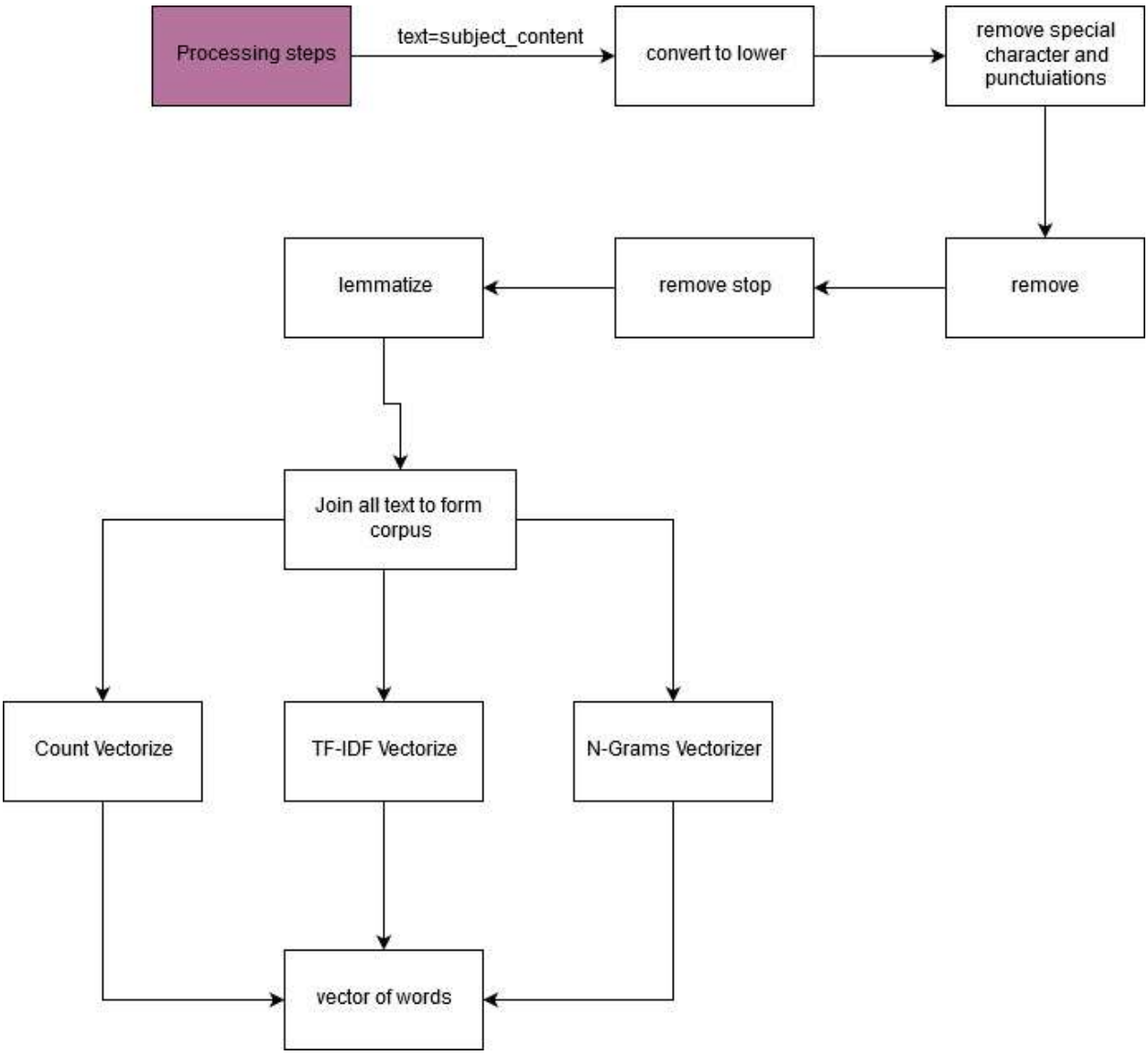

Figure 3: Data pre-processing details

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 determining 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. We did the study on multiple action names and the corresponding remarks to arrive at this conclusion

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.

3. 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:
    a. Removed 75000 rows from the dominant class
    b. For all the classes having less than 30 requests, we have combined them as others.

