kainos®

AWS Comprehend Evaluation

16th March 2020

Status	Definitive
Security	Commercial - In Confidence
Version No.	1/0

THE NATIONAL ARCHIVES

Contents

1. Executive Summary

This document provides a summary of the data science and engineering work undertaken in 6 weeks to deliver a working end-to-end prototype service for auto-archival of documents for The National Archives.

The report documents the capabilities of the Amazon Comprehend service for Natural Language Processing (NLP) along with findings related to the supporting Amazon services and bespoke code required to provide a data pipeline which feeds the Comprehend custom machine learning model for document classification.

The prototype developed provided the following capabilities:

- Process ~375GB of unstructured document data using secure public cloud storage and services
- Index, filter and extract text from 15 different document formats across all input documents in 2 hours
- Provide a data science pipeline for removing highly similar documents and preparing the labels and text for ingestion into AWS Comprehend
- Train binary and multi-class classifiers using AWS Comprehend which are able to accurately classify documents both across preserved / not preserved classes as well as individual disposal schedules

This report provides details of the following aspects of the prototype delivery and discovery outputs:

- Comprehend modelling findings and results
- Data pipeline tooling and development work
- Assessment of off-the-shelf functionality and bespoke development
- Lessons learned and alternative approaches considered
- Skills and expert knowledge required
- Indicative AWS hosting and operating costs

2. State of AI market in the Information Management and Compliance space

Broadly, there is a shift away from large up-front EDM software purchases to more tailored, flexible and agile solutions using the public cloud.

Public cloud computing provides a cost-effective way to store and process large volumes of unstructured and semi-structured data at scale and machine learning can aid organisations in reducing the scope of the effort required to apply structure to and extract value from these large datasets.

Challenges exist in the adoption of machine learning, both through skills to develop and evaluate viable solutions, but also the skills required to operationalise these efforts.

There is a disparity between the number of organisations who consider machine learning and AI to be important strategic initiatives and the number who feel confident in implementing them and taking their prototype solutions to production.

Due to the variety of data and potential use cases, it can be difficult to identify an off-the-shelf solution what can address an organisation's specific needs. More often a holistic end-to-end solution may entail use of a COTS product, some bespoke development, use of public cloud infrastructure and investment in the skills and cultural change required to meet business and user needs.

As applied to information management and EDRM, techniques for AI and Machine Learning will usually encounter some or all of the following challenges:

- Training needs to be accurate can be costly and time consuming to manage without an investment in automation
- Generalising classification to a wide variety of data sources and unstructured documents is challenging without bespoke code written to standardise the data
- Interpreting model results and making necessary adjustments in training of the classifier can be difficult without an understanding of classifier performance metrics

3. Amazon Web Services - Comprehend Overview

The core product under evaluation as the subject of this report is AWS Comprehend, a Natural Language Processing (NLP) product available from the AWS machine learning services suite.

Comprehend provides an API which wraps several off-the-shelf models for common NLP tasks such as key phrase extraction, sentiment analysis and topic modelling which can be used out-of-the-box without any custom model training.

Comprehend also provides an API to allow for custom models to be trained using an organisations own data, so that users may tailor a classification process which meets their own specific business requirements.

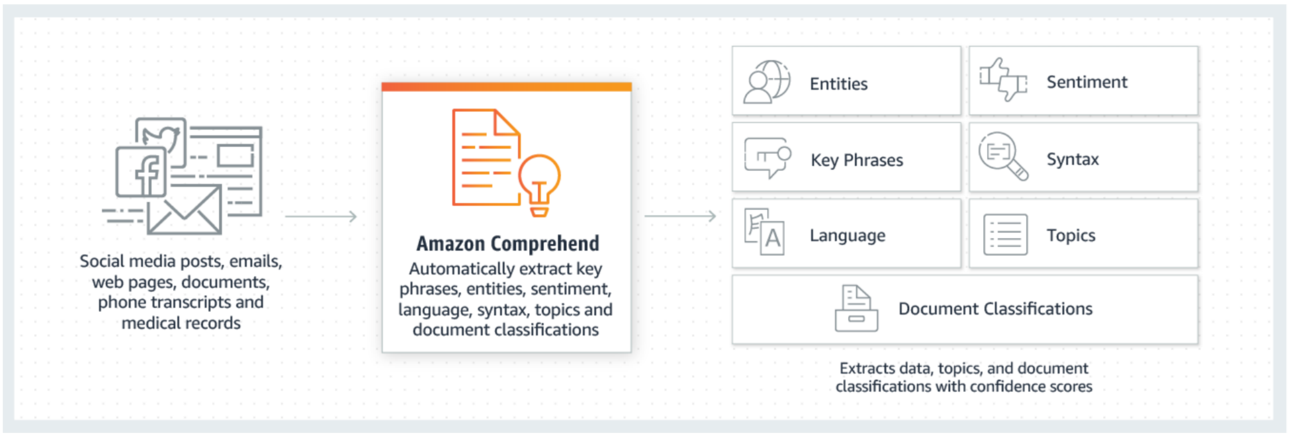

Social media posts, emails, web pages, documents, phone transcripts and medical records
Amazon Comprehend
Automatically extract key phrases, entities, sentiment, language, syntax, topics and document classifications
Entities
Sentiment
Key Phrases
Syntax
Language
Topics
Document Classifications
Extracts data, topics, and document classifications with confidence scores

Figure 1 Comprehend NLP Functionality (ref. AWS)

Comprehend is a fully managed service – there are no servers to provision with a flexible "pay for what you use" pricing model (ref. Appendix B)

The service's potential is most fully realised when taken within the context of the powerful integration that is available with the rest of the AWS technology platform, with which it is designed to integrate.

As demonstrated in the detail of this evaluation, use of services such as Simple Storage Service (S3) for text storage and AWS Compute and Textract services for document pre-processing can deliver an end-to-end capability for processing and classifying a wide range of unstructured data.

4. Data Collection

4.1 Import from varied data sources

4.1.1 Amazon S3

For this evaluation all data was initially stored with the Simple Storage Service (S3) which is Amazon's web-scale cloud storage service.

Each of the AWS services used in this prototype can be configured to read directory from S3 buckets – a bucket being the route storage container or top-level directory – without requiring any custom components for integration.

These services are designed with S3 as the mechanism for file storage in mind and it is specifically provided for this purpose. The general recommendation is that all input data is made available in an S3 bucket as the initial data ingestion step for the pipeline.

For example, the following bucket structure was used for the prototype pipeline:

Bucket	Contents	Tools
tna-poc-staging	Bucket to store input files	Input for custom code + Textract
tna-customer-files	Store index files containing the list of Objective and Shared Drive input file locations and their basic metadata	Comprehend, Textract, custom code
tna-extracted-text	Store raw text from text extraction process	Output of custom code + Textract Input to Comprehend batch
tna-tar-bucket	Store archive produced by Comprehend batch process	Output of Comprehend batch
tna-classifier-test/ output	Store classification output	Output of Comprehend classification

In addition to these main items, we created other intermediate buckets which Comprehend uses to store files during model training runs and for ancillary outputs such as audit log information.

4.1.2 Local File System

As stated above, being cloud services and with the intended use case that of bulk classification of documents, the recommendation is to store the input data in S3.

Apart from the close integration with the S3 service and the AWS text processing tools, the co-location of the data within the AWS environment is recommended to ensure that processing is both performant and secure.

The initial upload of the input data from local disk to the AWS S3 service took approximately half a day to transfer the ~375GB of data over the corporate internet connection using the secure S3 command line client.

Another possibility, not evaluated as part of this proof-of-concept is to use AWS Elastic File System (EFS) which is an NFS-compatible file system solution from Amazon. This can be used to mount on-premise file storage for direct access by resources in the AWS cloud.

Note that this option is predicated on a private networking link being established between the on-premise infrastructure and infrastructure in the AWS account.

This can either be accomplished virtually (using a private VPN) or with a dedicated fibre connection (Amazon DirectConnect). Whilst this makes a solution viable from a security perspective, the performance (and financial) cost of transfer of data over the network would still need to be assessed and taken into account.

4.1.3 Other Options

Batch processing operations such as Comprehend training require an S3 location to efficiently read the set of target files. However, the APIs for Classification and certain APIs provided by Textract and other services accept data from direct HTTP POST connections.

During evaluation, indexes were used which stored the list of text items to be passed to Comprehend for classification. Whilst this was stored in a CSV file for convenience, database options such as Relational Database Service (RDS) and DynamoDB (Amazon's NoSQL offering) were also considered as storage options for this data and in a production system one of these would be the better option for maintaining indexes.

5. Pre-processing

5.1 Data format extraction

5.1.1 Supported Formats

The table below shows the full list of documents types which were considered as part of the evaluation and whether they were included in the prototype:

Extensions	Description	Included in prototype?
.doc / .docx	Microsoft Word	Yes
.pdf	PDF	Yes
.msg	Email	Yes (body, no attachments)
.xls / .xlsx	Excel	Yes
.txt	Text	Yes
.csv	Comma-Separated Values	Yes
.htm / .html	HTML files	Yes
.pptx	PowerPoint	Yes
.rtf	Rich Text Format	Yes
.jpeg, .png, .tiff	Images	Yes
.mov, .mp4 etc.	Video	No*
.mp3, .wav etc.	Audio	No*

* Please refer to section 9.2 Additional Data Formats for further information on audio/video

5.1.2 AWS Textract

Amazon Textract is a managed service for extracting text and metadata from scanned documents.

For the purpose of this evaluation, the OCR functionality which extracts raw text and positional information from the documents was used, however the service also provides the capability to extract form and table data in a structured format.

None of this functionality requires any pre-training or custom model development.

AWS Textract supports the following three document formats for scanned documents:

- PDF
- JPG/JPEG
- PNG

As such, AWS Textract was evaluated primarily as an extension of the text extraction process developed for the prototype solution.

It was found to have generally superior accuracy on scanned documents in the above formats compared to using the open source 'Textract' library written in Python.

Given that this naming is confusing – we will refer throughout the remainder of this document to 'AWS Textract' and 'OSS Textract' when referring to either the Amazon service or Open Source Python library respectively.

5.1.3 Pipeline Development

The following diagram illustrates the end to end process which was created as part of this evaluation to process the raw input data formats across the range of documents supplied as part of both the Objective/EDRM and Websites/Shared Drive data sets and to use Comprehend to train and classify documents using that extracted text.

