CSE 6748: Applied Analytics Practicum

Siddharth Gudiduri

GT ID: 903464761

Overview

For CSE 6748 project, I was tasked to build an end-to-end process that will automate contract reviews. Our goal was to develop a process mimicking an experienced attorney, but with enhanced speed and accuracy. Such a service will not only maximize document’s review but will also minimize risk involved with human error mitigating unacceptable or missing clause. Proof of concept Code for this project can be found at the following location: <https://github.com/sgudiduri/CSE-6748> . Note this code is just for your preview and not company’s active repo check-in as there any many details within the algorithm and check-in omitted. Contract Review Automation (CRA) is broken down into required parts and three bonus parts. Stages of CRA involves around Data Analysis, Model development, productionizing code, Creating Model API, Deploy to PaaS, Testing. Bonus tasks involve implementing minikube, Pocket base, Caching via Redis and Scaling concepts.

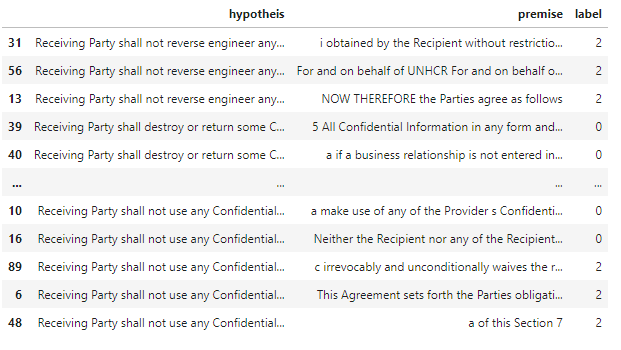
1. Data Analysis

Data is received json format. Training data contains 423 documents, and test 123 documents. structure shown below

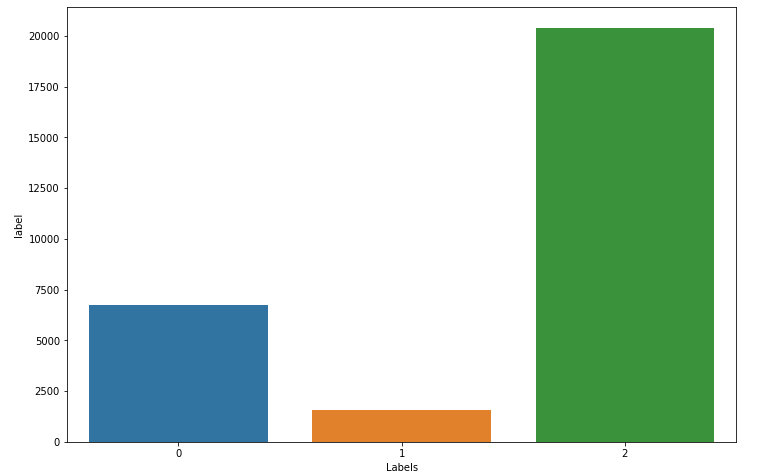


At the document level, we have “text” key that contains full document text, and “spans” key splitting text into a list of premises. The key “annotation sets” is a list containing multiple annotations for a given documents. At the annotation level, every key “nda-1”, “nda-2”, etc. is a hypothesis labeled either entails, contradicts, or is neutral to the given document. The “spans” key under each hypothesis is indexed the “spans” key at the document level. Example – “nda-1” entails the spans 1, 13, and 91. Here, span 1 at the document level corresponds to sentence text indexed between characters [25, 89]. The “labels” key describes the text sequence for each hypothesis.

Next step as part of data analysis is to process data and extract features, I will need for building. This feature engineer step was recorded and will be used in machine learning pipeline at a later step to process incoming data. Below is the extracted Tibble used for Model building.

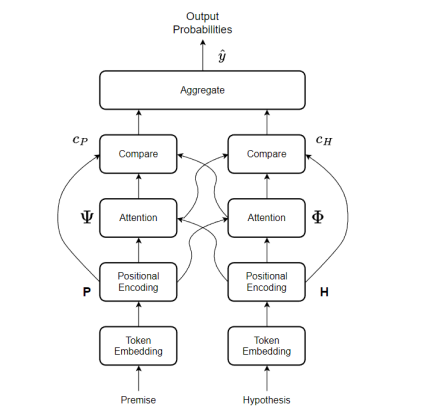


I found data to be imbalanced and this will play a role in selecting hyperparameters for model building. Below is screenshot which shows higher frequency of neutral cases followed by entitlement and contradiction. Code for research here:



2. Model Building

I modified one of the existing architectures Decomposable Attention Model with some minor changes like adding additional layers, dropouts, and attention mechanism.

