Capstone Project Report for legal document text classification

Project Overview

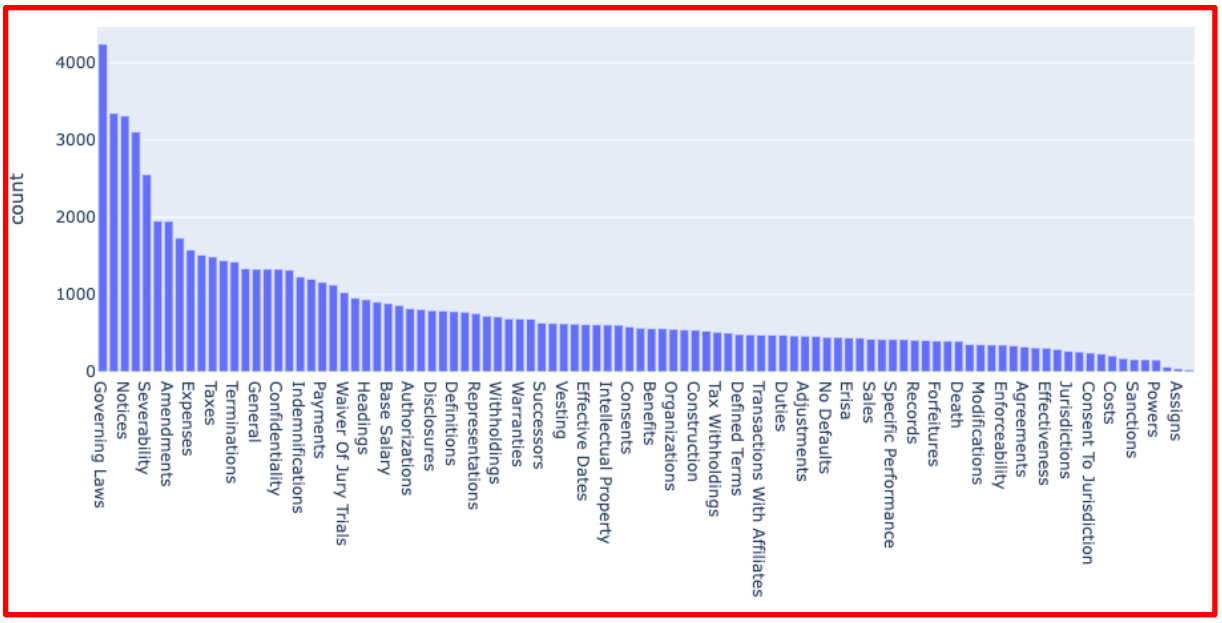
Unstructured free text is a prosperous source of information; however, extracting insights from it can be a complicated and highly time-consuming task. In recent years, thanks to many breakthroughs in the field of Natural Language Processing (NLP), which is a subfield of Machine Learning (ML), extracting actionable insight has significantly simplified. One of the essential parts of NLP is text classification which allows Data scientists to transform an unstructured corpus of text data into a structured format. During the past couple of years, novel algorithms such as Transformers[1], Universal Language Model Fine-tuning for Text Classification[2] and pre-trained language models such as Universal Sentence Encoder[3] and Pre-training of Deep Bidirectional Transformers for Language Understanding (BERT)[4] has improved the quality of text classification tasks while requiring much less tagged data to achieve state of the art results.

Problem statement

In this project, I created a SageMaker pipeline for a text classification application. The pipeline takes raw text data with its corresponding tag in a parquet format and uses varieties of NLP tools to train a text classification model. And at the end, the best-trained model that was selected based on the F1-score from the hyperparameter tuning step was selected and deployed to a SageMaker endpoint with autoscaling for real-time inference. Bespoke docker images for each node were developed and used to maximize flexibility.

Dataset

In this work I have used LEDGAR[1] dataset which is a multilabel corpus of legal provisions in contracts with label set of over 12’000 labels annotated in almost 100’000 provisions in over 60’000 contracts. The data was obtained from the US Securities and Exchange Commission (SEC) filings, that are publicly available. Each label represents the single main topic of the relevant contract.



