## Smart Case Analysis

# Background

Customer chooses the high level category of service while raising a support ticket and ticket is assigned to respective queue.

Most services PS engineers queues are overloaded. Enterprise support have observed that we have cases categorised based on services, however there is a pattern with in the service for type of cases comes in each service based on the industry segment or geography.

e.g. EC2 Related

* + 1. SSH issue
    2. Security Group Related
    3. Hardware Degradation

PS engineer spends lot of time in handling these similar cases, lot of time goes in identifying the nature of problem. Each new engineer takes time to get accustomed to the solution of same problems.

Challenge is there is no mechanism to identify these common categories or groups within the services. Second part of the problem is classifying new cases in those categories.

# Objective

Idea is to identity these common groups within a top level service. This data could be categorised based on geography and industry as pattern may differ based on that.

Once we have groups are identified , we shall use that data to classify any new case coming in and possibly build a recommender system with suggested solutions for that categories.

It will provide PS engineers quick suggestion and  once the model matures we can even automate the responses using a Bot. It will definitely save time for PS engineers where we are currently  facing challenge to scale and minimise the repetitive work they doing

Finally when new case comes we should be able to classify that in one of these topics.

# SOLUTION

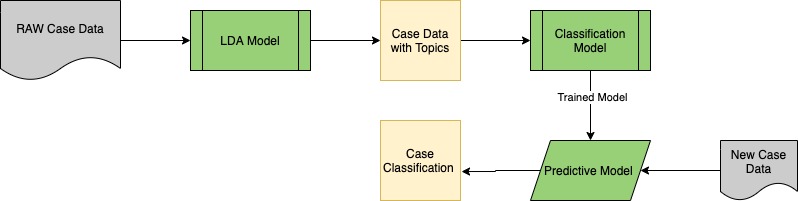
We have implemented this solution using Amazon Sagemaker. Dataset used for the implementation is a publicly available support case data .

Note: AWS support case data have customer details and would add the complexity of security and limit the data available to us.

Technology Used – Sagemaker Notebooks, Python Gensim library, Sagemaker BYOC, Keras, tensorflow, Sagemaker model deployment.

This solution is implemented in two parts first Topic Modelling and second Classification.

**Process diagram**



## Identify the Topics

First task is to identify these common pattern and identify the possible topics under each service. Since there is no insight in this pattern so this will need unsupervised LDA learning to identify these topics.

e.g.

#### Input

|  |  |  |
| --- | --- | --- |
| **Case ID** | **Case Service** | **Case body** |
| 2325445435 | EC2 | Not able to login through SSH. |
| 6876876398 | EC2 | Instance went down. |
| 8907970009 | EC2 | We not able to connect to instance after restart |
| 87809787089 | EC2 | Why the hardware degradation. |

#### Output

|  |  |  |  |
| --- | --- | --- | --- |
| **Case ID** | **Case Service** | **Case body** | **Topic** |
| 2325445435 | EC2 | Not able to login through SSH. | SSH |
| 6876876398 | EC2 | Instance went down. | Hardware |
| 8907970009 | EC2 | We not able to connect to instance from Database | Security Group |
| 87809787089 | EC2 | Why the hardware degradation. | Hardware |

### Implementation Details

#### LDA

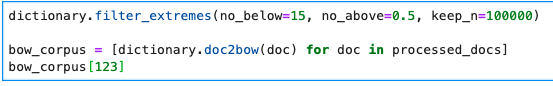
#### 

#### Steps of execution

* We picked the number of topics as 5 ahead of time.
* Document Processing
  + Each document is represented as a distribution over topics and each topic is represented as a distribution over words.
  + We used  NLTK’s Wordnet to find the meanings of words, synonyms, antonyms, and more. In addition, we use WordNetLemmatizer to get the root word.
  + We then read our dataset line by line and prepare each line for LDA and store in a list.
* First, we are creating a dictionary from the data, then convert to bag-of-words corpus and save the dictionary and corpus for future use.

dictionary = gensim.corpora.Dictionary(processed\_docs)

* Create BOW word corpus



* We then tried finding out 5 topics using LDA using genism LDA model by passing it the corpus of words , 5 topics and dictionary.



* Finally we visualise our Topics using pyLDAvis library and give topics appropriate names with the help of SME.

## 

## Classify the New Cases

Once have identified the list of Topics under each service , nest task is to build a classifier which will identify which topic new case belongs to.