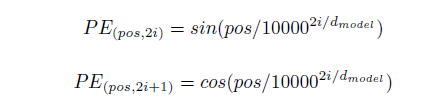


Token Embedding:

continuing I used Global Vectors (GloVe) embedding for word representation, an unsupervised learning algorithm for obtaining vector representations for words, with training performed on aggregated global word-word co-occurrence statistics from a corpus. To perform word embedding, we use GloVe 6B 100d.

Positional Encoding:

Below formula uniquely encodes information about the position of a token.



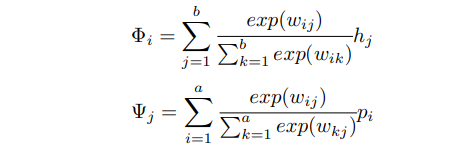
Here, d is the embedding dimension, pos is the position of the token in the sequence, and i maps to sin and cosine functions.

Attending:

We perform soft alignment of the premise and hypothesis essentially achieved by passing the input premise and hypothesis through a multi-layer perceptron and then computing soft attention weights

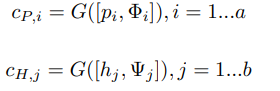


where F is the multi-layer perceptron with ReLU nonlinear activation that maps pi, hj to a hidden dimension space. This allows us to calculate the projection of the premise over the hypothesis. The intuition behind the alignment model is based on a bidirectional RNN used as an encoder and decoder



Comparing:

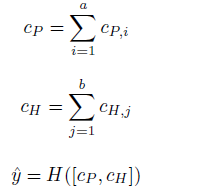
In the compare section, all the tokens from one sequence, with their corresponding weights are compared with a token in the other sequence.



The representation is the concatenation of premise token pi and the softly aligned weight representation for that token Φi . A similar operation is performed for the hypothesis as well. As the concatenation operation is performed along the embedding dimension, the multi-layer perceptron G maps input dimension equal to twice the embedding dimension, to the number of hidden units.

Aggregating:

The final step performed by the decomposable attention model is aggregating the information obtained from the comparison step. The information in the comparison vectors is aggregated through a summation operation. The summed-up results are now fed into a multi-layer perceptron H and are mapped to the number of outputs - Entailment, Contradiction and Neutral. Below are learnable parameters:



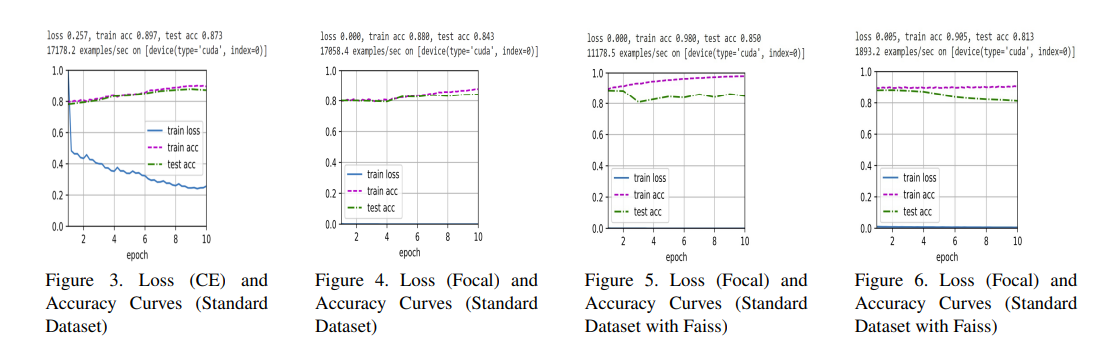
Focal Loss:

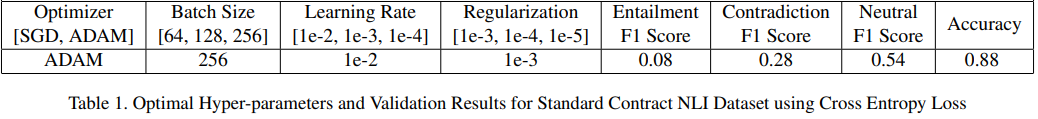
While training the decomposable attention model, we use focal loss as there exists class imbalance among the 3 classes - Entailment, Contradiction and Neutral.

