Building accurate text classifiers

Arthi Venkataraman
Wipro Technologies
72, Electronics City, Bangalore
arthi.venkat@wipro.com

ABSTRACT

This paper brings out the various techniques we have followed to build a production ready scalable classifier system to classify the tickets raised by employees of an organization. The end users raise the tickets in Natural language which is then automatically classified by the classifier. This is a practical applied research paper in the area of machine learning. We have applied different machine learning techniques like active learning for improving the accuracy of the prediction and have used clustering for handling the data issues found in the training data. The approach we used for the core classifier combined the results of multiple machine learning algorithms using suitable scoring techniques. The overall solution architecture used ensured the meeting of any production grade software system attributes of reliability, availability and scalability.

Use of the system has given more than 50% improvement in the tickets re-assignment index and more than 80% accuracy has been achieved in correctly identifying the classes for the tickets. The system is able to perform at scale, has response times well within the expectations and handles the peak load.

Key takeaways from this paper include:

- How to build live production ready classifier system
- How to overcome the data related challenges while building such a system
- Solution architecture for the classifier system
- Deployment architecture for the classifier system
- Being prepared for the kind of post deployment challenges one can face for such a system

Benefits of building such a system include Improved Productivity, improved End user experience and quick turnaround time.

KEYWORDS

Natural Language Processing, Machine Learning, Active Learning, Classification, Clustering, Name Entity Recognition, Committee-based approach, Cluster then Label, Shallow semantic parsing, Ontology, Cognitive Process Automation, Partial Training, TF-IDF (Term Frequency Inverse Document Frequency)

1 INTRODUCTION

Across the world there is a significant pressure to drive costs down. Employees want better working conditions and better pay while at same time customers want better service and quicker responses. One answer to this is Cognitive Process Automation (CPA). CPA can be used to automate different business processes so that they can be performed consistently and quickly with reduced manual effort.

Employees face a lot of issues across different functions in the organization like Payroll, Infrastructure, Facilities, Applications, etc. When employees raise a ticket they are traditionally asked to manually select a category for their ticket from a hierarchical menu driven interface. This leads to a lot of wrong selection of choice by end user. This in turn causes tickets to be raised in wrong bucket and delays the resolutions due to re-assignments.

We have built a system which accepts a Natural language text input from end user and

DOI: 10.17781/P001921
Published in Research Gate

automatically classifies the ticket in the right category. The technical steps needed to build such a system include cleaning the data, natural language processing, machine learning both supervised and un-supervised, working with large data and continuous learning. In addition system should work in near real time.
This paper brings out the challenges we faced, the techniques we used to overcome these challenges, results obtained and some of the future work we want to do. This paper falls in the category of Industry applied research. We have researched the different techniques for our problem areas. Techniques were customized and improvised for our use case to build the system. Key focus areas of this paper are the Data Cleaning and Augmentation, Building the Classifier and Deployment.

2 BACKGROUND

Incidents in the organization are logged using different incident logging mechanisms. All of them have a traditional User interface which users navigate based on their understanding of the problem statement. There was no natural language interface. There were many in-accuracies in the incident logging process since it was dependent on the users understanding of the issue and whether they chose the right system, right screen and right item in screen to log the problem. Users would navigate, select the classification, enter their problem and submit. The issue with this was that users frequently did not know under which category to raise the ticket and hence raised the same in the wrong category. This leads to re-assignment of tickets where the ticket landed in the wrong bucket and hence was continuously passed around before it landed in the correct bucket to finally get resolved. There was more than 35% re-assignment index.
A huge volume of tickets are raised across functions every day. Hence a 35% re-assignment was in effect compounding the load of an already loaded support team. A system had to be designed which understands the natural language description entered by end user to automatically log the incident in the correct category.

3 DATA RELATED CHALLENGES

There were many challenges to address while building this solution.
Input Data related challenges

- Classification to happen on unstructured text (Unstructured data)
- The input data had 3000 + classes. (Data with large number of classes)
- Ticket data was unclean with unwanted texts including names, dates, etc. (Unclean Data)
- There were also a significant number of tickets assigned to wrong classes. (Wrongly Labelled data)
- Ticket data was imbalanced with some classes having large number of tickets and others having too few tickets. (Imbalanced training data)
- Large size of data which could not fit in memory together

4 DEPLOYMENT RELATED CHALLENGES

- System was to be designed for a live use of hundred thousand users
- System has to be designed for 1000+ simultaneous users
- Response has to be under .5 sec for the system to be usable
- System was to be accessible across the globe both within and outside the organization network for all the employees

We will delve deeper into how the data related challenges were handled in a dedicated section for the same. The deployment related challenges are addressed as part of the Solution approach section.

5 PRIOR RESEARCH

There is a lot of research addressing each of the challenges brought out in the previous section. Few of the key ones are highlighted in this section.
Sanjoy (2008) handles large training data with minimal label using active learning approaches. Ullman (Chapter 12) details many algorithms in the “Large Scale Machine Learning” chapter. Each of these algorithms has potential application for the current use case. Gunter (2010) discusses how classifications returned by multiple classifiers can be combined using novel boosting techniques to improve the accuracy of classification. Zhu (2009) discusses how semi supervised techniques can be used to handle combinations of labelled and un-labelled data to build classifiers. Haibo discusses how synthetic sampling techniques can be used to handle imbalanced classes.
Handling large number of classes in machine learning is a separate focused research area. Active learning has also been used frequently for handling large number of classes. Jain (2009) has found a probabilistic k-nearest neighbors technique to efficiently handle very large number of classes. Sotoris (2006) brings out the different techniques in handling imbalanced data. These cover techniques of random under sampling, oversampling, etc. There are also techniques which handle this issue at the machine learning model level itself. (C4.5 Tree, One class classification, is some of the possible algorithms.).
Classification of service tickets by correlating different data sources as well as contextual services has been done in Gargi (2014). In Julia (2015) classification of data with noisy labels has been done using two class classification and application of inter class. In Weizhang (2014) active learning of labels for training data using LSI subspace signature is discussed. In Andreas (2015) an ensemble of classifiers is used. The aim of this is to be able to classify using data which is represented in different ways. In Bruce (1996) it was found that an ensemble of classifiers performed much better than individual classifiers

