APPLIED LANGUAGE PROCESSING FOR QUESTION ANSWERING ON REAL-WORLD DATASETS

Michael Zeolla

Worcester Polytechnic Institute
Computer Science
mjzeolla@wpi.edu

Matthew Lund

Worcester Polytechnic Institute
Electrical and Computer Engineering
mtlund@wpi.edu

ABSTRACT

With the dawn of the internet, data has become an overgrowing commodity, and gaining value from this data is a practice with growing importance. This research study dives into the applicability of natural language processing, and deep learning for this task, and its role in parsing, understanding, and gaining value from this data. Specifically, this paper explores approaches to building an effective Question Answering pipeline on real-world datasets.

Index Terms— Deep Learning, Information Retrieval, Generative Models, Question Answering, NLP

Contributions

- **Michael Zeolla**: Label Prediciton, Retrieval, ElasticSearch, Generation, Writing/Reviewing, Dataset Creation, Data Processing, Methodology, Introduction, Experiments, Conclusions
- **Matthew Lund**: NER, Sentiment Analysis, EDA, Writing/Reviewing, Dataset Creation, Research, Data Processing, Related Work, Experiments

1. INTRODUCTION

Data has become one of the most valuable commodities since the advent of the internet, with many companies leveraging it to enhance the customer experience. Often, businesses create their own internal data warehouses to gather and store exclusive information on user interactions within their space. For instance, companies like Amazon, Yelp, and countless others collect sales data, along with detailed user feedback, to better understand customers. Additionally, the format of this data can take many shapes and sizes, such as numerical values, categorical entries and free text. This wealth of data allows companies to fine-tune their services and create more personalized experiences for their users by extracting key information from the data and making informed decisions.

It's important to note that each data format has its common techniques for extracting value, varying in its level of complexity. One of the most common methods for storing data is in the format of free text, as this format is similar to how people communicate every day and is the easiest to collect from real-world users. However, free text information, while convenient, is hard for machines to interpret and understand.

This research study aims to apply advanced techniques in Natural Language Processing (NLP) on real-world free text datasets, with the goal of creating an informative Question Answering (QA) pipeline. Such a pipeline would provide real-value to user-based companies, such as Yelp or Amazon, where user input/output plays a critical role in the experience and their likelihood to continue engaging on the site. A practical application would be a QA pipeline which is able to intake user questions about different businesses and provide accurate and valuable answers to the user. By applying a generally applicable dataset, such as the Yelp dataset, [1] to this task, the team aims to expose a common approach and methodology to apply similar pipelines for alternative datasets and demonstrate the value of such developments. The paper will explore the development process and applicable skills to build a resilient pipeline.

2. RELATED WORK

2.1. Natural Language Processing

Natural Language Processing (NLP) is one of the most critical and growing fields in relation to understanding and evaluating text inputs. This field involves the main practices for understanding and interpreting human language by processing data and representing it numerically. NLP enables machines to extract and gain meaningful value from text based information, and in some cases even generate human-like text. Overall, the field of NLP is ciritcal to any QA pipeline as human computer interaction (HCI) is often in the format of typed inputs from the user. By applying NLP and its underlying techniques to pre-process and vectorize free-text inputs from the user, the QA pipeline will be able to learn and respond.

2.2. Retrievers

Within NLP, retrievers refer to systems or models that help find relevant information from a large collection of documents, based on a query or input. Their main job is to retrieve pieces of information that are likely to contain

the answer to a user's question or match the query's context. With search engines, a retriever might be responsible for scanning billions of web pages and selecting the ones most relevant to the search term. In more advanced systems, like in question answering or information retrieval tasks, a retriever might narrow down the results by ranking or filtering the most relevant passages, which can then be processed by another model (like a reader or generator) to produce a more refined answer. [2]

2.3. Retrieval-Augmented Generation

Response generation refers to how models can create contextual and relevant replies to inputs like questions or statements [3]. Unlike how retrievers pull existing information, it will take the context of the query to generate a new response. The GPT models generated by OpenAI, for example, is often used to generate responses where the end goal is to produce text that both answers the original query and sounds like natural language.

Retrieval-Augmented Generation (RAG) is a technique that combines both retrievers and generation [3]. The idea behind RAG is that, instead of just generating a response from the model's internal knowledge, it first retrieves relevant information from a large dataset (like a search engine pulling up documents based on queries). Then, the model uses that retrieved information to generate a more informed and accurate response.

3. METHODOLOGY

Before building a QA pipeline, the first step is to create a document store, which will be responsible for holding all the related QA context paris. This document store, will be the main source of information to genearte an appropriate response for the user. It is imperative that the document store contains details about the types of questions users will be asking.

Once the document store is setup, the overall pipeline development can being. The QA pipeline process can be broken into a set of 3 sub-tasks, which help to generate relevant and applicable responses: (1.) Document Retrieval, (2.) Metadata Filtering, and (3.) Responding.

3.1. The Yelp Data Store

The team choose the Yelp dataset, which is a publicly accessible collection of businesses and review information, from the popular website Yelp. The raw Yelp dataset is broken into sub-datasets, with the applicable ones being the business and review datasets.

The business dataset contains details about registered businesses on Yelp, each with a unique id, location details, rating score, related categories, and much more. The review data set contains specific handwritten reviews about individual businesses, the primary fields being business id, review id, text, review rating score, and others. [1]. When joining these sub-datasets together based on the business-review keys, the resulting dataset is a set of hand-written reviews for each business, perfect of a QA pipeline. By retrieving and understanding the underlying business reviews, new user questions can be answered by sampling responses.