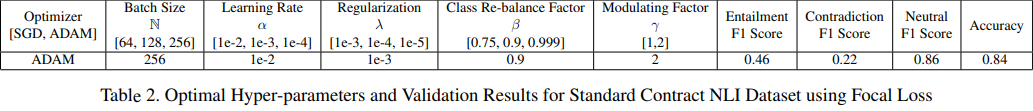


The β hyper-parameter can be tuned to perform reweighting. When pt is small and consequently, (1 − pt)γ is close to 1, then Focal loss becomes classic cross entropy, and would result in incorrect classification by the model. As the model adjusts its weights, Focal Loss scales down the contribution of easy examples during training and instead focuses on the harder examples, resulting in an improvement in prediction accuracy for the minor classes.

Results:



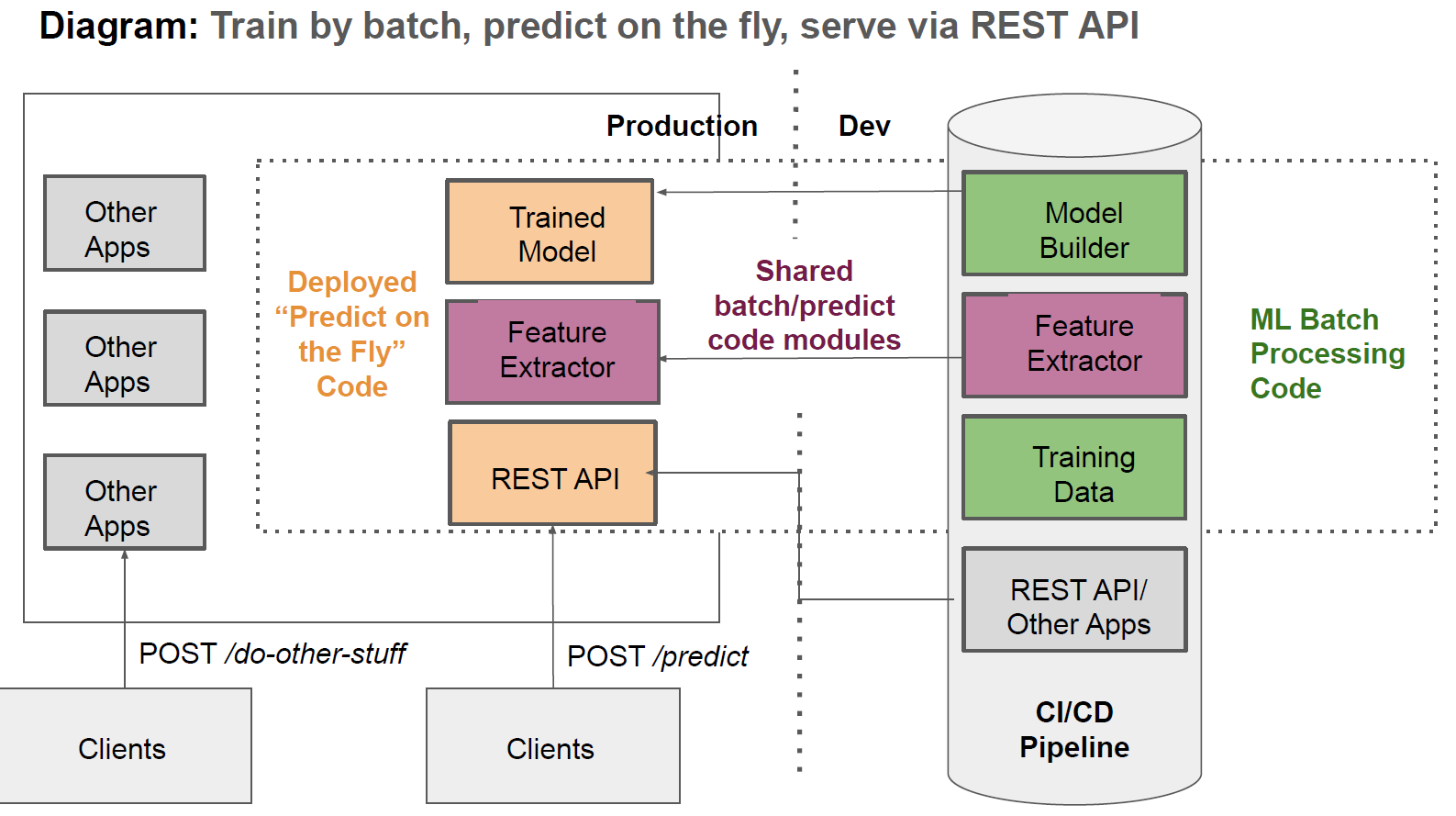




using the above hyperparameter we got the best precision, recall, f1-score along with accuracy and loss. After selecting model and hyperparameter our next task was to productionizing code.

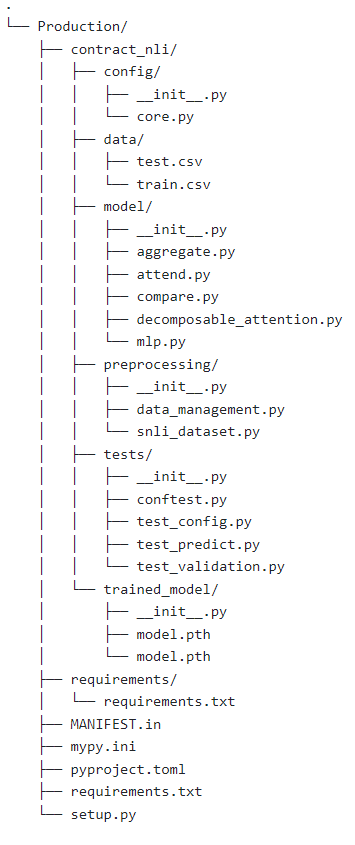
3. Architecture Component Breakdown

Next our goal is building the below architecture by creating model package, web api and CI/CD pipelines for package and api. Below is architecture breakdown