6 SOLUTION OVERVIEW

Solution had to address the different challenges listed above. Figure 1 represents the main high level modules of the solution:

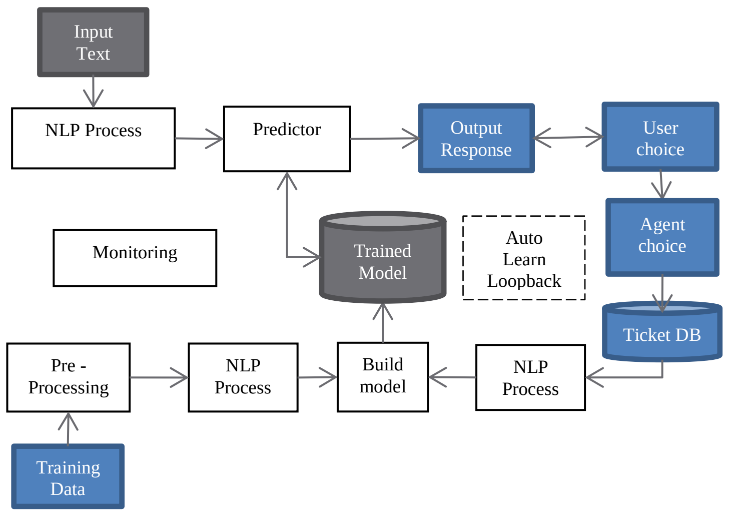

Figure 1: Solution Diagram

6.1 Pre-Processing

This step is done to ensure data is suitable for building accurate models. This is an offline activity and happens each time additional data is available to train the system. Different machine learning techniques are applied based on the understanding of the data as to what is applicable.
The data is analyzed to understand its characteristics. If it is found to have less clean training data but a larger unclean or unlabeled data set, then techniques like cluster then label as well as identification of most unclean data for manual labelling is followed. If it is found to have imbalanced classes then random oversampling techniques will be applied. If there are too many classes then cluster followed by auto labelling techniques will be applied to reduce the number of classes. All these

techniques are discussed in the Handling Data Related challenges section.

6.2 NLP Process

This block accepts unstructured text. The unstructured text is tokenized. Stop words are removed. Special named entity recognition is run to remove irrelevant words like user names and contact details. Then the important features are extracted. Features can include bi-grams / tri-grams / n-grams, presence of certain important terms, etc. A vectorization is done on features. This is followed by compression to lower dimensionality.

6.3 Build Model

This block accepts the suitably processed features from the NLP process block.

Different algorithms are available as part of the Model tool box. A one-time configuration of the set of chosen algorithms for the class of input data is done. Once the algorithms are chosen the different parameters for the algorithms is also tuned. This happens one time initially. This is repeated regularly when a significant change in data composition is foreseen. Every time a new data set arrives from the NLP processing block the models are re-built as part of this. Different models are built as output of this block.

There are two sources of input data for this block. One is the historical training data. The other is the feedback from the usage of the system which comes back from live data. The historical data goes through the Pre-processing block as well as NLP process block. The live data goes through only the NLP process block.

The composition of this module is described in detail under the Implementation approach section.

6.4 Trained model

This is the model store. The trained models are stored inside this store. The models in the store are updated whenever the build model block is triggered with new inputs.

6.5 Prediction

Every time a new text arrives it is passed through the NLP block. The output of this is then fed to the prediction module. Prediction module predicts the correct classes for the given input. It predicts using the models stored in the trained model store. The prediction model is responsible for running the different ensemble of models, combining results from different models and scoring the same. This combining and scoring can be done in different ways. It is described in detail under the Building the Classifier system.

6.6 Automatic Learning Loop Back

This module is responsible for ensuring the accuracy of the system increases with use. This is done by implementing a feedback mechanism. The usage of the choices provided by the system is tracked to provide feedback. Whenever a ticket is closed in a class which is different from the originating class it is fed back as training data for the closure class. The ticket database is monitored regularly for such cases. These tickets are then fed through the NLP process block and then to the Build model module to re-build the models.

6.7 Monitoring Process

There is a need to continuously monitor the different processes. Monitoring will be done to check if the different blocks are up and running. The response time for a prediction is tracked. If there is a degradation of performance below a certain limit suitable corrective actions will be taken. Any process which is not responsive for a pre-configured amount of time will be re-started. Suitable logging of such incidents will be done. Root cause analysis will be done for all such cases to ensure the issue will not re-occur in future.

7 TECHNIQUES FOR HANDLING DATA RELATED CHALLENGES

This section brings out how we can handle the different data related challenges. It brings out under what premise which approach will hold good.

Summarizing the different techniques

Table 1: Techniques followed to solve data challenges

ID	Data Challenge	Technique followed
1	Unclean data	Named Entity recognition tagging and filtering out the tagged data Stop word removal
2	Wrongly labelled data	Active learning
3	Large number of classes	Clustering followed by Auto Labelling
4	Imbalanced data	Random oversampling for some classes based on given accuracy.
5	Too less data	Active learning

7.1 Handling Unclean Data

Premise

- Training data is not clean with unwanted text including names, dates, and unwanted words.

Philosophy

- Good understanding of the data and what is causing it to be un-clean
- Creation of pattern recognizers / machine learning models to identify entities to tag and remove tokens causing issues
- Named Entity Recognition techniques used to identify dates and proper names. These were then filtered from the text.
- A domain specific stop word list to be created. This will be used to further filter the non-needed text. Concept graphs were extracted with focus only on the nouns. This will again be used to weed out unwanted text.

Targeted End Result

Clean corpus with all the unwanted data removed.

High Level Approach

Named Entity Recognition Approach (NER)

- Train classifier for named entity recognition
- Tag the data using the NER
- Remove the offending data
    - This processing has to be done on both the testing and training sets.

Stop Word List Approach

- A domain specific stop word list was created.
- This was used to further filter the unwanted text.

7.2 Increasing Available Clean Training corpus

Premise

- Difficult to manually get an accurately labelled complete corpus
- Have a small clean base corpus.
- Have a larger wrongly labelled / un-labelled data

Philosophy

Active Learning Approach to build a training corpus from un-labelled data.