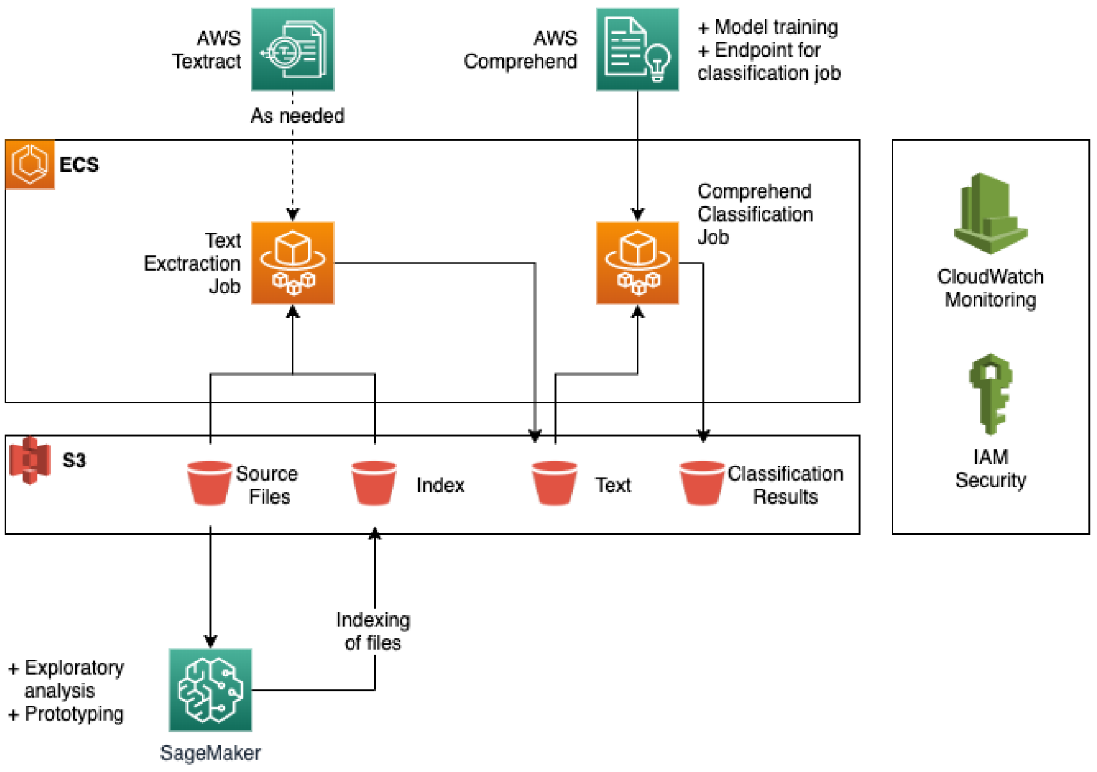


Figure 2 E2E data pipeline

Development Process

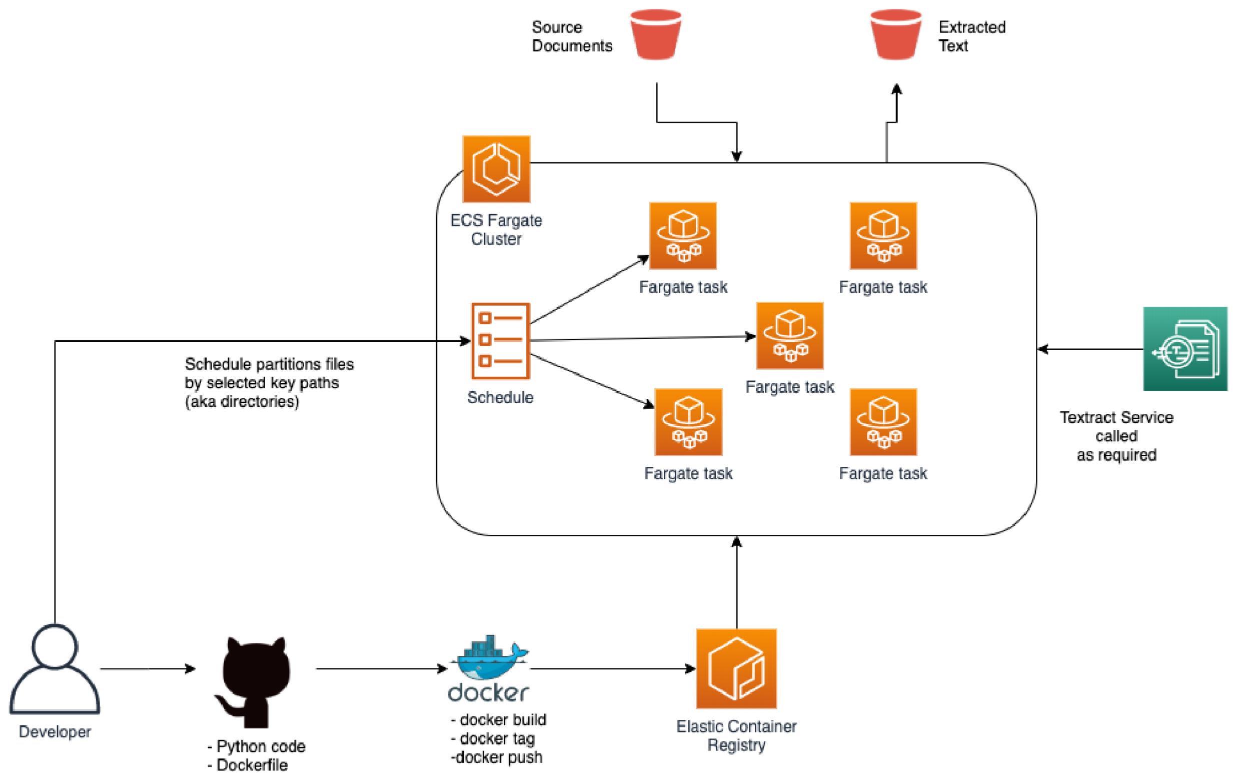


Figure 3 Text extraction process

The figure above illustrates the following development process for text pre-processing:

- Developer writes text processing code in Python and packages it using Docker
- Docker image is published to Elastic Container Registry (ECR) to make it available for use in AWS
- Developer writes a task schedule which defines the text processing batch job in Elastic Container Service (ECS)
- The ECS job pulls in the configured Docker container from ECR and runs the job. The job will have been configured to pull in documents from a source S3 bucket and write the extracted text out to another target bucket location.

Task Definition

The schedule is split into two jobs – one for the Objective/EDRM data and other for the Websites/Shared Drive data.

The code which runs within each task performs the following steps for both input sources:

1. Read the pre-indexed list of files for the data source
2. Filter the index using a configured list of:
    a. File extensions (e.g. only read documents, exclude media like .mp3)
    b. Prefixes (e.g. only process documents under "Websites/clive 7awkins")
3. Loop through the filtered list and process in fixed sized batches:
    a. Attempt to extract text from each file in the batch locally
    b. OPTIONALLY: Send to AWS Textract if it could not be processed locally
    c. Filter out:
        i. Small files (< 100 characters in length)
        ii. Empty files

    iii. Special characters (e.g. |)
  d. If the text was valid then output it to the S3 extracted text bucket

Performance

Like Comprehend and most other AWS services, AWS Textract exposes an API which allows external clients and processes to interact with it.

Thus an API call to AWS Textract is inherently slower than calling the Python code to invoke the OSS Textract function on a document.

It may take approximately 1 minute to process an average sized PDF (say 1MB) using the Textract Asynchronous API.

As a general approach we have used the OSS Textract library to process all documents which are in one of the non-PDF/Image formats and for these formats we only hand off a call to AWS Textract where we are unable to process the PDF/Image locally.

Output

The final output of this process is a new S3 bucket which contains all of the text which the job has been able to extract from documents with the supported file extensions.

The directory structure of the output retains the same hierarchy and thus file names as the original data and we use this convention to link the two. The only difference is that any individual files now have the **.txt** extension in the output bucket.

The table below shows some of the statistics for processing the files for text extraction using the concurrent task schedule shown in the figure above

Totals are post-filtering, i.e. do not include documents that we have filtered out based on their file type.

The % of files not extracted accounts for cases we drop due to:

- Small file (< 100 characters)
- File just contains garbled content (special characters)
- No text content was extracted
- Other errors (e.g. there was an issue with the file encoding)

Between the two data sets, generally we see a lower overall percentage of usable text content in the Shared Drive data, which we would expect given the nature of the data set.

NOTE: Total run time is **per task** so the total elapsed time for processing the entire Objective data was 1hr 7 mins and for the Share Drive data 2hrs 20 mins – the time of the longest running task.

Objective data set

Task ID	Sub-directories processed	File counts	Total Run time
1	a/DA/ a/DS_1/ a/ASD/	Total to process: 23,994 Total extracted: 22,299 (93%)	1hr 7 mins
2	a/LPaD/	Total to process: 20,474	1hr 5 mins

	a/MaC/ a/PMO/ a/LS/	Total extracted: 19,539 (95%)	
3	a/KaIMT/ a/ITO/ a/IP/ a/GA/	Total to process: 23,995 Total extracted: 22,968 (96%)	59 mins
4	a/PPDaCM/ a/RaAE/ a/SP/ a/WA/ a/CEaE/ a/DS/	Total to process: 13,273 Total extracted: 11,856 (89%)	45 mins
5	a/Am/ a/CaT/ a/CC/ a/20032008fp/	Total to process: 16,943 Total extracted: 15,220 (90%)	42 mins

Website data set

Task ID	Sub-directories processed	File Counts	Total Run Time
1	DSD Newsletter/ [redacted] _docs/ Taxonomy/ Moving Here/	Total to process: 3086 Total extracted: 1786 (58%)	18 mins
2	FOI/ EngD/	Total to process: 1677 Total extracted: 1189 (71%)	10 mins
3	Webteam/ [redacted]	Total to process: 18,243 Total extracted: 9173 (50%)	2hrs 20 mins

Redacted under FOI exemption 40(2)

Deployment and Orchestration

As mentioned above, ECS was used to provide the platform on which text extraction and bath comprehend processing jobs could be run.

Specifically AWS Fargate was used which allows the Docker containers which package the applications to be run in a serverless manner on ECS.

The benefit of this is that developers can concentrate on running and testing the application without having to explicitly specify and configure a cluster of dedicated compute hardware.

Fargate allows tasks to be triggered by other events or to be scheduled via simple CRON syntax (i.e. a date and time) which was the option chosen for the prototype.