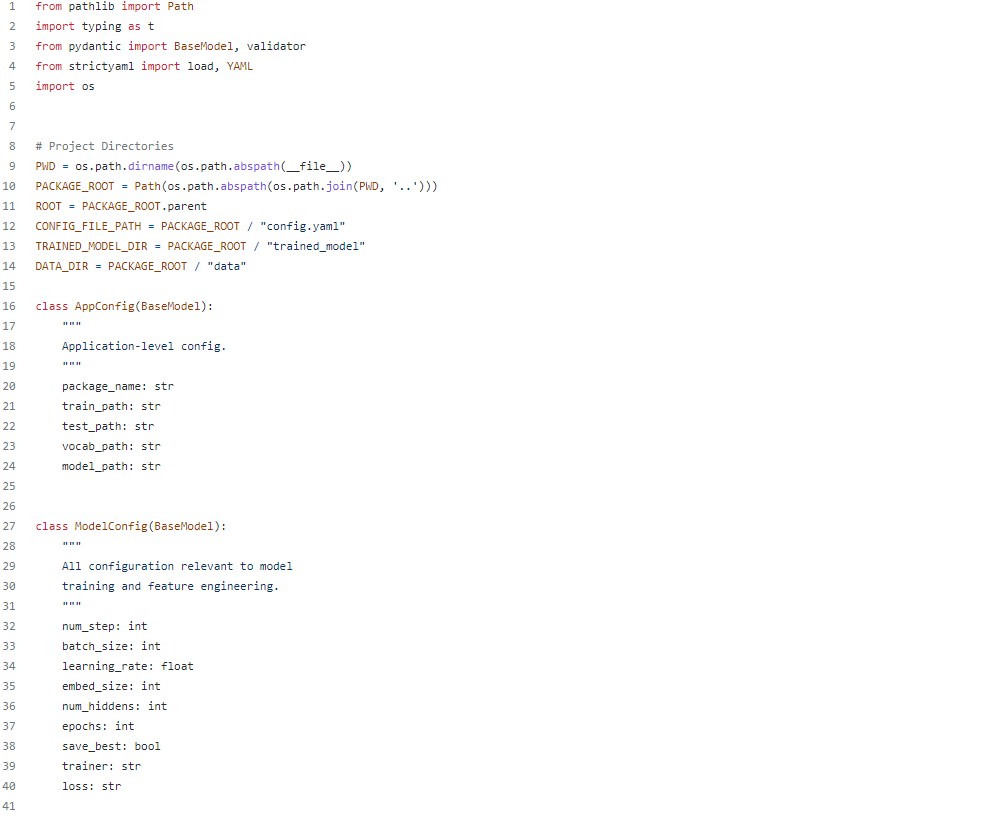


4. Production Model package

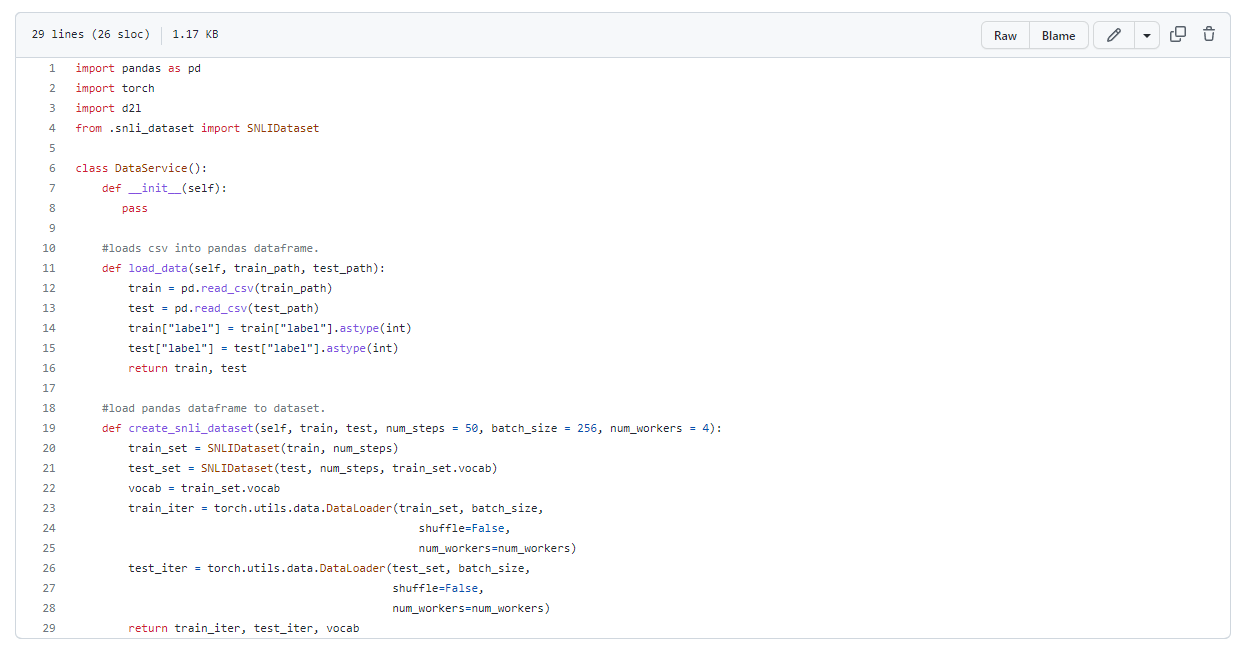
Continuing with project, next steps were to write production code designed to be deployed to end user. I continued to focus on Testability, Maintainability, Scalability, Performance and Reproducibility. Below is the package structure created breaking down research code into separation of concern components, meaning each module has single responsibility in doing its job. So config package, will only contain modules used for configuration. Testing package will only contain modules designed for testing. NOTE, company code has lot more unit tests and ensemble of models with various hyperparameters.