- Use the small clean corpus to build a bigger corpus using the Active Learning approach
- Rather than clean complete corpus, find instances which are most un-clean
- Manually curate this un-clean set and label them
- Higher benefits since we are targeting tickets which are most un-clean.

Targeted End Result

To get a larger corpus for training the machine learning algorithm

Alternate High Level Approaches

- Identify the most un-clean instances for manually labelling
  - Train classifier with core set of clean tickets
  - For each unlabeled instance
  - Measure prediction uncertainty
  - Choose instances with highest uncertainty
  - Manually validate and label these instances.
- Self-Training
  - A classifier is built using smaller corpus. This is used to predict the labels of the un-labelled data
- Label using 1 nearest neighbor
  - Find the nearest labelled neighbor and label using this
- Cluster and then Label
  - Cluster the tickets
  - Build a classifier for each cluster using labelled instances in that cluster
  - Classify all un-labelled instances in that cluster.

Detailed Approach to Identify the Most Unclean Data for Re-labelling

Voting approach:

- Get the uncertainty of classification for an instance based on the disagreements amongst members of the voting group
  - Members of the voting group are the set of algorithms selected for voting
- Tickets ranked on descending order with tickets with maximum disagreement being placed at top of list
- Top x% (say 20%) of tickets will be chosen for further cleaning
- Most successfully used method
- A classifier is built using smaller corpus. This is used to predict the labels of the un-labelled data

Detailed Approach for Increasing the Training Data Set Using Cluster-then-Label

We selected a clustering algorithm A (k-means) and a supervised classifier algorithm L. (Support Vector Machine)

Input is a set of training points some of which are labelled and the rest are not.

- All the input training data points are clustered using the k-means algorithm.
- For each resulting cluster, get all the labeled instances in this cluster (Say S)
- If S is non-empty, learn a supervised predictor from S: fS = L(S).
- Apply fS to all unlabeled instances in this cluster.
- If S is empty, use the predictor f trained from all labeled data.

The output will be the labelled data based on the supervised classifier L.

7.3 Technique for Handling Wrongly Labelled Data

Premise

- There is a list of labelled training data which are known to have been wrongly labelled

Philosophy

- Use the wrongly labelled data to build a classifier to identify further wrongly labelled data in training corpus

End Result

Have a more accurately labelled corpus for training the machine learning algorithm

Detailed Approach

- Build a classifier to identify mislabeled data.
- Collect mislabeled data and use the same to train this classifier. This assumes there is some kind of pattern to the mislabeling.

- This could be used to predict other data if they are mislabeled.
- Re-label manually all mislabeled data

7.4 Technique for Handling Large Number of Classes

Premise

- Data set has too many classes
- Many of the classes were highly correlated. There is a possibility that many of these labels are very close to each other
- Is there a way closely related labels can be identified and merged automatically

Philosophy

- Group similar labels using different techniques

Targeted End Result

Have a smaller number of classes. Training data set will be grouped based on the smaller set of classes.

Different Approaches

There are many available approaches in literature. These include reducing the classes by calculating the similarity of the classes, Reduce the number of labels by finding the correlations between labels using generative models, etc. We went with the approach of clustering followed by auto labelling. We have detailed the approach in the following section.

Detailed Approach – Cluster Followed By Auto Labelling

- Cluster the entire training set
- Number of clusters will be set to the target number of classes we are looking for
- Post clustering identify the labels of original data set which fall into a cluster
- Choose the label with maximum data in a cluster as the name of the cluster
- To improve quality an audit can be done of the clustering to ensure there is indeed a logical grouping of similar labels

7.5 Technique for Handling Imbalanced Classes

The issue is that traditional classifiers seeking an accurate performance over a full range of instances are not suitable to deal with imbalanced learning tasks, since they tend to classify all the data into the majority class. We need to ensure classes with less data are also suitably represented. One positive point of the domain of our interest was typically the classes with less data are correspondingly the less important class. Hence a lower accuracy on the classes with less data can be tolerated.

Premise

- Data set has variation in the number of instances available for each class
- Some classes have lot of data and others have very less

Philosophy

- Choose a representative data set so that
    - Accuracy of classifications are good
    - Classes with lots of data do not overwhelm the classes with very data

Targeted End Result

Good classification accuracy. Results of classification are usable.

Different Approaches

There are many available approaches in literature. These include:

- Adaptive synthetic sampling to generate a balanced sample
- Oversample the classes with less data to ensure comparative data across classes
- Random under sampling of classes with more data

Detailed Approach – Oversample the Classes with Less Data

- Process the training data
- For classes with less data randomly pick data from these classes till number of

tickets in these classes in equivalent to the other classes

8 MODEL BUILDING

Once the data pre-processing is over we will come to the actual task of building the classifier models. Algorithm should have following characteristics:

- Is a supervised machine learning algorithm
- Handles text classification
- Can handle the size of the training set
    - Many algorithms will satisfy 1 and 2 above however when used for training they never completed the training cycle.
- Prediction should be in near real time

8.1 Choosing the algorithms

An iterative process is followed to shortlist the correct algorithms. Different algorithms perform well with different kinds of data. Based on benchmarking of different algorithms the best performing algorithms are selected

- Parameters for selection will vary from domain to domain
- Key Parameters which we considered in order of importance are :
    - F- Score, Precision, Re-call
        - Without good benchmarks in this system would give wrong classifications
    - Model Prediction Time
        - Near Real time system needed this
    - Model building Time
        - Certain algorithms did not complete the training cycle and hence rejected.

After every cycle the above parameters are verified. If there is a scope for improvement in these parameters then another cycle is performed till we get the required level of performance. If for any algorithm we are unable to reach the desired level of performance and there is no scope for improvement then the algorithm is rejected.

At the end of this process the set of algorithms which are relevant are selected. These algorithms are enabled as part of the Build Model block. Whenever new training data arrives it is passed through the Build Model block. The output of this block is a set of trained models which are stored in the Trained Model store.