Comprehend Custom Classification

After the Text extraction jobs have been completed, the Custom classification job will be called to classify each of the extracted text files stored on S3. The job passes each file to a classifier, which is trained to classify TNA documents using comprehend. This will be covered in greater detail in the modelling section of this report (6. Models).

5.2 Duplicate detection

While each document had a unique file key, a portion of them shared identical content with at least one other document. This presents two issues, specifically by introducing sampling bias into the model training and producing overly optimistic performance metrics during the validation and/or test stage.

Consequently, duplicate documents were removed during pre-processing to ensure that the model did not overfit during training, that is learn the patterns in the training dataset too closely so that it would not generalise well to new, unseen data.

As with complete duplicates, documents that are near-duplicates, those that share an extremely large proportion of text to at least one other document, would also need to be removed from the dataset that would be used to train the model to ensure generalisability and limit training bias.

The similarity between documents was measured by first converting each text document into a vector (a numerical array with magnitude and direction), more specifically a term-frequency inverse-document-frequency vector, and then calculating each document vector's cosine similarity scores with every other document vector, producing a nonnegative symmetric matrix of cosine similarity scores.

	Document 1	Document 2	Document 3	…	Document n
Document 1	1.00	0.64	0.97	…	0.98
Document 2	0.64	1.00	0.81	…	0.99
Document 3	0.97	0.81	1.00	…	0.75
…	…	…	…	…	…
Document n	0.98	0.99	0.75	…	1.00

As shown in the example above, these similarity scores ranged from 0 – 1, by which a higher score indicated a higher similarity between documents (1.0 indicating perfect similarity, for example). A threshold of 0.97 was established as a means of identifying documents and their near-duplicates, although this was subject to change depending on the amount of data used for training and testing. Documents that had a similarity score of ≥ 0.97 with one other document or more (other than itself, of course) were dropped from the dataset accordingly.

5.3 File metadata

As part of the data pipeline for filtering and text extraction, the initial stage is to pre-index all of the files stored in the source S3 bucket.

This index provides basic file metadata such as the key, extension/type and datetime information and enables fast access to different subsets of files as desired by filename prefix (i.e. a particular sub-directory) or by extension.

This more general metadata does not impact on the Comprehend modelling approach as the classifier is trained purely on documents contents, however there are other capabilities of the service which may be used as a way of building up a more sophisticated metadata catalogue.

Comprehend offers additional natural language processing API's, which can be used to gather insights on the data. We evaluated entity recognition, which successfully extracted entities such as organisation, date or location and returns a confidence score. We ultimately didn't use this service, as we believe the custom classifier has it's own inbuilt entity recognition and doesn't require this information to be passed in. If this information was needed for external analysis of the documents, the API could easily be called and returned.

6. Modelling

In addition to having the ability to obtain various insights from documents such as entity or key phrase extraction, sentiment analysis and topic modelling, AWS Comprehend offers the capability to train custom models in order to classify documents.

This is a two-step process. First, a custom classifier is trained on the classes of interest and once the classifier is trained, unlabelled documents can be sent for classification.

The Objective (labelled) data set is composed out of over 90,000 files across 31 repositories and 20 disposal schedules. Given the size and complexity of the Objective data set, a number of modelling approaches were taken to demonstrate how Comprehend performs across different classification use cases.

A summary of the use cases is presented below:

Use Case 1 (2 Class Generic Model):

Evaluate performance of a 2-class model trained and tested on documents across all repositories.

Use Case 2: (2 Class Model, Limited Repositories):

Evaluate performance of a 2-class model trained on a subset of repositories.

This is done to see if applying selection logic with respect to which repositories are included in the train/test data sets offers an improvement in model performance over Use Case 1.

Use Case 2.1: (Multiclass Model, Limited Repositories):

A variation of use case 2, demonstrates performance of a multiclass model on a subset of repositories.

This is done to demonstrate if disposal schedules can be predicted accurately.

Use Case 3: (Single Repository, Test on Separate Repository):

Evaluate performance of a model trained on one repository and tested on documents from a different repository.

The scope of this use case is to demonstrate if a model can generalise across different repositories / departments.

Use Case 3.1: (Single Repository, Small Training Subset):

Evaluate performance of a model trained on a small subset of documents (150/class) from one repository and tested on the same repository.

This is to demonstrate if a model offers acceptable performance while trained on a very small subset of data.

6.1 Inputs

When training a Comprehend classifier, the data must be in a single .csv file. For each line in the file, the class name is placed first, followed by the text of the complete document.

CLASS	Text of document 1
CLASS	Text of document 2
CLASS	Text of document 3

Comprehend requires a minimum of 10 documents per class to train the model although 150 documents per class has been attempted as a minimum as part of this report (use case 3.1).

The total size of the training document must be less that 5GB and each individual document within the .csv must be less than 10MB. All documents used for the use cases below were below 100KB in size, which constitute around 98% of the total .txt documents.

6.1.1 Use Case 1: Generic 2 Class Model

A total of 92,905 documents split across 31 repositories were part of the initial corpus of documents for the training and testing of the generic 2 class model.

Out of these documents, 67430 documents remained after similar documents were removed as well as documents above 100KB. With a 90-10 ratio of training to testing documents, 60,687 documents were used for training and 6,743 were used for testing.

A summary of the input dataset is provided in the table below:

	No. of Documents	Class Distribution
Total (pre-cleaning)	92,405	70,811 Not_Preserved
		22,094 Permanently_Preserved
Total (post-cleaning)	67,402	50,089 Not_Preserved
		17,341 Permanently_Preserved
Training set	60,687	45,080 Not_Preserved
		15,607 Permanently_Preserved
Test set	6,743	5,009 Not_Preserved
		1,734 Permanently_Preserved

6.1.2 Use Case 2: 2-Class Model Trained on 5 Repositories

Compared to the generic 2 class model trained on the entire text corpus, the second use case focused on a specific set of repositories:

- Digital Archiving (21,664 documents)
- Information Policy (10,463 documents)
- Archives Sector Development (3,841 documents)
- Collection Care (1,826 documents)
- Strategic Projects (1,292 documents)

These repositories comprised of 27,579 documents in total. After removing files above 100KB in size as well as duplicates and near duplicates, 27,256 documents remained. With a 90:10 ratio of training to testing documents, 24,531 documents were used for training and 2,725 were used for testing.

	No. of Documents	Class Distribution
Total (pre-cleaning)	27,579	18,227 Not_Preserved
		9,352 Permanently_Preserved
Total (post-cleaning)	27,256	17,983 Not_Preserved
		9,273 Permanently_Preserved
Training set	24,531	16,184 Not_Preserved
		8,347 Permanently_Preserved
Test set	2,725	1,799 Not_Preserved
		926 Permanently_Preserved

6.1.3 Use Case 2.1: Multiclass Model Trained on 5 Repositories

Use Case 2.1 makes use of the same repositories as Use Case 2, but instead of the Permanently Preserved/Not Preserved category labels, is labelled according to the 15 disposal schedules for these documents. Of these 15 disposal schedules, 12 are in the "Not Preserved" category and 3 and in the "Permanently Preserved" category.

A distribution of these documents across disposal schedules can be found in the table below:

	No. of Documents	Class Distribution
Total (pre-cleaning)	27,522	Not Preserved: 02 5959 24 4085 05 2626 16 1372 23 1277 20 870 07 776 11 574 25 202 03 193 10 183 28 109 Permanently Preserved: 21 1745 24B 722 33 6829
Total (post-cleaning)	27,199	Not Preserved: 02 5959 24 4085 05 2626 16 1372 23 1277 20 870 07 776 11 574 25 202 03 193 10 183 28 109 Permanently Preserved: 21 1745 24B 722 33 6828
Training set	24,479	Not Preserved: 02 5272 24 3611 05 2329 16 1235 23 1148 20 777 07 697 11 516 25 173 03 171 10 157 28 98 Permanently Preserved: 21 1535 24B 641 33 6119
Test set	2,720	Not Preserved: 02 586 24 402 05 259 16 137 23 128 20 87 07 77 11 57 25 19 03 19 10 17 28 11 Permanently Preserved: 21 171 24B 71

🟩	🟩	33	679

6.1.4 Use Case 3:

A total of 5,074 documents were considered from the 'Information Management' repository, with the intention of training on the entire corpus of documents within this repository and testing on a separate, unique repository to measure generalisability.

Once documents exceeding 100 KB had been removed, and duplicates and near-duplicates had been removed, the remaining 4,652 documents were used for training. Documents from the 'Strategic Projects' repository, amounting to 3,815 total, were then preserved for testing.

A summary of the input dataset is provided below:

	Number of Documents	Class Distribution
Total (pre-cleaning)	5,074	3,365 – Not Preserved
		1,709 – Permanently Preserved
Training set (post-cleaning)	4,652	2,991 – Not Preserved
		1,661 – Permanently Preserved
Test set	3,765	3,443 – Not Preserved
		372 – Permanently Preserved

6.1.5 Use Case 3.1:

An additional model was trained using a small sample from the 'Information Management' repository, 300 documents – 150 documents per class, which would then be tested on the remaining documents within the repository, totalling 4,352 documents. This would give an indication of Comprehend's ability to identify patterns that distinguish permanently preserved documents and those that aren't with the near-minimum amount of training data.

