Table of Contents

[AWS Comprehend Analysis 2](#_Toc60489080)

[Detecting custom entities in documents 2](#_Toc60489081)

[More info 2](#_Toc60489082)

[Detecting and masking custom entities in documents 2](#_Toc60489083)

[Training the model 2](#_Toc60489084)

[Running the analysis 3](#_Toc60489085)

[Processing the analysis output using Python 3](#_Toc60489086)

[Problem with more than three entities 4](#_Toc60489087)

[Problem with entities list 4](#_Toc60489088)

[Problem with offsets 4](#_Toc60489089)

[Problem with identification 4](#_Toc60489090)

[Misspellings are recognised 4](#_Toc60489091)

[Processing the results 4](#_Toc60489092)

[APPENDIX 5](#_Toc60489093)

[ The issue of not recognising multiple entities in the same doc. 5](#_Toc60489094)

[ The issue of spelling mistakes. 5](#_Toc60489095)

# AWS Comprehend Analysis

## Detecting custom entities in documents

The built-in ‘entities detection’ works to find common entities in a document such as name, location, numbers, etc. To find entities specific to a situation, it is necessary to use ‘custom entity recognition (CER)’. It is like the ‘EntityRuler’ of spacy (<https://spacy.io/api/entityruler>) where you provide a list of words/phrases as a ‘label’ and ‘pattern’.

Unlike in spacy, here we let the AWS Comprehend to learn from the training document so that the labels are associated with context. While this provides better matching when annotated documents are used, the entity list identifies the labels without any context sensitivity.

I suppose that the internal process is the same as in NLP, so that the labels (called ‘type’ in AWS) are associated with the pattern (called ‘text’ in AWS) and added to the NLP model.

In this document the application of CER to detect pharmacy brands in a collection of document lines. This method will probably be useful in identifying profanity words/phrases in census feedback.

### More info

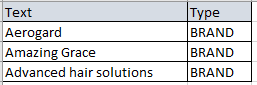
See this [tutorial](https://aws.amazon.com/blogs/machine-learning/build-a-custom-entity-recognizer-using-amazon-comprehend/) for details. For a summary of steps, relevant to own AWS account, see the document, ‘AWS Comprehend Tutorial.docx | Tutorial: Custom entities analysis’

### Detecting and masking custom entities in documents

To demonstrate the use of CER, we will be using a list of pharmaceutical products that have the brand names in the titles. To train the CER, we will use an entity list (‘brands.csv’, 1662 lines) consisting of pharmacy brand names and a training document (‘avs4.csv’, 5715 lines) that contains the product titles.

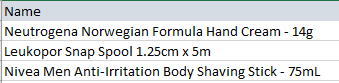
The ‘brands.csv’ has the following structure, with column headings as ‘text’ and ‘type’ (unchangeable).

**brands.csv: Table 1**



The ‘avs4.csv’ has the following structure:

**avs4.csv: Table 2**

****

### Training the model

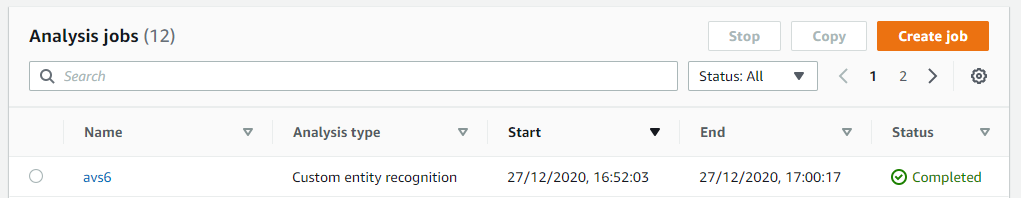
The first step in the analysis is training the model with the entity list and training doc. Quick steps are given below. For details, see ‘AWS Comprehend Tutorial.docx | Tutorial: Custom entities analysis’

* Launch [Amazon Comprehend](https://ap-southeast-2.console.aws.amazon.com/comprehend/home?region=ap-southeast-2#welcome)
* Click ‘Analysis jobs | Create job’
* Choose ‘Analysis type | Custom entity recognition’
  + This will open up a ‘Select recogniser’ dropdown plus a link to ‘[Train new recogniser](https://ap-southeast-2.console.aws.amazon.com/comprehend/home?region=ap-southeast-2#create-custom-entity-recognizer)’
* Click the link to open the trainer input page.
* Give a name for the recogniser.
* Give the ‘brands.csv’ and ‘avs4.csv’ as an S3 URIs
* Choose an IAM role.
* Click ‘Train’.
  + It takes around 21 min to complete the training with these two docs. The time may be different with other docs.
  + When finished, the new recogniser will appear in the list under ‘Select recogniser’ on the analysis page.
  + Choosing it from the dropdown will show a link to manage the recogniser, including deleting it.

### Running the analysis

Running the job is as described in ‘AWS Comprehend Tutorial.docx | Running Analysis Jobs on Documents in Amazon S3’

On completion, the job will be listed under Analysis Jobs. Click the job’s link to open and download the output.tar.gz.



Extract the output as **‘entities\_output’** (name is unchangeable) and uploaded to **S3/entities-results/** (name unchangeable).