9 PREDICTING USING THE CLASSIFIER

In live use whatever incidents are entered by user are passed through NLP Process block. The output of this is then fed to the predictor. The predictor predicts the output classes using the models in the trained model store. The key responsibilities of the predictor are

- Predict using the models in the Trained model store
- Combine the results of multiple models in the store
- Score the results
- Share the predicted results with the score to the end user

9.1 Combining and Scoring the Results Using a Committee

This section elaborates the approach followed to combine results of multiple classifiers and score the results. The models built by the shortlisted algorithms are considered members of a committee. Each of the models predicts possible classes for the given input. It is possible for a model to return more than one class also. The different unique responses can be combined. The scoring can be based on the number of occurrences of a class across models. A choice which is predicted by all models will be given maximum weightage.

It is also possible to use the individual probability scores for occurrence of a class as predicted by a model. The challenge here would be to design a way to combine probabilities given by different models.

Scoring of results can also follow a weighted combination approach. It is possible to assign more weight for choices given by certain models of the ensemble based on their historical prediction accuracy.
Reliability of the system was ensured by bringing up the accuracy of classification through use of ensembling techniques across multiple machine learning algorithms. The further score associated with each response.to ensure user can get the confidence in the response enhanced the reliability of the system.

10 SCALABLE AND AVAILABLE DEPLOYMENT

The system should be designed to handle the stated number of live simultaneous calls. It should be able to respond for prediction requests within the near real time performance constraints.
The system should be horizontally scalable. There will be multiple instances of the prediction system each residing on one machine. To understand the number of instances a benchmarking is to be done on an instance of the prediction solution to evaluate what is the peak load capacity for an instance beyond which the response time starts degrading. Based on the peak simultaneous load to be supported, the number of instances to be provisioned will be calculated. The load balancer will be configured to balance the requests across the different instances based on the performance and load on each of the instances. An auto recovery script will be created for each instance which will ensure that if the performance drops for any instance then the instance will automatically re-start.
System is to be accessible across the globe both within and outside the organization network for all employees. Hence services need to be deployed across special machines which are accessible outside the company network. Based on origin of request if request is outside the company network, load balancer will ensure that it will be serviced from these machines.
The designed solution ensured high availability since the load balancer automatically re-directed requests based on server load and uptime. No requests were sent to any server which was slow in processing requests or was down .

Please see Figure 2 for the Production deployment diagram.

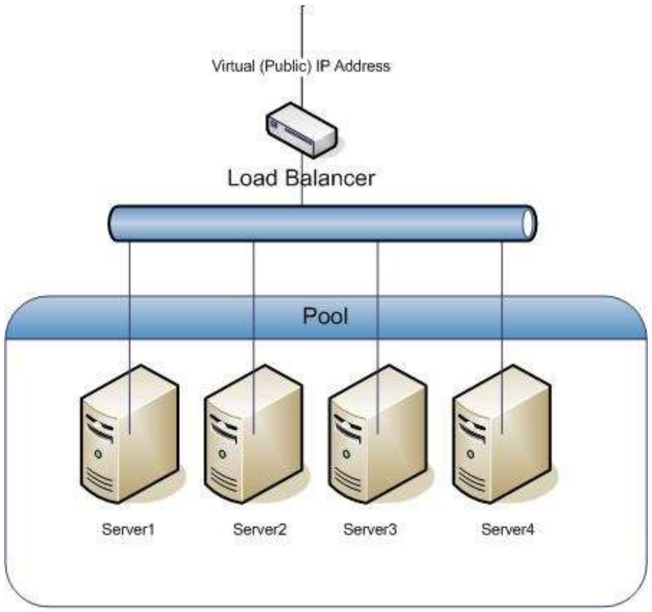

Figure 2: Production Deployment Diagram

11 PSEUDO CODE

This section brings out the pseudo code in Python for some of the key parts of solution. The calls referred to are available as part of the sklearn library in Python.

11.1 Unwanted Words Removal

Training data has lot of words which do not add value for the prediction. Examples include The, is , or, etc.
Function removeStopWords removes these words.

Accepts input parameter text which is the string from which the stop words needed to be removed
Accesses the global list myStopWordList. This is the list of stop words

```
def  removeStopWord(text) :
    text = ' '.join([word for word in
text.split() if word not in
myStopWordList])
    return text
```

11.2 Selecting training data for the classifier

- Given labelled data set has to be split between train and test sets
- Train sets will be used for training classifier
- Test sets will be used for testing classifier
- Split can be decided based on available labelled data
- Assuming it 70 : 30 where 70% of the labelled data was used for training

```
trainlen = .7 * len
Xtrain = l[0:trainlen - 1]
Ytrain = y[0:trainlen - 1]
Xtest = l[trainlen : len - 1]
Ytest = y[trainlen : len - 1]
```

- If l holds the entire train sentences and y holds the corresponding prediction values
- len is the Total data set length
- trainlen – Holds the length of the training data
- Xtrain
    - Holds the list of input data to be used for training
    - Each item of list is one sentence in training corpus
- Ytrain is a list of class labels which are in same order as the X_train data
- Xtest – Holds the list of held out set to be used for testing
- Ytest – Is the class labels for the held out test set

11.3 Vectorization

This step converts a collection of raw documents to a matrix of TF-IDF features.

Xtrain

- Holds the list of input data to be used for training
- Each item of list is one sentence in training corpus
- This sentence is post the stop word removal step

Initialize the Tfidf vectorizer.

```
vectorizer = TfidfVectorizer(….)
```

Create the vectorized output for the training data

```
X_train =
vectorizer.fit_transform(Xtrain)
```

11.4 Intelligent Feature Reduction

- Select features according to the k highest scores.
- The output of the TF-IDF vectorizer can be fed to the Chi square test call to reduce the number of feature and only retain the ones with the highest scores
- Y_train is a list of labels which are in same order as the X_train data
    - The first element in Y is the label corresponding to the first sentence in X_train

  Code snippet for this:

```
ch2 = SelectKBest(chi2, k='all')
```

  - In this case we have used the chi-squared
  - We have also opted to select all the features for chi-squared

```
X_train = ch2.fit_transform(X_train,
y_train)
```

- Using chi square test ensures only retaining most relevant features where most relevant features are those which have higher correlation with the labels.   This test will weed out non-correlated features.

11.5 Training the classifier

This step selects and trains the classifier.