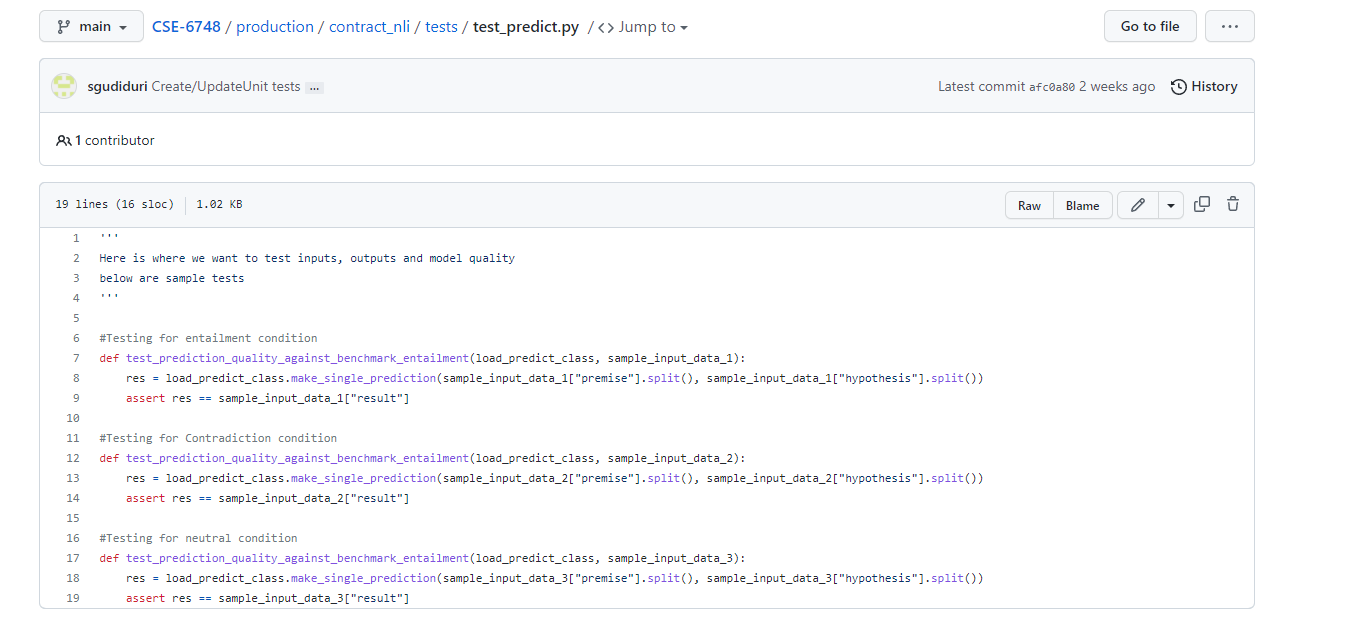
I created various model modules to build a deep learning package as you can see from the above folder structure. I have used dependency injection pattern, i.e. passing objects that objects need instead of creating them, helped in creating scalable and testable code. Below is an implementation of a python library called pydantic which makes configuration code easy and compiled into an object that can be passed into various parts of the application.