A summary of the input dataset is provided below:

	Number of Documents	Class Distribution
Training set	300	150 – Not Preserved
		150 – Permanently Preserved
Test set	4,352	2,841 – Not Preserved
		1,511 – Permanently Preserved

6.2 Outputs

For an input data set that is formatted as one document per line, the output file will contain one line for each line in the input document. Each line will contain the file name, the line number of the input line, the classes found in the document and the confidence level for the particular classification. These outputs have then been transformed into confusion matrices as well as model performance ratings such as accuracy, precision and recall.

```
{"File": "test_v3_less_than_10.csv", "Line": "13686", "Classes": [{"Name": "PERMANENTLY_PRESERVED", "Score": 0.8441},
{"Name": "NOT_PRESERVED", "Score": 0.1559}]}
```

Figure 4: Example of Comprehend classification output

6.3 Model Customisation

6.4 Model Pipeline development

Development Process

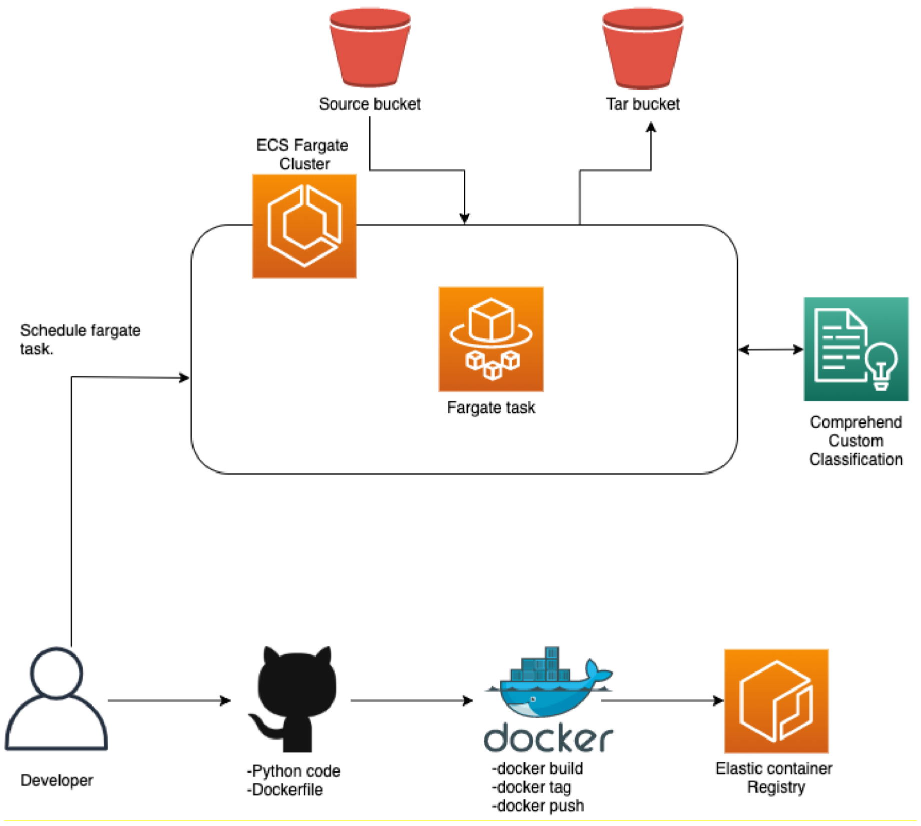


Figure 5 Custom classification process

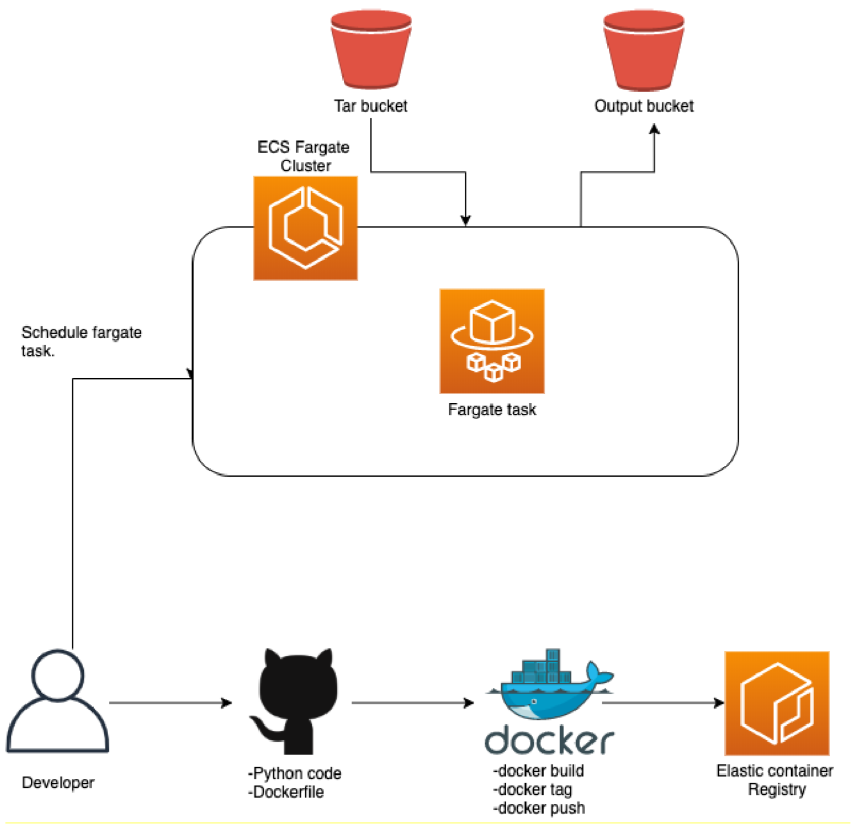


Figure 6 Custom classification Extraction Process

Task definitions

There are two jobs which must run synchronously, the first job performs the classifications and the second performs the extraction process. The files can be either multiple files within a single folder or a single csv, they will be classified and then output in the same format. The job ID created by the custom classification job must be passed into the custom classification extraction job.

Custom Classification

1. The task job calls the custom classifier, with specified variables from the user
    a. The input bucket
    b. The file input format- whether it is a folder of text files or a single csv
    c. The output bucket-tar bucket
    d. The prefix- the file or folder to be classified
    e. The data access role
    f. The document classifier
2. This will then run via comprehend and store the result in a tarfile in S3. The call will return a comprehend classifier job id that will refer to the tarfile and can be used to extract the contents.
3. The tarball file is zipped and will require extraction, but contains the classifications produced by the classifier, specifying the classes used.

Custom Classification extraction

1. This task job uses specified variables to extract the tarball
    a. The input bucket- tar bucket
    b. The file input format- whether it is a folder of text files or a single csv

   c. The output bucket
   d. The prefix- the file or folder where the extracted file will be stored
   e. The job id of the classification
2. The task job will locate and read in the tarball from the specified input bucket on S3 via the extract_tarball class. The returned result is a list of dictionaries, each containing the classification of a file.
3. The tarball is then processed dependent on the input format
   a. Store single file:
      i. Creates a data frame of predicted classes and extracted confidence scores
      ii. Writes data frame to single csv
      iii. Stores csv in S3, under specified location and original file name.
   b. Store multiple files
      i. Processes each dictionary in the list, extracting each file as json
      ii. These are written to S3 as the original file name, under the nested directories.

Performance

The jobs themselves take a very short time to run, with the Custom Classifier making it's call to the comprehend function in under 1 second. For the classification for the websites files (11843 files) it took 6.06 minutes to classify. The Custom Classification extraction job took 20 seconds to extract the classifications of these files and write them to a CSV.

Output

The outputs are the files classifications. They will be output in two formats. Either as a single csv file, e.g. website_text.csv, in a folder by the same name in the output bucket, listing not preserved or preserved and the confidence for each. Or they will be saved as the same file they were extracted from, with a json containing the classification results, e.g. source txt.

6.5 Sharing / Export

The pipelines that have currently been configured for text pre-processing and data ingestion into Comprehend could be re-used to train classifiers on documents from additional repositories.

There is an element of 'fine tuning' such as adjusting thresholds for removal of highly similar documents from the training data set as well as selecting disposal schedules which have over 100 documents in order to train the model with a sufficient number of documents per class / disposal schedule.

6.6 Interpretable / Explainable Results

Building a confusion matrix gives a good indication on the classes / disposal schedules for which adding more data could help model performance. For example, if a high percentage of documents for a particular repository are misclassified, adding more document samples to that repository can help aid model results.

The models that have been created for this report have been developed by iterating through a number of input data set configurations and assessing model validation metrics and confusion matrices in order to recommend the best performing models.

6.7 Support "Technology Assisted Review"

The current Comprehend models can be re-trained if the records manager makes any corrections to the input training data set although automatic re-training of the model is not supported. Model performance metrics, which are covered in the next section, allow the user to evaluate the model and refine the model based on this feedback.

With respect to outliers, these are not automatically highlighted by the model. The most likely outcome is for an outlier to be classed as a 'false negative' (not preserved). Certain bespoke code can be written to identify documents that have different structures to what the model was trained on initially and highlight these to the records manager.

6.8 Evaluation of Results

Upon training job completion, Comprehend offers a number of model performance metrics based on a validation dataset that is automatically set aside (10% of the training data set by default). These metrics provide a way of assessing model performance during training, however, in order to get a true reflection of model performance, a separate test dataset has been used.

Test data set results have therefore been presented on a model by model basis in the following subsections. These test results are for the test data sets detailed in the Inputs subsections.

In addition to a confusion matrix, a number of metrics have been presented for the test data sets to cover model performance. These metrics have been provided to assess overall model performance as well as performance across each class.

- Accuracy
- Precision
- Recall
- F1 Score

6.8.1 Use Case 1 Results

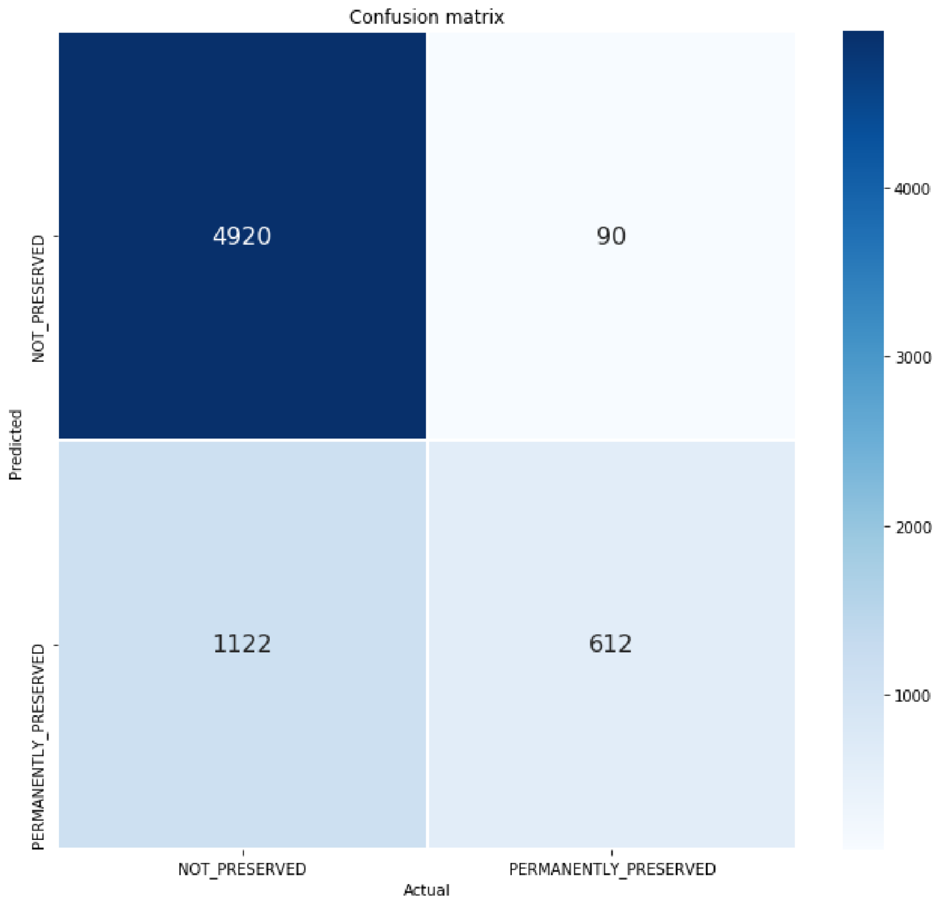


Figure 7: Confusion matrix for generic 2 class model

Overall Performance

Accuracy	Precision	Recall	F1-Score
0.82	0.84	0.66	0.79

Class Performance

Class	Precision	Recall	F1-Score
Not Preserved	0.81	0.98	0.89
Permanently Preserved	0.87	0.35	0.50