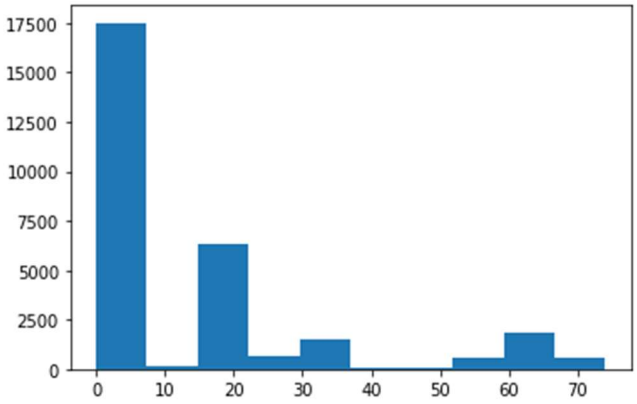

Figure 4: Class (or department distribution)

4. 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**

5. **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.

$$\mathbf{F1 = 2 * (precision * recall) / (precision + recall)}$$

When you have a multiclass setting, the *average* parameter in the f1_score function needs to be one of these:

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

a. 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:

$$\mathbf{F1class1 * W1 + F1class2 * W2 + \cdots + F1classN * WNF1class1 * W1 + F1class2 * W2 + \cdots + F1classN * WN}$$

therefore, favouring the majority class.

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

$$\mathbf{F1class1+class2+class3F1class1+class2+class3}$$

not favouring any class in particular.

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

$$\mathbf{F1class1+F1class2+\cdots+F1classNF1class1+F1class2+\cdots+F1classN}$$

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):

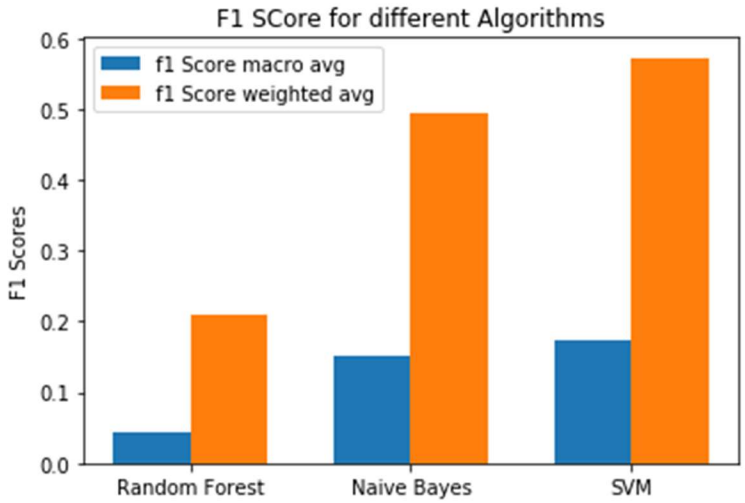

Figure 5: F1 Scores comparison

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):