- Code Snippet for classifier training:

```
clf = XXXX(param1=val1, param2=val2….)
clf.fit(X_train, y_train_text)
```

Where XXXX above refers to the relevant classifier

- We can try with the different classifiers inlcluding LogisticRegression, SVM, NaiveBayes, RandomForest, etc.
- At the end of this step clf will hold the relevant classifier trained for this data set

11.6 Predicting using the classifier

- Once classifier has been trained it can be used for predicting
- X_test – Holds the list of held out set to be used for testing
  Code flow for same:

```
x_test = vectorizer.transform(X_test)
x_test = ch2.transform(X_test)
pred = clf.predict(x_test)
```

- X_test will be passed through the same pipeline which is the vectorizer and k-best trainsform which was previously fitted with the training data
- Pred - Holds the list of predicted labels

12 HANDLING OUT OF MEMORY ISSUE WHILE TRAINING CLASSIFIERS

Out of memory issues were faced in two cases. The first case was when reading in the training data set. The second was when training the classifier. This section brings out how these were handled.

12.1 Reading a very large training set

Instead of reading the entire file contents into memory a batch wise read was done. The needed contents for a batch were read from the data set

12.2 Handling out of memory issue while training classifiers

Due to size of data many of the classifiers ran out of memory while building the models. To solve this problem we used the partial fit technique. Here instead of fitting the entire data set in one go we went for a batch wise incremental fitting of the classifier. We looped over the data set and incrementally re-trained the classifier in batches of 10000 each.

13 STAGE WISE DELIVERY

We applied the different techniques as detailed in the previous sections. Detailed below are the different stages we followed in the actual implementation of this system.

13.1 Stage 1 – Baselining by Building Initial Classifier Model

The overall requirements for the solution were clear and detailed out. We were to develop a classifier system which was to be able to accurately classify every ticket to the correct category. We received historical data dump of incident tickets with their closure category. Size of data was in range of half a million incidents and number of classes was in excess of 2500.

A classifier model has to be built which can handle text and the given scale of data. Logistic Regression was chosen for the initial classification step.

Building the classifier model

Technically the classifier model had to be able to handle unstructured textual data. To accomplish this given text description was read in. Each ticket was taken as an input point. The tickets were split into training and testing set.

We used 80% of tickets for the training and the rest were kept aside for testing. The Term frequency inverse document matrix was created for the unstructured text. This was followed by a compression step to reduce the dimensionality of the input vector. A logistic regression model was fitted for this set.

Predicting on the test set for this model gave an accuracy of less than 30%. There were classes which had no tickets in training set and only appeared in the test set due to the very less number of data points for many classes. Sample view of the classifier output is given in Figure 3.

ClassName	Precision	Recall	F1Score	Support
1	0.51	0.74	0.6	116
2	0.8	0.12	0.21	33
3	0	0	0	62
4	0.3	0.51	0.38	170
....				
Overall	0.33	0.37	0.33	90000

Figure 3: Classifier output

The initial accuracy check was well below 30%. The system was performing worse than a random flipping of a fair coin would do. We had a long way to go before the system could be useful.

13.2 Stage 2 – Data Cleanup

A detailed function wise data analysis was done. Data issues with each of the function were unearthed. This step was done in close co-ordination with the associated function teams.

Stage 2 a. Initial Data Analysis.

We used Pivot table of excel sheet to understand what is the number of classes and the split of data among the classes. This test revealed if

- Number of tickets per class is very less
- Number of classes is too large
- Data is imbalanced with few classes having too many training tickets

A View of the initial data is given figure 4.

Number of Tickets in Bucket	Number of Classes
1.00	400.00
2.00	300.00
3.00	100.00
4.00	75.00
5.00	75.00
6 to 10	300.00
11 to 50	800.00
51 - 200	480.00
201 - 400	98.00
Greater than 400	131.00
Total Classes	2759.00

Figure 4: Intial data analysis

From the above table we inferred following:

- 45% of classes had 10 tickets or less
- Less than 5% of classes had 400 tickets or more
- Only 26% of classes had more than 50 tickets

In addition we noticed

- There is one class which had around 56000 + ticket
- There are 5 classes with more than 4000 tickets
- There are 68 classes with 1000 tickets or more

Stage 2 b. Class Cleanup – Removing white spaces

A study of classes was under taken. The outcome of this study of that we noticed there are many classes with same names. The only difference was a matter of white spaces and few special characters.

For example there would classes like

- Hardware issue
- Hardware issue
- Hardware Issue
- #>>Hardware Issue

A solution for this was creating a small script which cleaned all class names. Cleaning of class names involved removing white space, removing special characters and changing all class names to the lower case. Post this data for all classes with same class names were merged. We saw ~ 10% improvement post this clean up activity.

Stage 2 c. Class Cleanup –Merging semantically similar classes

Post the previous step an audit of the class names was again done. A pattern of duplicate class names was noticed. Step 1a took care of the syntactic similarities of the class names. In this step we had to account for the semantic similarity of class names.
For example there were classes which read

- Printer issues
- Issues with printer
- Problems with printer
- Queries on printer issues

A human reader can understand that all these classes have similar intent. Having multiple classes for problems with similar intent led to the issue of same kinds of tickets being logged under the above different buckets. This led to the confusion of the classifier. To counteract this a tool was written which calculated the semantic similarity between two textual inputs. This tool was run to find the semantic similarity among different class names within a function. All semantically similar classes were renamed to a single class and tickets for all these classes were re-tagged to this new class name.
For example in above example there was a single class called Printer issues post this exercise.
We saw a further improvement of 10% in the number of classes post this exercise.

Stage 2 d. Class Cleanup – Spatial differentiation of classes with similar intent

We now turned our intention to the classes and the contents of the classes. We noticed following pattern. There were different class names for performing same functionality. The types of tickets under these classes were essentially similar. On closer analysis we uncovered that classes were created based on the resolving group. For example for the Payroll related classes there were separate teams handling the payroll queries and issues based on the region. Hence there would be following:

- India Payroll
- US Payroll
- UK Payroll
- APAC Payroll, etc.