Training Time: 3 hours 27 minutes 55 seconds

Classification Time: 4 minutes 39 seconds

6.8.2 Use Case 2: 2 Class Model Results

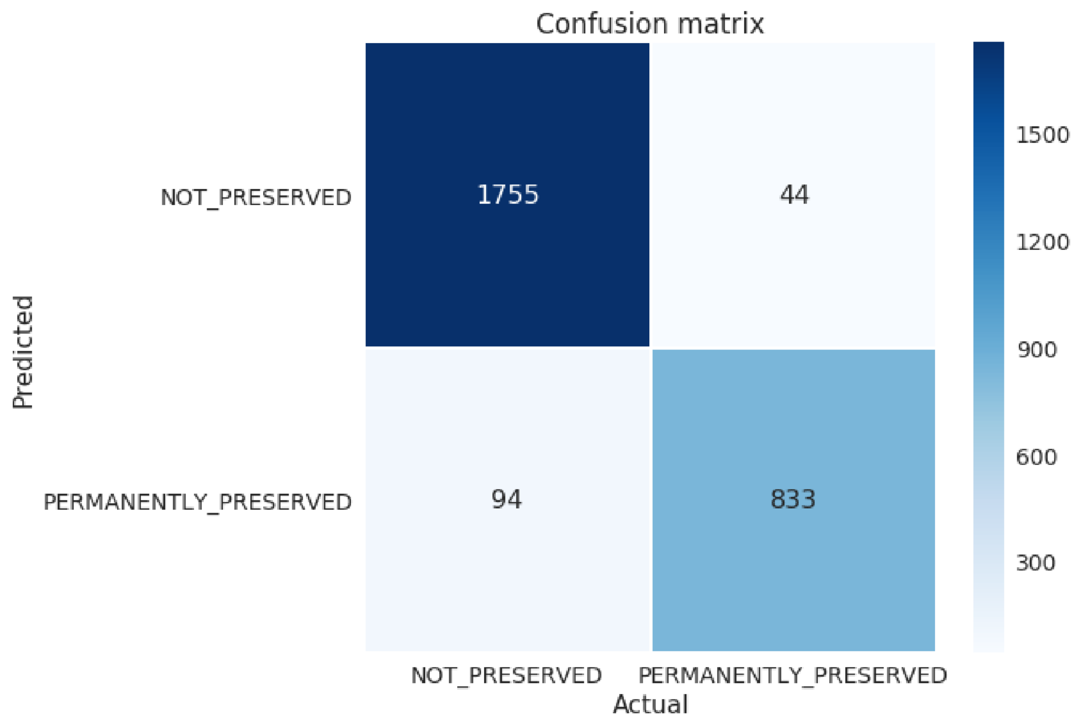


Figure 8: Confusion matrix for 2 class model trained on 5 repositories

Overall Performance

Accuracy	Precision	Recall	F1-Score
0.95	0.95	0.94	0.95

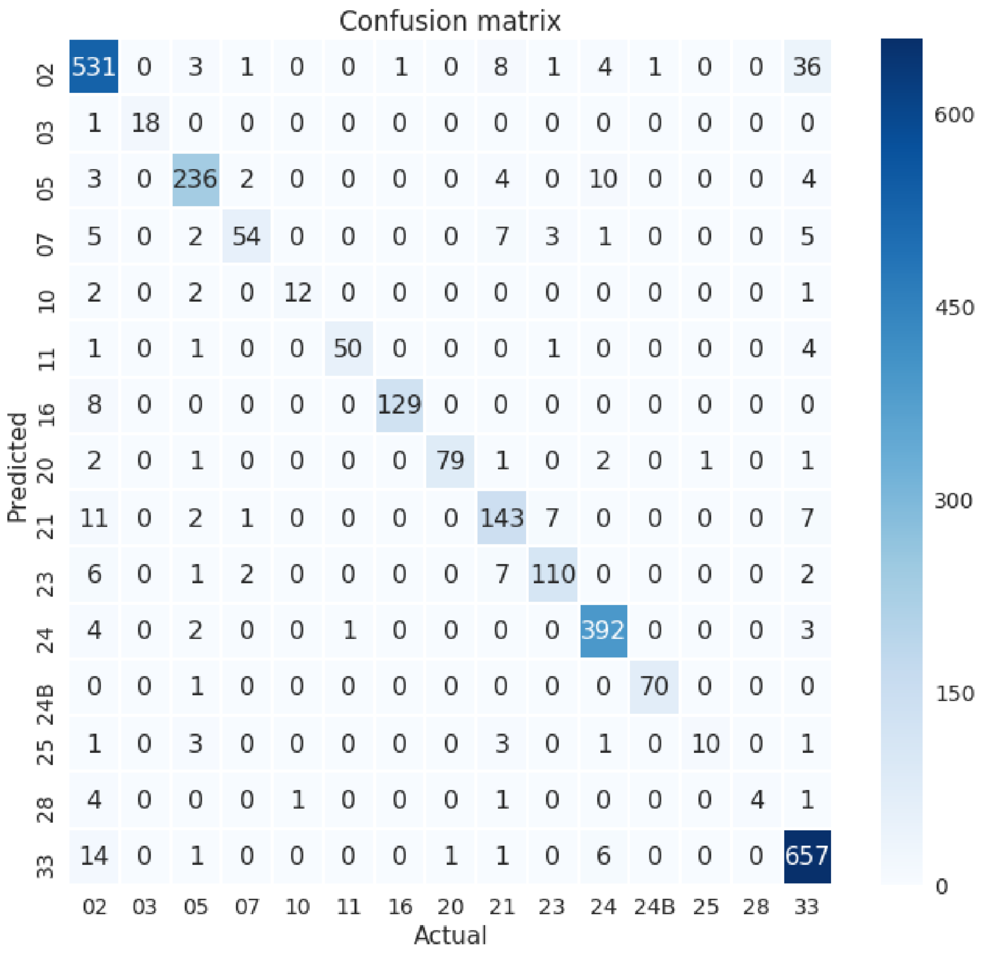
Class Performance

Class	Precision	Recall	F1-Score
Not Preserved	0.95	0.98	0.96
Permanently Preserved	0.95	0.90	0.92

Training Time: 5 hours 7 minutes 55 seconds

Classification Time: 7 minutes 39 seconds

6.8.3 Use Case 2.1: Multiclass Model Results



Predicted \ Actual	02	03	05	07	10	11	16	20	21	23	24	24B	25	28	33
02	531	0	3	1	0	0	1	0	8	1	4	1	0	0	36
03	1	18	0	0	0	0	0	0	0	0	0	0	0	0	0
05	3	0	236	2	0	0	0	0	4	0	10	0	0	0	4
07	5	0	2	54	0	0	0	0	7	3	1	0	0	0	5
10	2	0	2	0	12	0	0	0	0	0	0	0	0	0	1
11	1	0	1	0	0	50	0	0	0	1	0	0	0	0	4
16	8	0	0	0	0	0	129	0	0	0	0	0	0	0	0
20	2	0	1	0	0	0	0	79	1	0	2	0	1	0	1
21	11	0	2	1	0	0	0	0	143	7	0	0	0	0	7
23	6	0	1	2	0	0	0	0	7	110	0	0	0	0	2
24	4	0	2	0	0	1	0	0	0	0	392	0	0	0	3
24B	0	0	1	0	0	0	0	0	0	0	0	70	0	0	0
25	1	0	3	0	0	0	0	0	3	0	1	0	10	0	1
28	4	0	0	0	1	0	0	0	1	0	0	0	0	4	1
33	14	0	1	0	0	0	0	1	1	0	6	0	0	0	657

Figure 9: Confusion matrix for 15 class classification model

Overall Performance

Accuracy	Precision	Recall	F1-Score
0.92	0.94	0.83	0.92

Training Time: 28 hours 8 minutes 10 seconds

Classification Time: 4 minutes 37 seconds

6.8.4 Use Case 3 Results

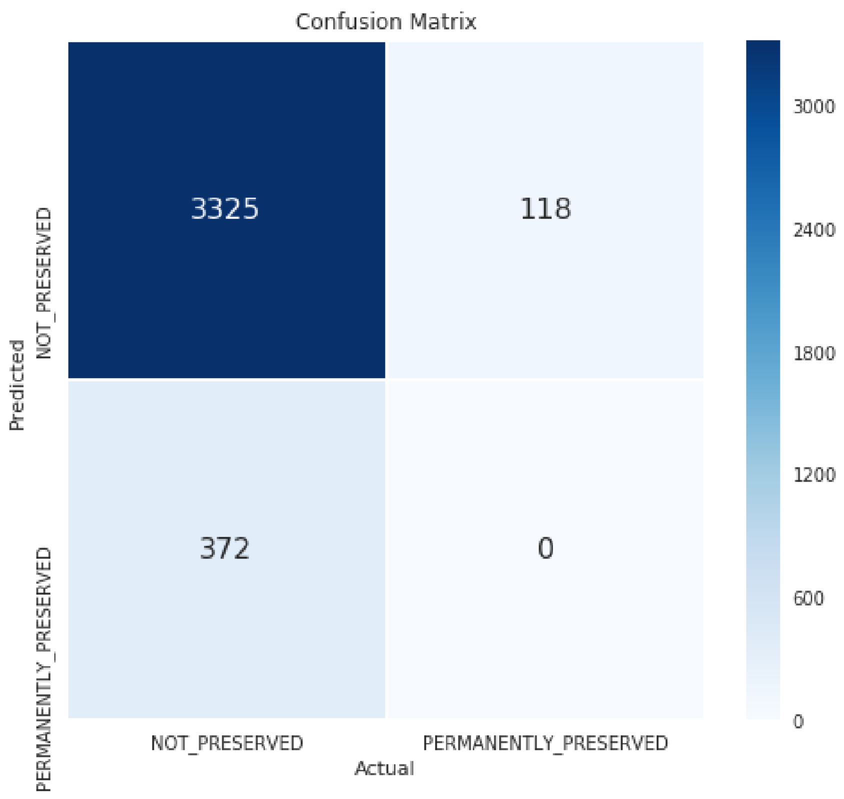


Figure 10 Confusion Matrix for single repository Binary Classification Model

Overall Performance

Accuracy	Precision	Recall	F1-Score
0.87	0.45	0.48	0.47

Class Performance

Class	Precision	Recall	F1-Score
Not Preserved	0.90	0.97	0.93
Permanently Preserved	0.00	0.00	0.00