As we can see from the above image, the number of available text for each “clause_type” ranges from ~4000 to ~20. In this work I will make a text classification for the top 5 classes and will add and “other” class to capture anything else. The following is the list of “clause_type” that I’m going to train a text classification model for:

1. Governing Laws
2. Counterparts
3. Notices
4. Entire Agreements
5. Severability
6. Others

Evaluation metrics

The evaluation metric for this work is going to be a multi-class F1-Score. This metric is a harmonic mean of precision and recall, which is the preferred metric for unbalanced datasets like we have here.

$$\text{F-1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The above equation is used to calculated F1-Score.

Solution statement:

Figure 2 depicts that pipeline that was developed for this work. In this section I will go over what each section is designed to produce.

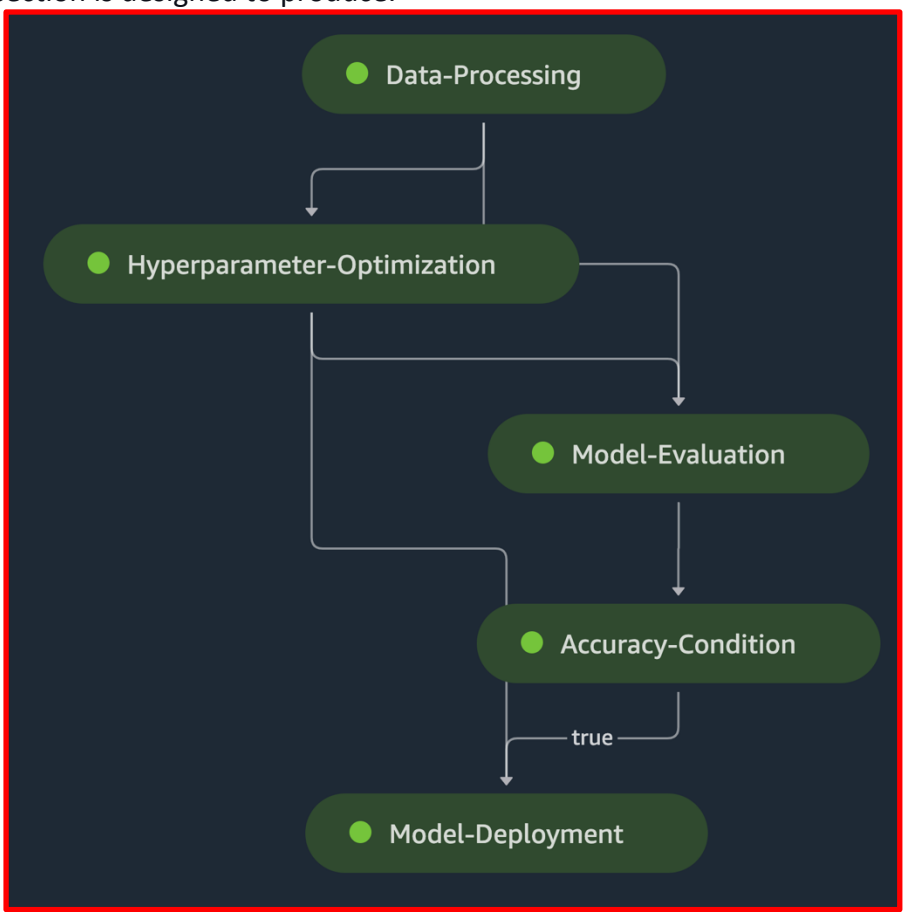


Figure2. SageMaker pipeline graph

Data-Processing:

Figure 3 shows an overview of the Data-Processing step which includes reading the data from an S3 bucket, process the data using the processing.py file, and uploading the processed data to a folder in S3 bucket that is then used in Hyperparameter-Optimization step and Model-Evaluation step.