From the classifier perspective all these classes would be similar since the ticket data usually will not have the region specific information. Hence these similar classes were bringing down the accuracy of the system. To handle this for all classes which were affected by region a region specific model was built. Based on the region the query is raised from the appropriate model was called.

Stage 2 e – Named Entity Recognition to identify noise words

To understand the data challenges a quick audit of 5% of random sample tickets was done for each of the functions data. Following categories of unwanted words were identified:

- People names
- Phone numbers
- Repetitive sentences across incidents which add no value to identify a ticket as they occur across different ticket. (For e.g. Please close this issue at earliest)

A named entity recognizer was written to identify names and phone numbers. A pattern recognizer was written to identify the repetitive non value added sentences. All these items were removed from the ticket corpus.

Stage 2 f Handling Large number of classes

As our analysis showed us we saw in excess of 2500 classes. Post the different techniques as

mentioned in previous sections we could bring this down to around 2000 classes. This was still a large number. To optimize this we explored similarity of issues raised across classes .This analysis was performed for each function. For example for the facilities management function there were close to hundred classes.
A clustering of tickets under this function was done. Many classes were found to be falling in same bucket.
For example following are details of some of the classes which were clustered together as part of each of the clusters.
Cluster 1:

- Food and Beverages
- Pantry issues - Availability of beverages

Cluster 2:

- Pest control issue - Mosquitoes / Insects
- Cleaning in Cabin / ODC / Restroom

Cluster 3:

- Office/Landline connections
- Teleconferencing

Cluster 4:

- Queries on 24/7 Project Cab booking
- Hiring a cab for late night travel / drop

Due to this exercise we could see around 15% reduction in the number of classes.

Stage 2 g – Handling imbalanced data

The problem of different classes having vastly varying number of tickets was still a major issue. Post the previous steps the number of classes was still in excess of 1600 classes.
On close analysis of data 1 class was found to have in excess of 50000 tickets. The class was machine sanitization. On auditing the ticket description for this class of tickets it was found that all tickets of this class follow the same standard description. It was a standard auto raised ticket tied to an employee's movement out of a project. Hence there was no need to have so many tickets. Hence just fifty tickets were taken for this class and the rest 45000 tickets were ignored.
An active elimination of all classes with less than fifty tickets was done. This was done by merging these classes with their nearest class. Post this for all other class with less than two hundred tickets a random over sampling was done to ensure there were a minimum of two hundred tickets across all remaining classes.
Post this exercise we can see

- A significant reduction of classes
- All classes having a minimum of two hundred tickets.

The view of the tickets versus class spread post clean up activity is given in table 2.

Table 2: Number of classes for the said number of tickets

Number of Tickets in Bucket	Number of Classes
0 - 200	251.00
201 - 400	98.00
Greater than 400	132.00
Total Classes	480.00

Figure representing the change on number of classes across stages

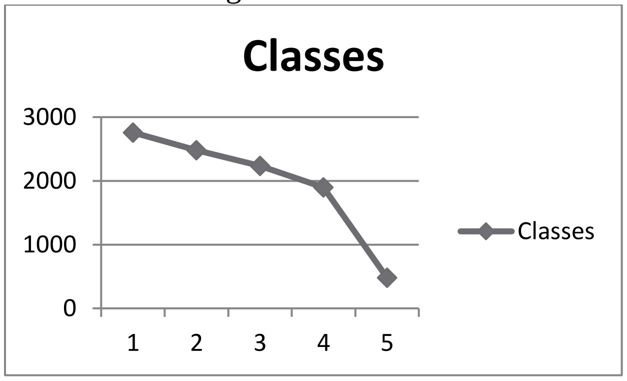

Figure 5: Number of classes across targets

13.3 Stage 3 – Building the Classifier Model

Once data cleansing step was done we focused our efforts in building a more accurate classifier. A list of feasible algorithms which were applicable to our context was short listed. Some of the algorithms which we considered include:

- Naïve Bayes
- Support Vector Machine
- Ridge Classifier
- Multi-Layer Perceptron
- K-nearest neighbors
- Random Forest classifier, etc.

Algorithm chosen should be able to accurately discriminate between close classes.

There were issues such as if user specified Printer was out of paper it had to classified to printer issues under facilities management. However if printer has hardware issues it had to be classifier under printer hardware problem categories. If there was a network issue which was in turn causing the printer not being accessible then it had to classify to the network category. If user just specified Printer issue then system would have return all related categories.

Similarly if problem stated is My XYZ application is slow then it should be placed under XYZ application. If problem stated is machine is slow them should be placed under desktop / laptop issues.

For each of the short listed algorithms a detailed analysis was done on all the listed parameters F- Score, Precision, Re-call, Model building Time and Prediction Time. We also took the precision measurements at individual function levels. Our conclusion was no one algorithm performed equally well for all functions. Also certain algorithms like the Random forest took very long time to complete the training process. Algorithms which took more time were dropped from consideration.

13.4 Stage 4 – Building the Predictor

The predictor block was developed to use the models trained by the Model Training block to predict the class using the NLP processed incident ticket. The scoring component of the predictor block combines the results of the multiple models and scores them. We faced many challenges in designing the scoring and resulting combination approach. Each of the individual models gave multiple results with probabilities. We had to combine the probability scores returned by the different algorithms. In the end we combined the unique responses across the models. The scoring was based on the number of occurrences of a class across models. A choice which is predicted by all models was given maximum weightage.

Once this combination was done, the system was tested with a hidden test set. The overall accuracy was calculated. If the accuracy was below a threshold then the training process was repeated by asking additional data from the respective function teams or by increasing the data pre-processing efforts for these classes.

This was followed by a few iterations of the following steps till acceptable accuracy was reached.

- Obtain more training data wherever possible
- Apply the Data cleaning approaches
- Perform Re-Training
- Perform Accuracy validation

This whole process was repeated till a satisfactory accuracy was obtained across the system. The system is now ready to be used for classification.

In our case the ensemble of Support Vector Machine and Ridge Classification along with some in house techniques gave maximum performance.

13.5 Stage 5 – Production Rollout

While we had the classifier running at good accuracy in isolation we had to plan for its integration with the company wide incident logging portal. At this stage we also needed to take care that the classifier will be able to handle the load of all the company users and respond in near real time.