Training Time: 36 minutes 53 seconds

Classification Time: 6 minutes 38 seconds

6.8.5 Use Case 3.1 Results

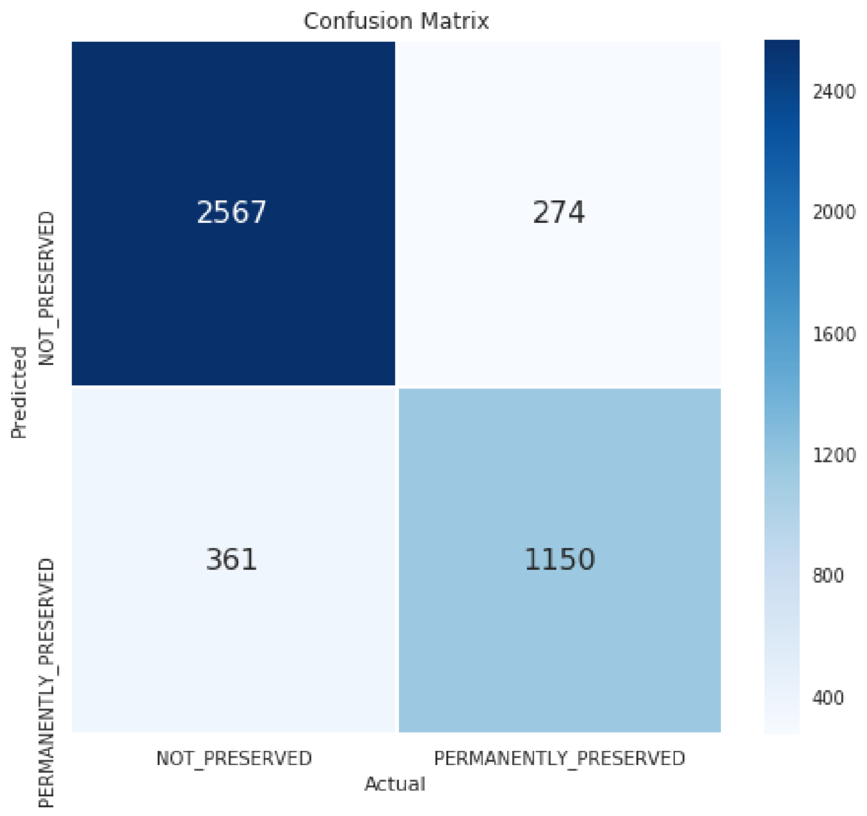


Figure 11 Confusion Matrix for Small Sample Binary Classification Model

Overall Performance

Accuracy	Precision	Recall	F1-Score
0.85	0.84	0.83	0.84

Class Performance

Class	Precision	Recall	F1-Score
Not Preserved	0.88	0.90	0.89
Permanently Preserved	0.81	0.76	0.78

Training Time: 12 minutes 53 seconds

Classification Time: 6 minutes 37 seconds

7. Results and Findings

Across the different use cases, it can be said that model performance is good/satisfactory as long as different factors need to be taken into account when training custom Comprehend classifiers. A number of observations have been recorded regarding these factors in the following paragraphs.

One of the first observation that can be made is around models generalising across different repositories. Use case 3 results have shown that a model that is trained on one repository is not able to correctly classify documents from a different repository. This can most likely be attributed to the highly dissimilar nature of documents across repositories.

Also with respect to a model's ability to generalise, the 2-class model trained on 5 repositories (*use case 2*) had better performance than the 2-class model trained across all of the repositories (*use case 1*). This could be attributed to the fact that the model used in *use case 1* did not benefit from removing repositories with a large class imbalance (>95% "Not Preserved" class) or from removing sets of documents which had less than 50 documents per disposal schedule (due to Comprehend not training well with under 50 documents per class).

The multiclass model presented in *use case 2.1*also gave good overall model results even though it used 15 disposal schedules for classification as opposed to "Not Preserved"/"Permanently Preserved" document classifications. Some of the disposal schedules such as 25 and 28 did not perform as well, which could be attributed to the relatively low number of documents for these disposal schedules in the training set (173 and 98 respectively).

Finally, use case 3.1 demonstrates that model performance is acceptable with a relatively small number of 150 documents per class. This could lead to the conclusion that a labelling effort of a few hundred documents per repository could be sufficient to enable a classifier that can be used across thousands of additional documents from that repository.

The observations across these use cases along with previous observations from the text pre-processing steps have been summed up below:

- Duplicate documents or documents with a high degree of similarity need to be removed from the training set as this could lead to bias in the model and subsequent poor performance
- Care needs to be taken to remove classes of documents/disposal schedules with less than 50-100 documents when training the model
- Highly imbalanced classes (>95% not preserved) should be subsampled in order to train the model on a more balanced data set
- Classifiers trained on one repository will not perform well on documents provided from a different repository
- Classifiers trained on 150+ documents per class have satisfactory performance, although the classifier will perform better with additional documents
- Multiclass models with 15+ classes can perform well provided all previous points are taken into account

8. Deployment / GUI

8.1 User Access Control

The AWS platform has a rich and powerful Identity and Access Management (IAM) service which acts as the lynchpin for securing cloud resources across all AWS products and services.

IAM provides the ability to manage identities through standard Role Based Access Control (RBAC) concepts such as Users, Groups and the Policies which define access to specific resources. It also includes the concept of Roles which provide a mechanism for other identities or application services to temporarily assume permissions to perform a particular task.

For example, the Users for this evaluation were organised into a single Group and the Policy for this group applied restrictions for all users, for example:

- Allow read-only access to S3 bucket containing source Objective and Websites documents
- Prevent creation of any resources outside of the eu-west-2 (London) region
- Provide least-privilege permissions to any S3 buckets used by the text extraction and batch training processes

Integration with existing, externally managed identities was not covered under the scope of this evaluation, but is possible to achieve using the AWS Cognito service.

This allows the AWS services to integrate with other identity providers such as Active Directory and can also provide Single Sign-On functionality.

8.2 Ability to track progress

All services included in the prototype development have integration with AWS's standard tooling for logging and monitoring – primarily CloudWatch.

This provided observability of the batch processes execution over time and the counts for records analysed, written, dropped etc. are taking from this logging information.

CloudWatch provides a greater degree of power than was exploited during this evaluation however, it also includes the facility to capture this logging information as metrics which can be added to dashboards and alarms and triggered actions can be configured based on user-defined thresholds and rules.

For example, alerts could be configured if a certain % of document text cannot be extracted from a new input data source, or character count could be captured as a metric from which exact costs could be calculated before passing the text to comprehend for classification.

8.3 User Collaboration

The Amazon platform is an inherently multi-user and multi-group environment for solution development.

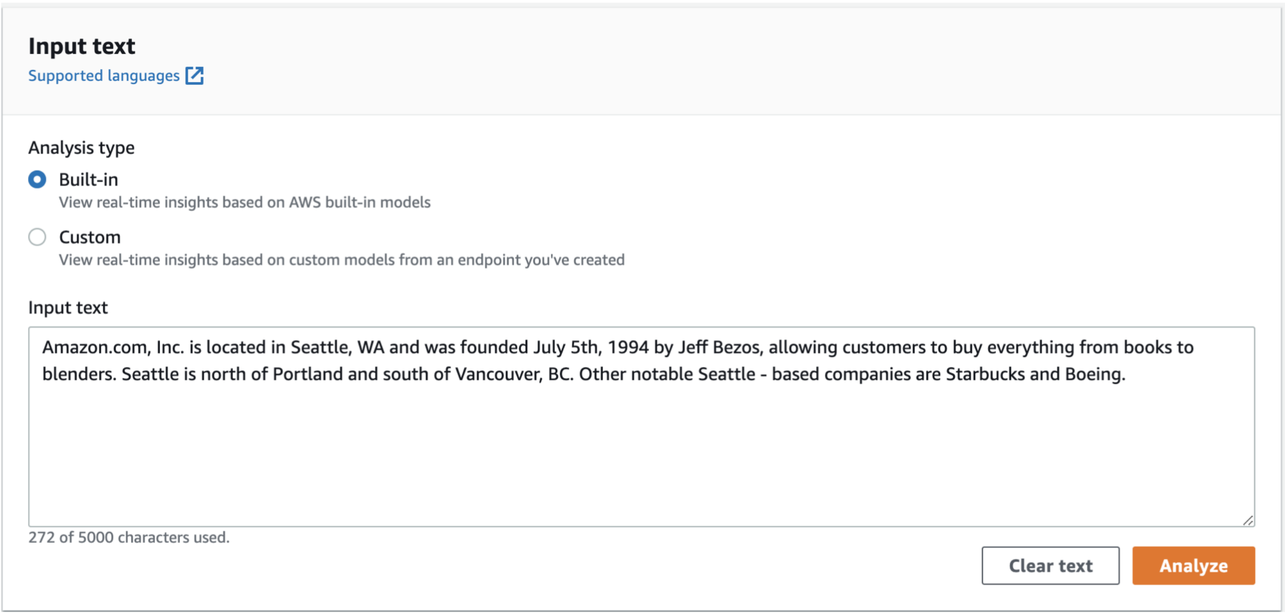
The team involved in building the prototype service mixed skills from both engineer and data science and were able to achieve success using the collaboration tools such as AWS SageMaker. SageMaker provides a means of sharing Python Jupyter notebooks for exploratory analysis and custom model development. The team primarily used these notebooks to test integration with AWS service APIs and also to analyse the document corpus. SageMaker notebooks made it easy to share both code and findings with other members of the team.