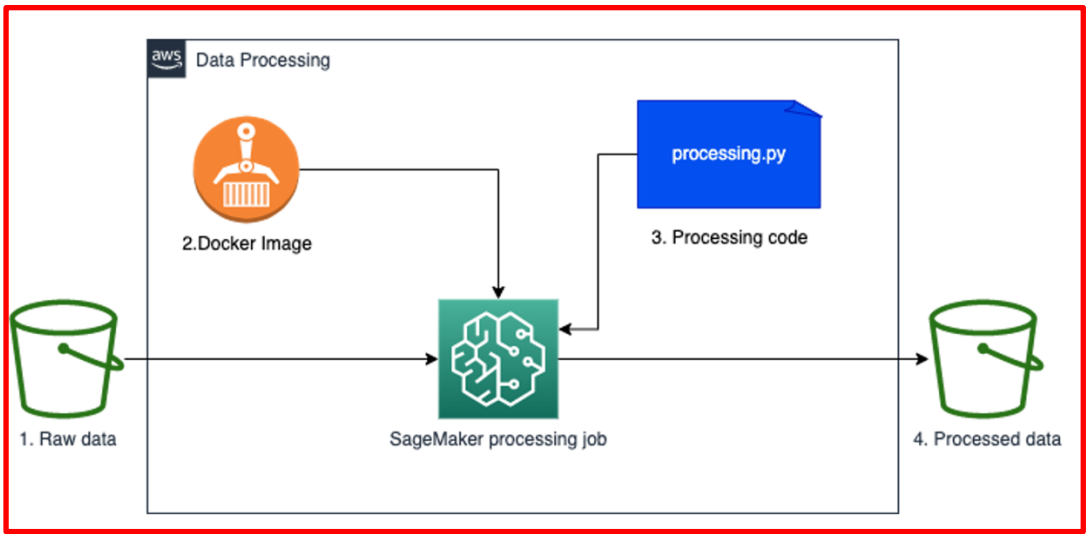


Figure3. BYOC Processing Job

The processing.py file is perfuming the following:

1. Load the parquet data file to a Panda DataFrame
2. Split the data into train/dev/test
3. Remove some stop words and clean the text
4. Reduce the number of target variables from 100 to six
5. Transform the text to vector embedding, using one of the Spacy language models
6. Save the DataFrame to a parquet file

Hyperparameter-Optimization

In this work the best model performance was selected using a HyperparameterTuner method from SageMaker pipeline. Figure 4 shows the overview of this step.

1. Processed data:
    - S3 bucket or feature store
2. Docker image on ECR
    - Python image contains all the libraries for model training
    - train.py: training workflow
3. Export model artifact to a dedicated S3 bucket

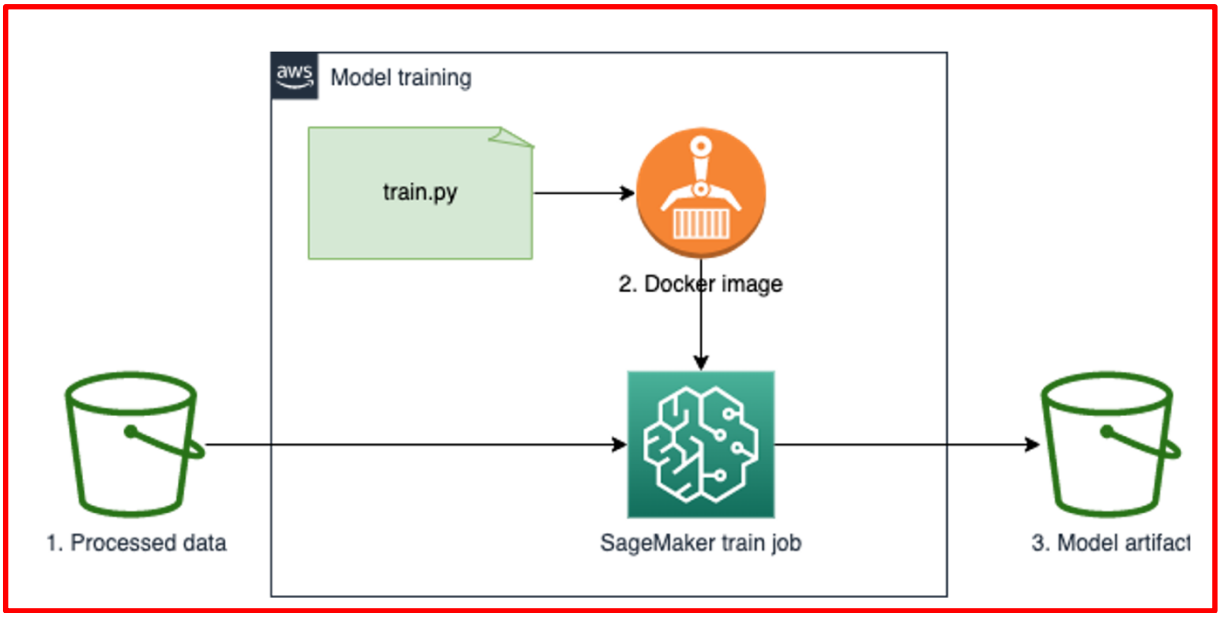


Figure4, model training and selection

To select the best model, I used the bayesian optimizer module for the hyperparameter tuning job in SageMaker and searched two parameters, n_estimatores and min_sample_split. Figrue5 shows that we have trained four models, and the own with the golden start is the best performing model, with an oval F1 score of 96%.