We created REST services of the classifier interface. It was deployed in a standard Windows 7 desktop with 8 GB RAM. A front

end user interface was created as part of the employee's incident logging portal. Employees could enter their issues in free text in this user interface. On submitting the incident a call was made to our deployed REST service. The service in turn responded with the set of predicted classes with scores.
To enable the handling of the load multiple such REST services were deployed across different machines. A load balancer was used. All requests from the employee portal hit the load balancer. The load balancer automatically re-directed the request to the correct back end service based on its knowledge of the services being up and the load on each of the services. The deployment diagram followed the Figure 2. The number of instances required was calculated based on load.

13.6 Stage 6 – Continuous Learning

Post deployment there were various teething issues with the system. This was more to do with the large scale infrastructure. Once this was resolved the next issue was accuracy of the system. Post roll out the system was performing at an average accuracy of around 70% which was not acceptable. This is when the continuous learning system was brought in. A list of all tickets where final closure class was not same as initial logging class was generated. These tickets were fed back to the system for learning. For extremely poorly performing functions we took a step back and repeated the Get Additional data, Data Cleanup and Re-train step to improve the accuracies.

13.7 Stage 7 –Monitoring

Post deployment there were various issues faced by employees. These included

- Complaints that the system never gave any classification
- Complaints that the new system gave wrong classification

Due to lack of upfront monitoring we were unable to pin-point where the issue was as there were multiple points of failures. For the first kind of issue where the user complaint was that the system never gave any classification the failures could be because

- Issue with front end
- Front end was fine but on submitting it was making a call to a wrong service
- Network issue
- Issue with different backend models

For the second kind of issue where users complained that they never got back any response it could be because

- User specified the issue badly
- Users specified the issue correctly but there was a genuine problem with the classification accuracy for the user stated problem

Hence there was a strong need for monitoring. To enable monitoring we enabled Logging at different levels. At end user interface level user raised ticket description, system returned classes, Time for system to respond were logged. At the classification level the ticket description, the classes returned by the system as well as response time were logged. Between these two logs we could easily pinpoint the cause of the different issues.
A continuous monitoring dash board was built to monitor parameters at different granularity levels. These included Number of tickets raised in a day; Number of tickets re-assigned, Number of Tickets open, Function wise view of metrics, etc. A script was also written which ensured that if the performance of the system was below some threshold a restart was done.

14 RESULTS

We will track the journey we went through before we could get a reasonably stable system. There were multiple releases made before stabilization. It will cover the issues faced in each release and improvements made.

14.1 Internal Release

(Baseline Internal Release / Not released for production)

Initially when we directly used our designed classifier system the maximum accuracy which we could get was ~ 60%. In many classes the accuracy was less than 30%. Since the solution had to support classes from different functions the quality of data and challenges with respect with data was different with respect to each of the functions. A detailed function wise analysis of data was done. In parallel for every function a study was undertaken as to the best data level techniques which can be applied for that kind of data. A benchmarking was also done for the most suitable algorithm for that data. Hence the correct combination of the data processing techniques and the machine learning algorithm was selected. After multiple iterations for each of the different functions a significant improvement in accuracy was seen. In testing case we could get accuracy in excess of 80%. A sample metric for a representative sub-function is given in Table 3.

On analyzing the metrics we see that, post the different machine learning steps data has become more balanced. The number of classes has also significantly got reduced. This metric was felt good enough and we made our first production release.

14.2 First Production Release

Within a day of our release we start getting complaints about the system. When we made our release we had not planned for any logging or monitoring. There was no way for a user to give us feedback directly. Whatever feedback we got was through informal channels. These included company social groups and escalation channels where employees directly complained to the top management. Adding to the issue was when a complaint came we could not pinpoint the issue due to lack of logging.

An immediate action which we undertook was to enable logging. We also debugged issues statically. On analyzing the complaint data we could spot the trend that user outside the company network were regularly unable to reach the service. On debugging this we found that calls were being made to a service which was not accessible outside the company network. This was fixed and a release was made.

14.3 Second Production Release

At this stage we had logging enabled in the system. There was still lot of complaints which were coming to us through the informal channels.

Two top complaints related to the system were

- System does not respond
- System does not give the correct classification

On monitoring the logs we figured that system did not respond because the network was slow at times. To handle this default classes were returned to users from front end if back end did not respond within a certain time.

Further analysis of the logs brought out the challenge of wrong configuration in front end and load balancer. All calls were by default routed to only one back end service. As a result the response was slow. This issue was fixed.

For the System not giving correct classes we could narrow the issue to two cases. The first was that the end user was not entering proper information to describe their issue. They were using meaningless statements like I have an issue or generic statements like Laptop issue. To resolve this feedback was given in user interface to properly describe their problem.

There were quite a few cases where the description was correct and the system did not give a proper response. We had to figure a way to handle this.

Table 3: Pre & post data clean-up metric

For 1 subset of services	
Initial Data composition	
Training Data	100000
Number of Classes	500
Average Tickets per class	200
Percent of classes contributing to 80% of Tickets	5%
Overall Accuracy	55%
Post cleanup data composition	
Training Data	100000
Number of Classes	100
Average Tickets per class	1000
Percent of classes contributing to 80% of Tickets	75%
Overall Accuracy	80%

14.4 Third Production Release

By this time users were well educated and all the basic issues with the system were sorted out. There were still many complaints from end users that they were not getting a correct classification. However till date we did not have a way to track the accuracy of the system. We created and deployed the monitoring system. Using this we could get a view of the classification accuracy and performance of the system across the different functions. Using this metric we went back to each of the functions to collectively improve the classification accuracy of the system.

One finding of deploying the monitoring solution was that the accuracy numbers in live use was much less than the accuracy which we got in our internal testing. This could be attributed to the multiple dependencies in a live deployment scenario. There can be issues with the

- End user interface going down
- Network going down
- End user's system having issues
- End user not knowing how to describe his problem

14.5 Further Production Releases

Post the monitoring system release, multiple regular releases were made to improve the accuracy of the system. Fixes include

- Domain specific keywords to improve classification accuracy
- Additional data
- Class consolidation
- More rigorous automated and manual cleaning of the training data

14.6 Current State

Use of this system has given > 50% reduction in transfer index as compared to manual process. There is a continuous improvement in classification accuracy. At outset many classes had less than < 30% accuracy. Through the application of the different techniques as mentioned in previous section > 80% accuracy has been achieved across all areas. In some classes we have seen an accuracy of > 95%.