8.3.1 Comprehend OOTB

Comprehend allows users to build custom natural language processing (NLP) models and use built-in models to analyse text either directly through the UI (console) or through programming software development kits (SDKs) in either Python, Java or C#. The built-in models analyse input text and provides the following:

- Named-entity detection
- Key phrases detection
- Dominant language determination
- Sentiment determination
- Syntax determination

Confidence scores for the above insights are also provided once analysis has been completed. Programming experience is not a requirement to perform small-scale analysis with Comprehend, as text can be input into the console text box and analysed directly.



[✓ Built-in] View real-time insights based on AWS built-in models
Custom View real-time insights based on custom models from an endpoint you've created

Input text: Amazon.com, Inc. is located in Seattle, WA and was founded July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable Seattle - based companies are Starbucks and Boeing.

272 of 5000 characters used.

Figure 12 Comprehend Console Input

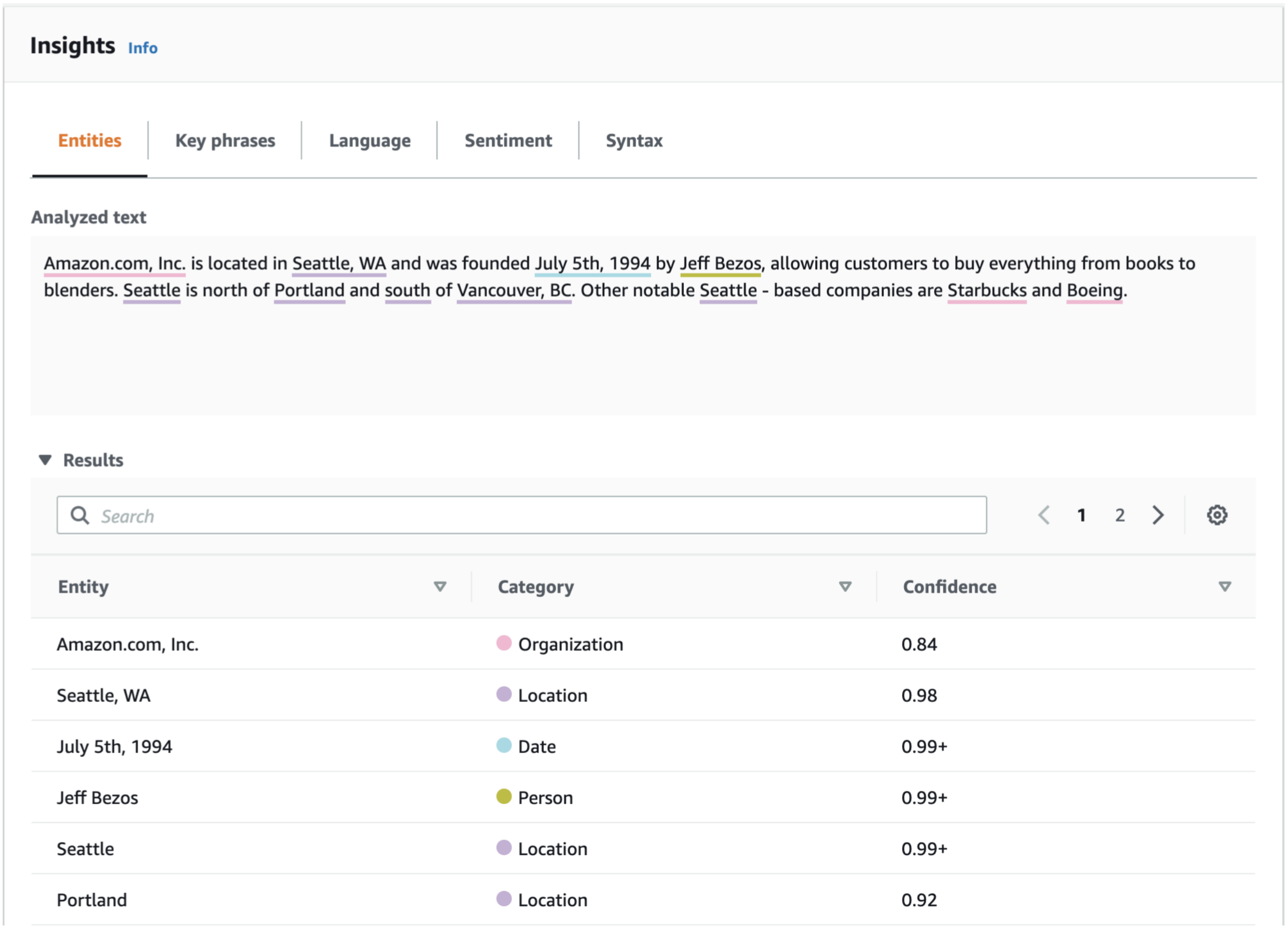
Given the nature of this project, however, with the number of documents to be processed being in the order of 100,000, analysis jobs can be performed in batch using one of the aforementioned SDKs. In this instance, the AWS SDK for Python, Boto 3, was used to analyse text extracted from the list of supported formats.

8.3.2 Comprehend Custom

In addition to the built-in models, Comprehend permits users to build custom document classification models that take advantage of its underlying deep learning architecture. These models take documents and their corresponding class labels, either binary, multi-class or indeed multi-label (in the instances when a document can have more than one associated class), as input to train a bespoke model. Customers can use the console for a code-free experience to build and train a model, and subsequently run analysis jobs, either in real-time or in batch, to classify unlabelled documents.

During training, the only mandatory configurations are the location of the S3 bucket storing the training data and an IAM role with access permissions to that S3 bucket. Furthermore, this IAM role can be created during the setup process.

Analysis jobs are set up in a very similar fashion, by which the analysis type must be



Amazon.com, Inc. is located in Seattle, WA and was founded July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable Seattle - based companies are Starbucks and Boeing.

Entity	Category	Confidence
Amazon.com, Inc.	● Organization	0.84
Seattle, WA	● Location	0.98
July 5th, 1994	● Date	0.99+
Jeff Bezos	● Person	0.99+
Seattle	● Location	0.99+
Portland	● Location	0.92

Figure 13 Comprehend Console Output

specified, custom classification in this case, the custom classification model that has been built, input and output S3 bucket locations, and an IAM role with relevant access permissions.

Alternatively, training jobs and custom classification jobs can be initiated using one of the AWS SDKs.

8.3.3 AWS End-to-End

Based on the criteria for this evaluation, the skills required to develop and end-to-end prototype on the AWS platform encompass the following areas:

Software development

Bespoke code was required to build a reliable, performant and maintainable pipeline for pre-processing of documents and to make most efficient use of the AWS PaaS services, both in the sense of record processing throughput and cost efficiency.

For example, because no one cloud PaaS service could reasonably cater for the full range of document formats under consideration, we looked to open source libraries such as the Python Textract library. This allowed us to process .doc/docx, Excel, Outlook messages etc.

To rapidly prototype solutions we leveraged Amazon SageMaker which is an environment for developing machine learning applications using Python notebooks.

SageMaker provides a great platform for collaboration between engineers and data scientists as it is usable by both and can just as rapidly be use for prototyping integration with AWS services as being used to test machine learning models.

As we refined and firmed up our approach to data pipeline components the code was migrated into standalone Python applications deployed via Docker (ref. Deployment Process)

Data Science

Comprehend offers a number of advantages with respect to training custom models as it eliminates the process of model optimisation which requires extensive knowledge of machine learning. This being said, the model needs to be provided with high quality data in order to obtain a performant model.

Given some complexities present in the data, knowledge of how to carry out exploratory data analysis and make relevant observations with respect to the structure of the data is required. As an example, within the Objective data set, there are numerous repositories with a high class imbalance. Writing custom code to identify these and apply the necessary filters is needed in this case.

As was mentioned in the Duplicate Detection section, additional methods had to be used to identify highly similar documents and informed decisions need to be made with respect to the thresholds that need to be applied in order to strike the right balance between the number of documents retained and the degree of similarity of these which could affect model bias.

Finally, good knowledge of evaluating model performance is required. Knowledge of metrics such as accuracy, prevision, recall and potentially having to apply trade-offs between these metrics can sometimes be necessary.

AWS platform / DevOps

All the services used as part of the evaluation prototype were delivered using serverless or PaaS services rather than needing to provision virtual networks or compute instances and the depth of expertise that this would require. Nevertheless, AWS platform skills are required to configure roles and permissions and connectivity between services were this relies on security group configuration.

Whilst most of the AWS services can be used through the web console UI, the benefits of the platform are only truly realised once the services and processes are automated both in their provisioning and in the ongoing monitoring and continuous improvement of an end-to-end service.

AWS CloudFormation is the recommended tool for provisioning resources through an Infrastructure-as-Code approach.

The benefits of this are:

- Speed – spin up the entire service end-to-end stack using a script, which can be automated via continuous delivery processes
- Consistency and repeatability – provisioning services has manual human intervention removed from the process
- Cost and time saving – reduced time to provision, reduced manual intervention, ability to quickly and consistently spin down resources when not in use
- Reduced risk – IaC is a form of documentation, changes can be strictly tracked and managed via version control, less risk of knowledge being siloed with individual engineers

CloudFormation resources were created for some elements of the evaluation such as IAM roles and policies, but not the majority of the pipeline due to time constraints. It is firmly the recommended approach to take services through to production on the AWS platform.

9. Future Research and Development

9.1 Scalability and Solution Architecture

To recap the deployment architecture from section 5.1.3 Pipeline Development, AWS Fargate was used to deploy multiple versions of the text processing task which could be scaled out to process different subsets of the input document sets.

The partitioning of the data for the prototype solution was fairly simple – splitting up the data set by folder path based on the number of files to be processed.

To aid this, an index was produced which could be used to filter the list of files by file extension or by a folder name prefix.