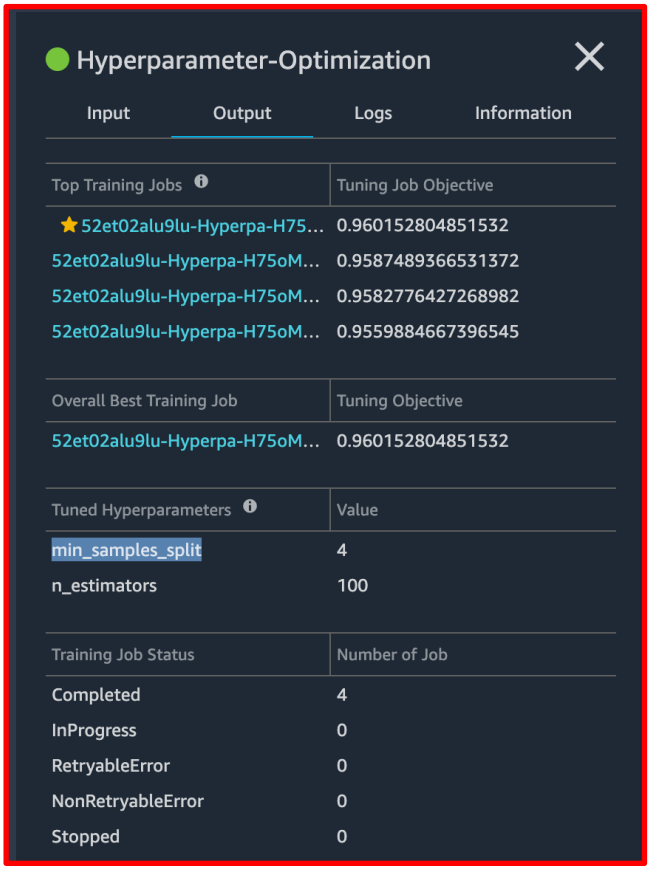


Figure5, Hyperparameter Optimization

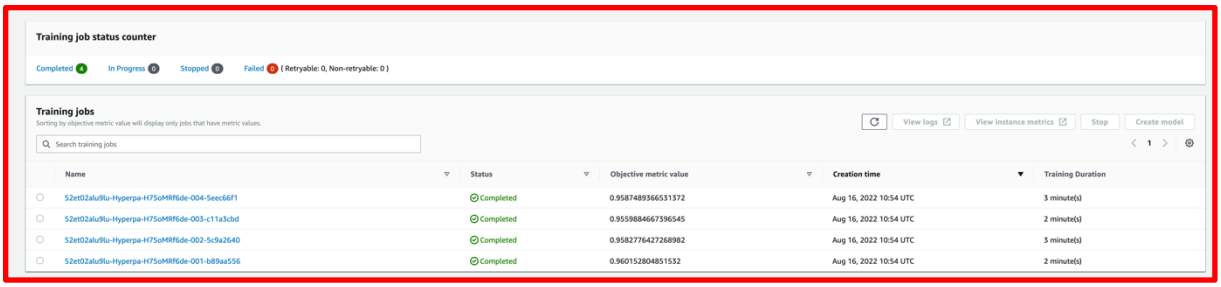


Figure 6. shows the training jobs

Model evaluation & Accuracy check

After identifying the best model performance, we need to ensure that the model performance is good enough for the model to be deployed or update the currently in-production SageMaker endpoint.