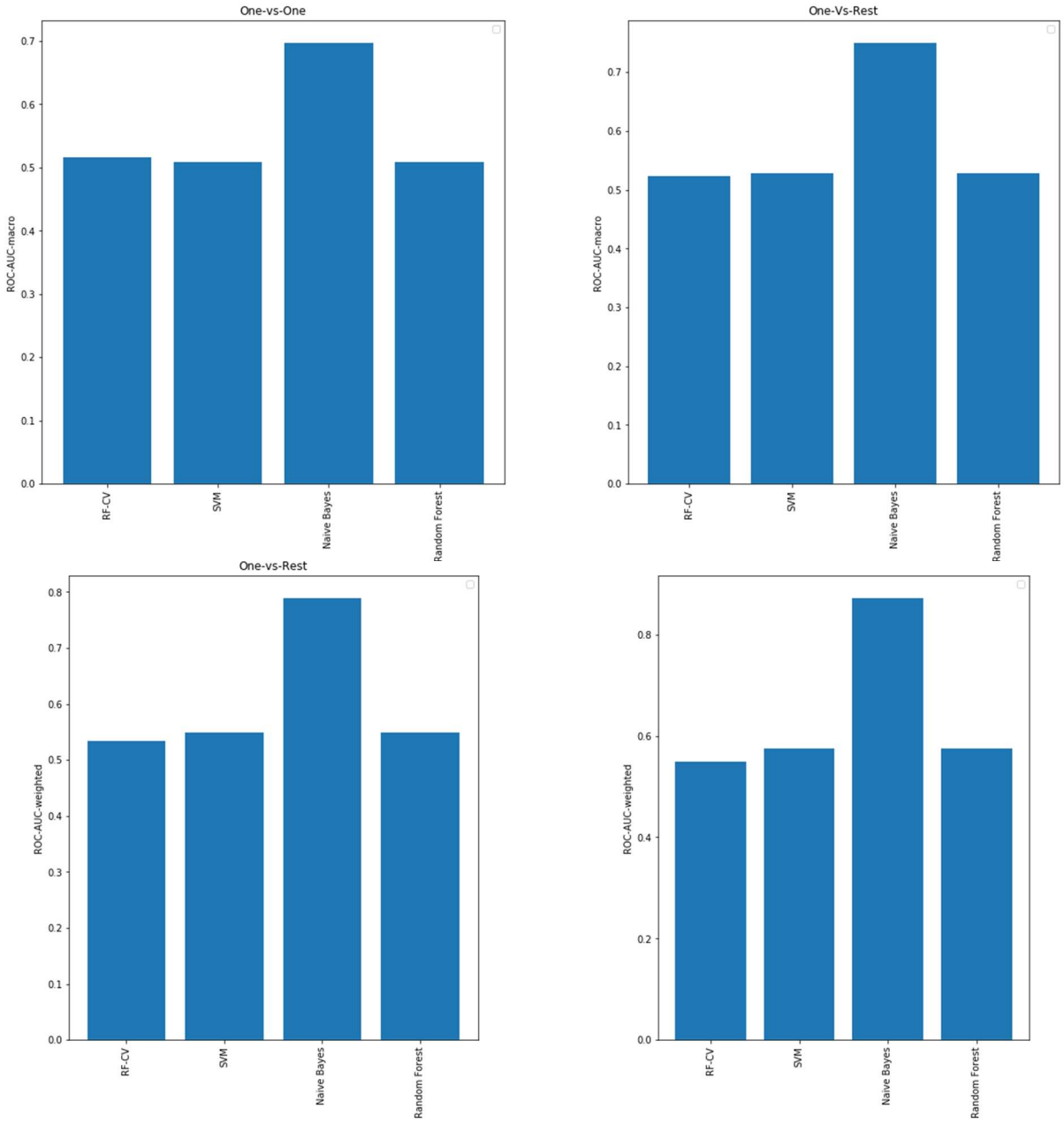

Figure 6: AUC-ROC comparison for different models

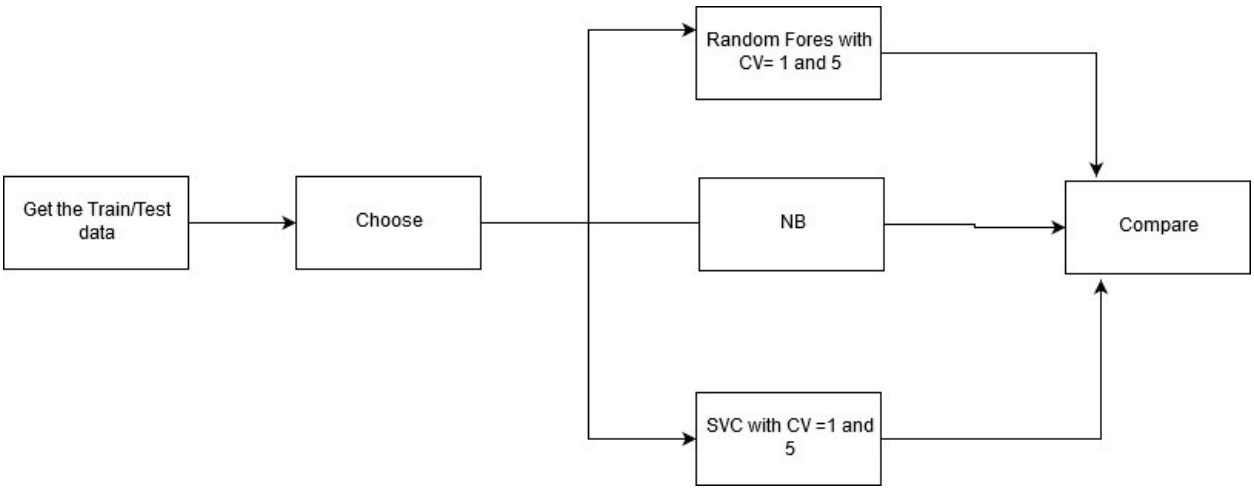

Figure 7: Model training

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.