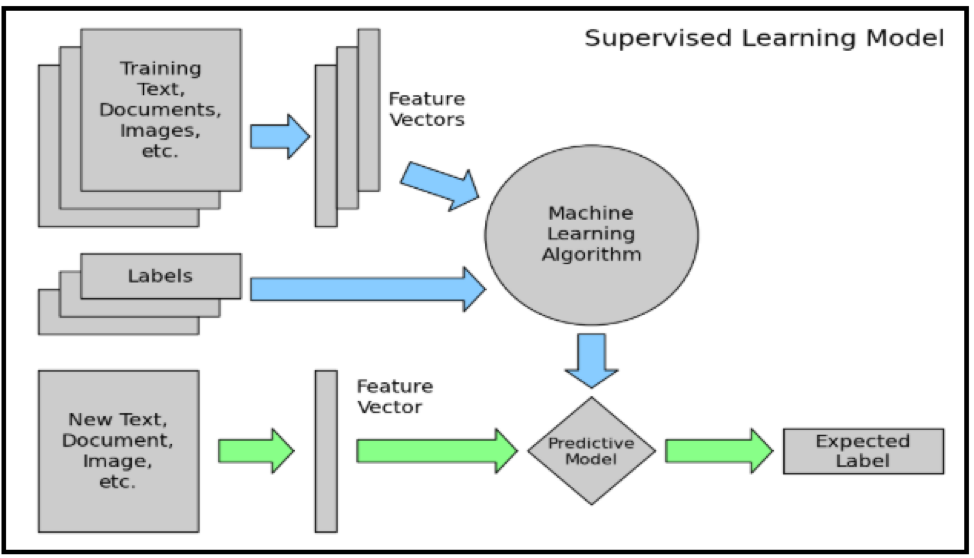
We shall use the neural network RNN to do the classification and trained using tensor flow and Keras.

We have containerised the algorithm and would be hosting and serving through Amazon Sagemaker bring your own container approach.

### Implementation Details

Deep Learning Model using RNN

1. Training text: It is the input text through which our supervised learning model is able to learn and predict the required class.
2. Feature Vector: A feature vector is a vector that contains information describing the characteristics of the input data.
3. Labels: These are the predefined categories/topics that our model will predict
4. ML Algo: It is the RNN algorithm through which our model is able to deal with text classification
5. Predictive Model: A model which is trained on the historical dataset which can perform label predictions.



#### 

#### Steps of execution

1. First step is to create the input directory structure as below and place the input files with the cases ticket data in input folder.

/opt/ml

|-- input

| |-- config

| | |-- hyperparameters.json

| | `-- resourceConfig.json

| `-- data

| `-- training

| `-- latest-ticket-data.csv

|-- model

| `-- <model files>

`-- output

`-- failure

1. In the container directory are all the components you we packaged the sample algorithm for Amazon SageMager:

|-- Dockerfile

|-- build\_and\_push.sh

`-- classify

|-- nginx.conf

|-- predictor.py

|-- serve

|-- train

`-- wsgi.py

1. Created the train.py file with simple deep learning model details
2. Tokenizer has built the dictionary of characters instead of words. This class allows to vectorize a text corpus, by turning each text into either a sequence of integers or into a vector.

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(train\_data.Description.values)

post\_seq = tokenizer.texts\_to\_sequences(train\_data.Description.values)

post\_seq\_padded = pad\_sequences(post\_seq, maxlen=MAX\_LENGTH)

<https://keras.io/api/preprocessing/text/>

1. Split the data in train and test sets.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(post\_seq\_padded, y, test\_size=0.05,random\_state=42)

1. Set the model hyperparameters for Kera Model class, set the embedding layer, dense, flatten and create the model using fit commond.

history = model.fit([X\_train], batch\_size=64, y=to\_categorical(y\_train), verbose=1, validation\_split=0.25,

shuffle=True, epochs=10, callbacks=[checkpointer])

<https://keras.io/api/models/model/>

At last epoch model accuracy improved to .71%

val improved from 0.71248 to 0.71669.

1. Finally save the modes using Keras model save.

tf.keras.models.save\_model(

model,

filepath,

overwrite=True,

include\_optimizer=True,

save\_format=None,

signatures=None,

options=None

)

1. Modified the predictor.py with below code
   1. Modified the Get method to check if model exists or not.

checker = os.listdir('/opt/ml')

health = checker is not None # health check here

status = 200 if health else 404

* 1. Modified the Post method tp pass the input with padding and load the model using kera load model.

cls.model = tf.keras.models.load\_model(model\_path)

1. Test the code locally first to see if training is working fine.
2. Once local test is successful than create the Docker image and push in in AWS ECR Service.
3. Use the image to build the SageMaker container and register the container by creating an estimator in SageMaker . Here we have used single ml.c4 instanse.

classify = sage.estimator.Estimator(image,

role, 1, 'ml.c4.2xlarge',

output\_path="s3://{}/output".format(sess.default\_bucket()),

sagemaker\_session=sess)

1. Deploy the model and create this will create the SageMaker endpoints for inference.

predictor = classify.deploy(1, 'ml.m5.2xlarge', serializer=csv\_serializer)

1. Use the endpoint to predict the topic of new cases usingthe predictor class through the endpoint we created in deploy step.

print(predictor.predict(test\_data.to\_csv()))

## Future Scope -

Once the Topics and classification is in place , we can built a Recommender Bot which can suggest the common solutions for given Topics. Initially this can recommend solution to PS engineers and once it matures we can expose it to external customers . This Bot can be used by AWS customers in self-service mode, reducing the cases and workload on PS engineer.