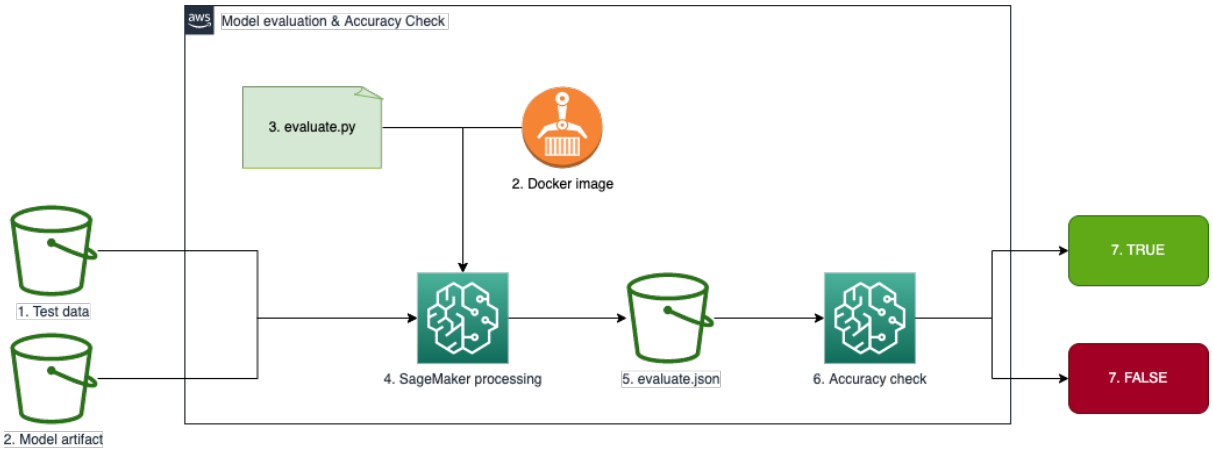


Figure7. model evaluation and check

Figure6, shows the steps that are part of the model evaluation workflow:

1. Load test data
2. Load the trained model
3. evaluate.py: contains the test logic: F1, edge cases, etc.
4. Save the output test result to evaluate.json
5. SageMaker SDK to check the evaluate.json
6. Provide a binary logic to move forward

The evaluation.py contains the code for:

1. Load test data to a DataFrame
2. Load pre-trained model from the training step
3. Make inference on the test data
4. Save the metrics such as Accuracy, F1, recall, etc. to the evaluate.json file

Endpoint Deployment

Figure 7 shows the diagram for the creating a SageMaker endpoint.

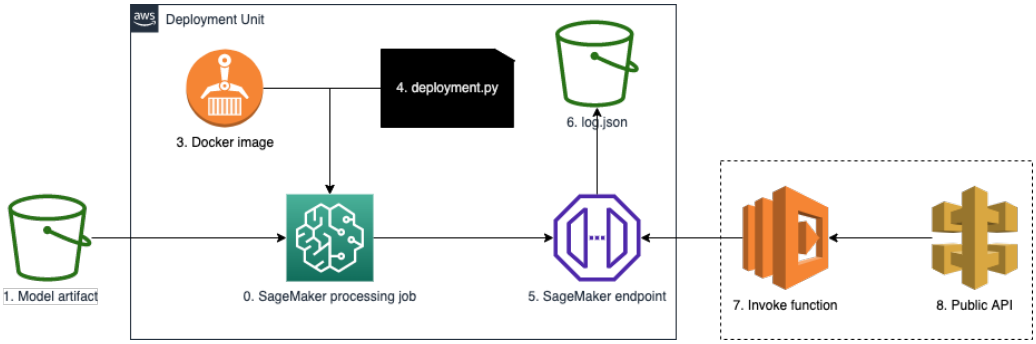


Figure 8. model deployment

1. Copy model artifacts to the container
2. Docker image with python endpoint
3. deployment.py contains all the deployment steps
4. SageMaker Endpoint
5. API logs saved to the S3 bucket for model monitoring
6. Lambda function that invokes the endpoint
7. API gateway to monitor and manage the API calls

Benchmark:

Due to the specific nature of this project with the suggested changes, there is no detailed study for a benchmark model performance. However, since we have six classes to identify in this dataset, if we assume that we have a random classifier to pick each of these classes at random with a normal distribution, we get to the accuracy of ~ 16%. The current f1-score after the current workflow is 96%.

References

1. Attention Is All You Need (31st Conference on Neural Information Processing Systems (NIPS 2017))(https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf)

2. Universal Language Model Fine-tuning for Text Classification [https://arxiv.org/abs/1801.06146]

3. Universal Sentence Encoder[https://arxiv.org/abs/1803.11175]

4. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [https://arxiv.org/abs/1810.04805]

5. LEDGAR: A Large-Scale Multi-label Corpus for Text Classification of Legal Provisions in Contracts [Don Tuggener, Pius von Däniken, Thomas Peetz, Mark Cieliebak] (https://aclanthology.org/2020.lrec-1.155/)