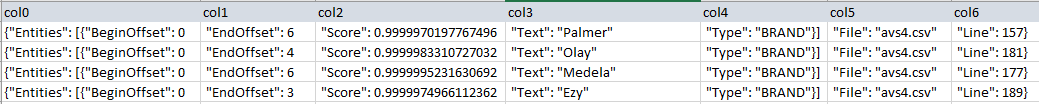
[[1]](#footnote-1)Go to the [AWS Glue](https://ap-southeast-2.console.aws.amazon.com/glue/home?region=ap-southeast-2#catalog:tab=crawlers) and create and/or run a crawler. See ‘AWS Comprehend Tutorial.docx | Load the Data into an AWS Glue Data Catalog’ for details.

### Processing the analysis output using Python

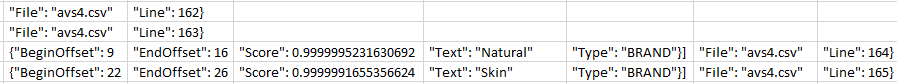
The output from the above Glue step is a database table that has columns as below.

**Table 3**





When there is more than one brand in a line, there will be more columns as…



### Problem with more than three entities

The rows in the results are limited to 17 columns. Each text item takes 5 columns for ‘BeginOffset, EndOffset, Score, Text and Type’. The next two cols are for ‘File and Line’. When there is more than one entity in the same doc line, the first 5 columns are repeated, and the ‘File and Line’ cols are added at the end. With three matches for the entity are on the same doc line, all 17 columns in the table are fully taken up. With a fourth entity on the same doc line, the col16 and 17 are used for the BeginOffset and EndOffset values for the fourth entity. There is no room for the Score, Text, Type, File and Line columns. This makes the line unusable, as the line number is not known.

Must investigate whether there is a setting or way around this.

### Problem with entities list

Phrases containing ‘and’ are split and only the first word is taken. e.g. “Head and Shoulders”, “Head & Shoulders” become ‘Head’.

Phrases with single letter as the second word are treated the same as above. e.g. “Baby U” becomes ‘Baby’

### Problem with offsets

Occasionally the BeginOffset is shifted 1 char to the right. e.g. "BeginOffset": 1, when it should be 0. This is not serious if only one or two chars. Otherwise, the masking will leave some chars out. E.g. “Ra\*\*\*\*\*\*” instead of “R\*\*\*\*\*\*\*” for “Rafferty”.

### Problem with identification

Not all entities in a line are identified. Though it sometimes identifies more than one entity in a line, most of the time it detects only the first. In some cases it does not detect any, even if the line is exactly or very similar to another line in the doc where it has detected the entity.

### Misspellings are recognised

Misspelled entities (e.g. ‘Klenex’ for ‘Kleenex’, ‘Niviea’ for ‘Nivea’, ‘Neutrogeena’ for ‘Neutrogena’, etc.) are recognised.

### Processing the results

Processing the above table requires a custom Python program. The ‘AWS QuickSight’ is of limited or no use.

The Python program created for the purpose is at the following location.

***C:\Projects\PycharmProjects\AWS\_Comprehend\parse\_comprehend-results.py***

The above program parses the AWS output for the entities, line number and the start/end positions in the document line. These words are then masked as ‘A\*\*\*\*\*\*’, where the first letter of the entity is retained.

**Output: Product title with the entity masked out.**

Z\*\*\*\*\* Anti-Chafing Cream - 75g

5698 R\*\*\*\*\*\*\*'s Garden Smooth Sweet Potato Carrot Apple 4 Months + - 120g

# APPENDIX

## The issue of not recognising multiple entities in the same doc.

The AWS Custom Entities Recognition (AWS-CER) is not picking up all occurrences of the named entities. In some cases it does not pick up an entity in one doc that was picked up in another doc. Also, if there are more than three entities in the same doc, the output lacks the line number. This makes the result unusable.

AWS-CER, however, picks up misspellings in entities. e.g. ‘Klenex’ for ‘Kleenex’, ‘Niviea’ for ‘Nivea’, ‘Neutrogeena’ for ‘Neutrogena’, etc.

## The issue of spelling mistakes.

My own Python software, ‘parse\_comprehend-results.py’, picks up all entities in the same doc. However, it does not pick-up misspelled words. A solution is to include a spell checker (*from spellchecker import SpellChecker*) that will correct common spelling errors using the *Levenshtein edit distance*[[1](https://en.wikipedia.org/wiki/Levenshtein_distance)]. This may not work with non-dictionary words such as brand names and profanity. For these, we must add a list of words/phrases by using the EntityRuler (*from spacy.pipeline import EntityRuler*).

The ‘brands.txt’ is a list of pharmacy brand names. When used with the input file, ‘avs4.csv’, which is the product list from a pharmacy, the misspelled names are corrected and detected.

1. This step is unnecessary for the analysis with Python. Instead, the ‘output’ in ‘output.tar.gz’ from the link in analysis results can be used directly. It has the same structure as shown in Table 3. However, the [problem](#_Problem_with_more) with more than 3 entities in one line persists. [↑](#footnote-ref-1)