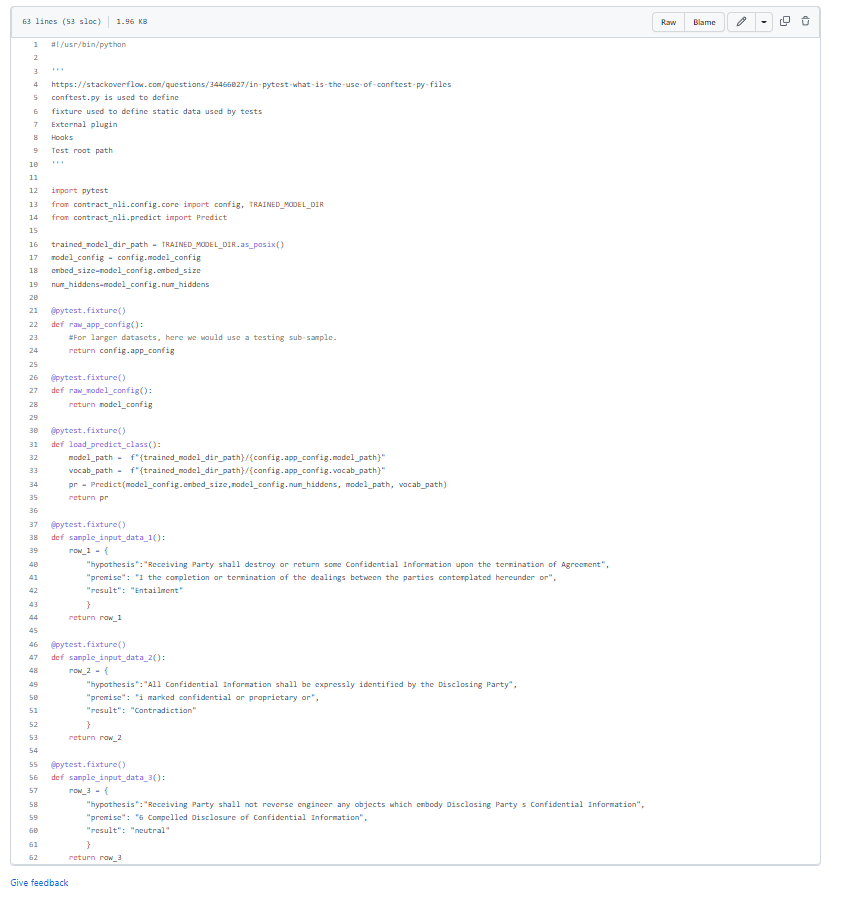


Below are some screenshots for Data Service class

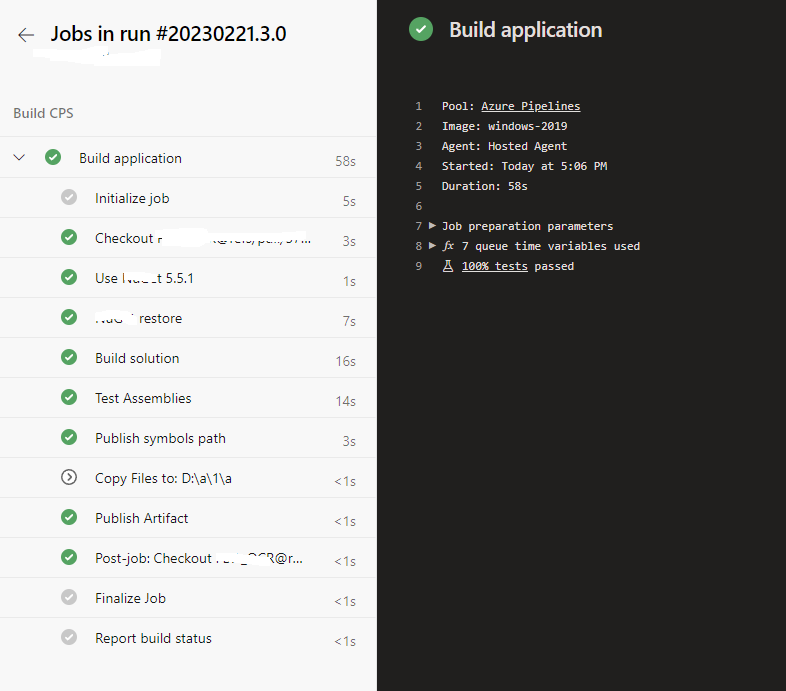


Below are some screenshots for testing code modules





Once I completed packaging production module, then started integrating Azure pipelines for CI/CD which stands for continuous integration, continuous delivery and continuous deployment. What this means is when a developer like me submits code for review and check’s in after approval, code goes through a process of building, testing, and publishing files to the private server. This is done so machine learning model can be integrated with a website or a web api, instead of creating monolithic application. Here is an example from my company pipeline when a feature has been checked in for this project.



Note, I created a similar example for this class as a POC before integrating with company code. This deployed on pypi is an experimental version and not the model package used at my company. Link can be found [here](https://pypi.org/project/contract-nli/)

Next steps:

To complete my project, I will need to implement Fast API to serve contract\_nli model in test. I will need to implement strategy to make single prediction and save a json file to make multiple predictions. Then I will containerize fast api and deploy as PaaS to company’s private server and so testing can begin in shadow mode and promote to clients. Given I have more time, I will work on implementing MiniKube(Kubernetes), Redis and Dynamic.