15 CONCLUSION

It is a daunting task to create a working prediction system with good level of accuracy and performance for a large scale live system while supporting diverse types of predictions. Text classification in real life is a challenging issue. It requires a deep understanding of the

data and kind of data cleansing algorithms which are required to handling these data types. It requires a close coordination with the different data sources to understand the data better so that the appropriate data techniques can be short listed, applied and the result verified.
To build a production ready system we will need to take into account the 3 main parameters of any software systems – Reliability, Scalability and Availability. The ensembling and scoring techniques ensured reliability of the system. The horizontally scalable architecture ensured scalability of the system. The load balancer helped in the availability of the system.

Key challenges are due to the data which brings in issues like volume, wrong mapping, insufficient data, and too many close classes. As a result a direct supervised classification approach will not work. Specific techniques would need to be used for cleaning the data, handling the wrongly mapped data, handling too many classes and handling the issue of too less data. A combination of techniques have to be used for cleaning including domain specific stop words and domain specific NER's to identify unwanted data. Clustering techniques have to be used to reduce the number of classes. Concept extraction techniques have to be used to enhance the quality of classification.
It requires a good understanding of the different possible machine learning algorithms which can be applied. The system should be designed in a flexible manner so that the appropriate algorithms can be easily applied or removed. A continuous learning of the system is required so that it can learn from use and be flexible to understand new classes which are not there in the initial training set.
A reasonably good accuracy has been obtained using the mentioned techniques and the system is being used actively by all the employees of the company. To further improve the accuracy additional techniques would need to be used including building a domain specific ontology and domain specific relationship / key concept extractor.
Building such a system has given the needed productivity improvements since the percentage of re-assigned tickets has gone down by greater than 50%. There is an improved End user experience since users no longer need to bother about navigating to the right selection and can just type their problem right away. There is a quicker turnaround time due to tickets falling to correct bucket and being allocated to the correct agent.

16 FUTURE WORK

Different approaches will have to be explored to ensure accurate classification even in face of the data issues as listed in paper. These would include semantic techniques and shallow semantic parsing to extract the relations. A domain specific ontology could be used to ensure only relevant and clean tickets will be used for training. Alternate techniques will have to be explored for the data cleaning as well as the supervised prediction approach. The self-learning loop has to be automated by finding a solution to work around users under specifying the problem and agents closing tickets under wrong closure category. There is work to be done in ensuring faster and more frequent re-training of model. There is also a need for improving the monitoring tools for this system.
From the entire system architecture perspective a further focus would be required in terms of handling even larger training data and large models. We would need to explore an automated class level splitting of models across systems. We are also exploring containers for deployment to ensure a possibility of live update of models without down time.

17. Acknowledgment

I give my heartfelt thanks to my organization Wipro who has enabled me to build this system. They have also provided the needed support to make this presentation. I would like to specially thank Mr. Ramprasad.K.R, Senior Fellow, Distinguished Member of Technical Staff, Wipro who has been my mentor and helped with the review of this paper. I would like to thank my family and specially my spouse and parents for all their support.

REFERENCES

[1] "Hierarchical Sampling for Active Learning" by Sanjoy Dasgupta, Daniel Hsu. Proceeding
ICML '08 Proceedings of the 25th international conference on Machine learning
Pages 208-215

[2] Chap 12 of book "Large Scale Machine Learning" by Ullman

[3] "New Boosting Algorithms for Classification Problems with Large Number of Classes Applied to a Handwritten Word Recognition Task" by Simon Gunter and Horst Bunke. Multiple Classifier Systems. doi:10.1007/3-540-44938-8_33

[4] Introduction to Semi-Supervised Learning by Xiaojin Zhu and Andrew B. Goldberg From the SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING #6, 2009

[5] Adaptive Synthetic Sampling Approach for Imbalanced Learning by Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li.

[6] Active Learning for Large Multi-class Problems by Prateek Jain (Univ of Texas) and Ashish Kapoor (Microsoft Research). IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2009

[7] Handling imbalanced data – A review by Sotiris Kotsiantis and et al. GESTS International Transactions on Computer Science and Engineering, Vol.30, 2006

[8] Towards Auto-remediation in Services Delivery: Context-Based Classification of Noisy and Unstructured Tickets by Gargi Dasgupta, Tapan Kumar Nayak, Arjun R. Akula, Shivali Agarwal, Shripad J. Nadgowda. Service-Oriented Computing - 12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014. Proceedings. Volume 8831 of Lecture Notes in Computer Science, pages 478-485, Springer, 2014

[9] Data Algorithms for Processing and Analysis of Unstructured Text Documents by Artem Borodkin, Evgeny Lisin, Wadim Strielkowski Applied Mathematical Sciences, Vol. 8, 2014, no. 25, 1213 - 1222

[10] Classification of Historical Notary Acts with Noisy Labels by Julia Efremova, Alejandro Montes García, Toon Calders in Advances in Information Retrieval. Lecture Notes in Computer Science Volume 9022, 2015, pp 49-54

[11] Classifying Unstructured Text Using Structured Training Instances and an Ensemble of Classifiers Andreas Lianos, Yanyan Yang. Journal of Intelligent Learning Systems and Applications, 2015, 7, 58-73 Published Online May 2015 in SciRes. http://www.scirp.org/journal/jilsa
http://dx.doi.org/10.4236/jilsa.2015.7200

[12] Feature Selection for Effective Text Classification using Semantic Information by Rahul Jain and Nitin Pise, International Journal of Computer Applications (0975 – 8887) Volume 113 – No. 10, March 2015 18

[13] Active learning for text classification: Using the LSI Subspace Signature Model. Weizhang Zhu and Allen Data Science and Advanced Analytics (DSAA), 2014 International Conference

[14] Combining Classifiers for Text Categorization. Leah.S.Larkey and W.Bruce.Croft. SIGIR '96 Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. Pages 289-297