There are several options which could be considered for future work with the current solution:

- Add a module to the code which can determine at run-time how many files in the index should be processed by each task
- Move the index to a dedicated database solution such as DynamoDB or RDS and enrich with additional metadata. Currently the index only includes basic file information such as filename, extension, size and timestamps relating to creation/modified dates
- Create a CloudFormation template to automate provisioning of the pipeline and its configuration

This solution provided a number of benefits for development of a prototype:

- No dedicated network or compute infrastructure required – solution is entirely on-demand and serverless
- Allows use of Docker as a simple and consistent packaging mechanism for bespoke code
- Low barrier to moving prototype SageMaker notebook code to a packaged standalone Python application

In a production version of this application, AWS Elastic Map Reduce (EMR) could provide an alternative solution with the following key points:

- Dedicated cluster configuration, but can still be run on-demand – i.e. suitable to a periodic batch processing workload which does not run 24/7
- Highly parallelisable tasks such as text extraction from unstructured documents are a natural use case for this framework. More sophisticated partitioning / load balancing of the file processing could be achieved though the coding skills ands framework knowledge required are more specialised. For example, some knowledge of frameworks/tools such as Spark and Hive would be required to get the most out of this service
- Much higher top-end scalability. Whilst a Fargate cluster can support up to 50 concurrent tasks it is not really intended to provide the same level of task management, orchestration and general robustness of distributed workloads that EMR is designed to handle

9.2 Additional Data Formats

Audio and Video

AWS provides the Transcribe service to provide speech-to-text functionality that aims to match the quality of manual transcription at a fraction of the cost.

It supports both batch and real-time transcription APIs.

It supports 16kHz and 8kHz audio streams and multiple audio encoding formats including WAV, MP3, MP4 and FLAC.

Again, these formats match a relatively low percentage of the overall document corpus – there are 91 files matching the extensions above in the Websites/Shared Drive data set.

As for document format processing there are also a wide range of open source solutions which could be considered for integration into a bespoke batch process such as Kaldi and Mycroft.

9.3 Other Tools

AWS Glue

Glue is a managed, serverless Extract Transform Load (ETL) tool. It is based upon open source technologies such as Apache Hive and Spark. It can provide a number of benefits for ETL jobs such as:

- Data source crawling (i.e. to identify new files for processing through the pipeline)
- Schema inference and suggests data format
- Data cataloguing
- Generates code for data transformation pipelines

Glue is better suited to structured / semi-structured data sources such as CSV or JSON data where the powerful schema inference and data crawling capabilities can take away a lot of the effort involved in building transformation pipelines. For unstructured document and image data these capabilities are less relevant and it was found that good metadata or catalogue information could not be generated for these sources automatically.

This led to the simple solution used in the prototype of pre-indexing the S3 document data with some basic accompanying metadata on file extensions and paths and sharing this via a CSV stored in S3.

AWS Kendra

Not covered explicitly as part of this evaluation, but worth raising as Kendra is specifically designed to process the types of file represented in the sample data sets.

Kendra provides enterprise search functionality for unstructured document data sets stored in S3 and achieves this by using machine learning internally. The use of ML allows data to be searched using natural language style queries.

It does also provide connectors to standard file systems, APIs, SharePoint, SalesForce etc. so it is not purely limited to S3.

Kendra is currently not available outside of the AWS US regions and also not specifically a classification tool, therefore it did not fall within the scope of tools which could be evaluated, but is included here for its potential utility for search on the TNA data corpus.

10. Appendix A

Multiclass model - Class performance

```
              precision    recall  f1-score   support

          02       0.90      0.91      0.90       586
          03       1.00      0.95      0.97        19
          05       0.93      0.91      0.92       259
          07       0.90      0.70      0.79        77
          10       0.92      0.71      0.80        17
          11       0.98      0.88      0.93        57
          16       0.99      0.94      0.97       137
          20       0.99      0.91      0.95        87
          21       0.82      0.84      0.83       171
          23       0.90      0.86      0.88       128
          24       0.94      0.98      0.96       402
         24B       0.99      0.99      0.99        71
          25       0.91      0.53      0.67        19
          28       1.00      0.36      0.53        11
          33       0.91      0.97      0.94       680

   micro avg       0.92      0.92      0.92      2721
   macro avg       0.94      0.83      0.87      2721
weighted avg       0.92      0.92      0.92      2721
```

11. Appendix B

11.1 Costs and Hosting Requirements

All costs provided in this section are indicative and represent usage during the evaluation prototype development against off-the-shelf AWS service costs and against the workloads with filtering and data reduction in place.

In the development of production services, Amazon account owners will work in partnership with clients to tailor pricing appropriately to service requirements and the business case.

Cost Calculator

AWS provide a cost calculator tool which can be used to provide baseline costs against off-the-shelf pricing:

https://calculator.s3.amazonaws.com/index.html

It allows multiple standard AWS services such as S3, RDS, EC2 etc. to be added and figures entered for data consumption, throughput etc. To provide an estimate of ongoing monthly operating costs.

All prices in USD.

Main Prototype Costs

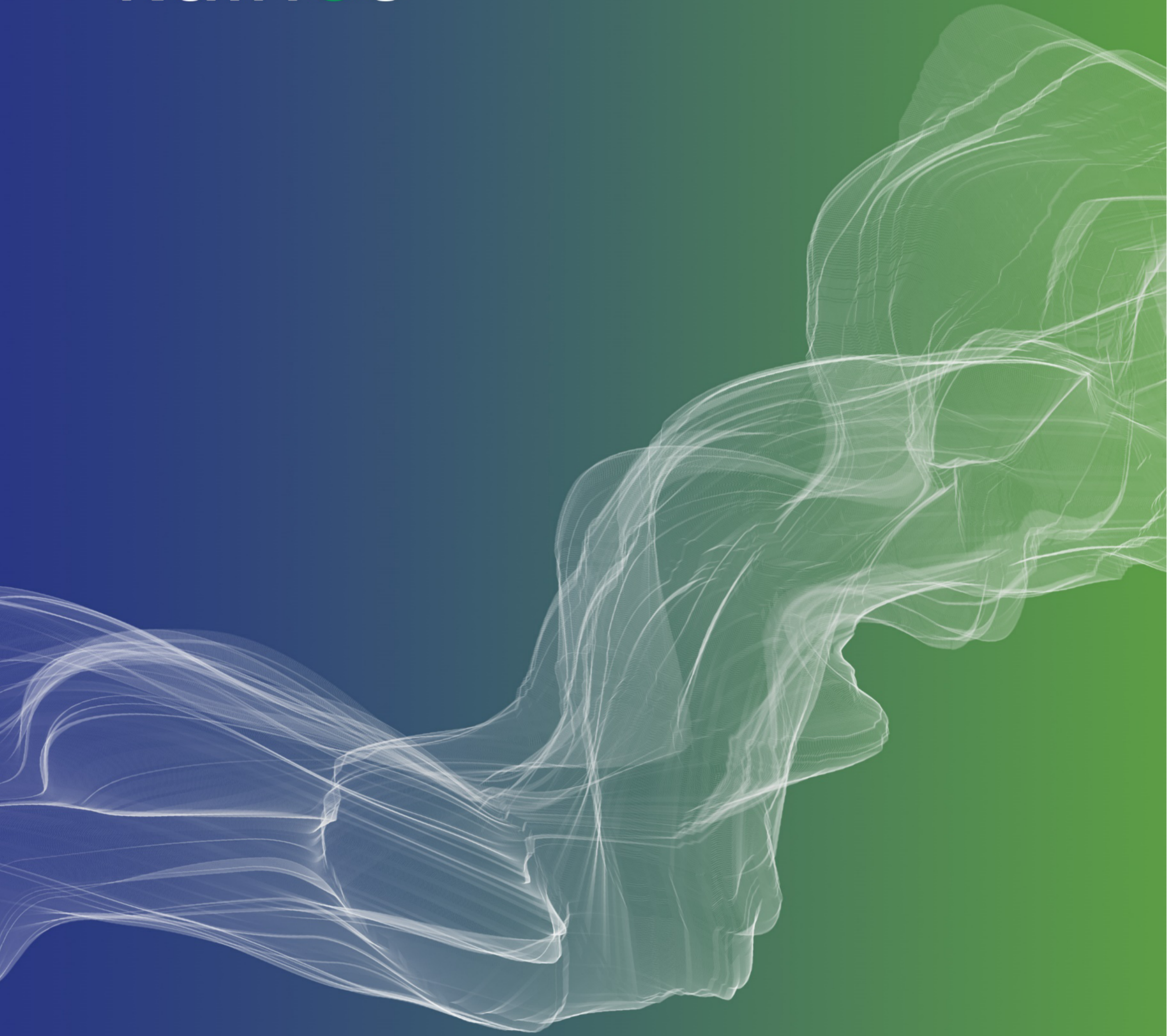
Service	Description	Base costs ($)	Usage	Usage cost
S3	Scalable file storage	1st 50TB/mo: **0.023** Data transfer out: **0.09** Transfer in + inter-service usage is free	~375GB storage + general inter-service usage for the prototype	**$15**
ECS, ECR, Fargate	Container services	Fargate vCPU: **0.0128 p/hr** per GB: **0.0014 p/hr** ECR: **0.085** per GB stored	Fargate vCPU and memory usage over 2/3 week period plus ECR image storage	**$3**
SageMaker	Data science notebooks	Dependent on underlying EC2 instance cost, e.g. ml.t2.large: **0.1299 p/hr**	Ad-hoc exploratory analysis	**$65**
Textract	OCR service	First million pages: **0.0015 p/page** > 1 mill pages: **0.0006 p/page**	Limited to PDFs and images which could not be extracted locally	**$6**
Comprehend *	NLP service	Custom classifier training: **$3** per model per hour Custom classification: **0.0005 per 100 characters**	Filter documents > 100KB leaves 98% of the documents and a 75% reduction in character count for classification	Classification **$594** Training **$283**

*The 100KB limit on text inputs to the Comprehend classifier is a measure taken within the constraints of the evaluation timeframe, the recommendation from Amazon is to filter the file list based on document size, but by using techniques to reduce document content and thus character count – the end result on reducing cost will be much the same, but with even better coverage on % of documents processed. Adopting this recommendation would be the correct route to explore in a production service.

12. Commercial statement

Rate and pricing information is confidential and commercially sensitive and covered by the 'Commercial Interests' exemption (s.43) of the FOI, as the release of this information is likely to prejudice the commercial interests of Kainos and is likely to adversely affect its (and the Customer's) future negotiating position. The period that this information should be confidential for should be 5 years.

kainos®



kainos.com

Follow our story:

@KainosSoftware

Search 'Kainos'

Search 'Kainos'