Overall, there are over 150K business entries within the data, along with 6.9 million reviews. From that, the team created a sample of 5.7 million unique business-review entries. The joined collection of business-review's act as the primary document store, with the review text being the document context, with the additional fields, such as location, and business categories, as metadata.

With over 5.7 million samples, information retrieval needs to be fast and efficient; otherwise users may waste too much time waiting for a response. To accomplish this, Elastic Search was applied as the indexing platform for storing and retrieving samples. Elastic Search is a memory efficient, cost effective, and fast solution for information storage and retrieval [4]. Relying on Elastic Search allowed the team to work with the entire set of Yelp business reviews, without reducing the dataset size.

3.2. Document Retrieval

Within the QA pipeline, when generating a response, it is common to pass a collection of related documents to the responding mechanism, so that it may make more informed conversation. However, this is commonly a pain point for many generative responders, as they can only accept a limited context-window [5], and it is often the case that many of the documents within a document store will be irrelevant to the user query Q. Processing unrelated documents will inherently consume more processing power and increase response time when interacting with a user. It is crucial to ensure that only relevant information is considered when sampling a response, as this leads to more context-rich answers and reduces computation time. This is where a retriever comes in, since a retriever is a system, typically a trained ML model, designed to identify and retrieve documents that are relevant to the original input based on context-query scoring.

There are two common approaches towards solving the document retrieval problem, (1.) classical machine learning, and (2.) deep learning. The former applies hand-structured data representation techniques, such as TF-IDF, to represent text numerically, then indexes, and retrieves documents. TF-IDF, which stands for Term-Frequency Inverse-Document-Frequency, is a statistical method used to evaluate the importance of a word within a document relative to a collection documents [6]. In the case of TF-IDF, each document is converted to a numerical array by comparing each term against its occurrence across the document set. This representation is know as a sparse representation, which has benefits for scalability, interoperability and simplicity [6]. However, the disadvantage of features like TF-IDF, is their inability to capture the semantic meaning of the context.

On the other hand, deep learning techniques, such as NLP and transformers, can play a major role in retrieval. Transformers, and their underlying LLM implementations, are capable of understanding the semantic meaning behind the input text, and embed the input Q as a N-dimensional numerical array [7]. This is known as a dense representation, where the underlying meaning behind each value is largely unknown to the human eye, but deep learning models are able to identify patterns within the data. Once a text is encoded using the model, the encoding can be compared to other encoded texts, to get the top K documents within the N-th dimensional space.

Both sparse and dense representations offer unique advantages and challenges, and the decision to use one over the other is primarily influenced by the specific task at hand and the nature of the datasets involved. In the case of the Yelp QA pipeline, the team conducted experiments with both types of retrievers, as seen in section 4.6.

3.3. Metadata Filtering

Metadata document filtering is the process of generating a subset of documents from a larger data source. Metadata filtering is different than the overall retrieval process, described in section 3.2, which filters documents directly based on their main content. Metadata filtering involves removing items based on a 3rd set of information, such as metadata for each document. Additionally, metadata filtering is specific to the underlying dataset, and the exact implementations may vary depending on the dataset and task. However, while the implementation may differ, the underlying idea remains just as beneficial.

Method #1: Categorical Filtering. As mentioned in section 3.1 the Yelp dataset includes key pieces of metadata for each business, and part of this includes a multi-label categorical field describing the type of business. For example, the labels "Restaurant", "Italian" and "Pizza" may be associated with a pizza parlor, while "shopping center" and "fashion" can be associated with a clothing store. Ultimately, these categorical values associated with each business provide a unique method for applying metadata filtering. Within the QA pipeline, the team applied these categories by filtering for just those related businesses, aka "documents", which are associated with the same tags as the user query Q.

While the businesses within the document store have categorical information baked into their data model, new user queries gathered at run-time will not. This is the primary hurdle associated with categorical metadata filtering, since to properly implement within a QA pipeline, the user query Q must be identified with one, or many, of the tags. To accomplish this, the team explored applying NLP techniques, along with neural networks, to correctly identify the categories applicable to each new user prompt. Category identification is a multi-label classification problem, which can be resolved by applying neural network architectures, such as transformers, to gather the hidden semantic meaning of each input Q, and associate it with the underlying input tags.

Method #2: Named Entity Recognition. Named Entity Recognition, also known as NER, is the process of extracting key named entities from a given prompt. [8] For example when looking at the Yelp dataset, information such as the business name, city, and state would be highlighted as organizations or locations. In terms of the QA pipeline, the user would enter a query like "Is Target good for buying clothes?", or "What is the best bar in Boston?". The pipeline would be able to retrieve entity types from the queries. For example, the second query about Bostom would recognize "Bostom" as a city, and search through all relevant information (reviews/ tips) for that tag. This aims to narrow down the scope and allow for a more reliable and contextually accurate answer.

Method #3 : Sentiment Analysis. Once the contextual information is found from the entity recognition, the pipeline needs a way of determining if the review is positive or negative for the subject in question. The Yelp dataset for instance gives the star ratings for the reviews of the businesses. Using this information alongside the reviews can help to train a sentiment model that would predict the score for different named entities in the prompt. For example, if the user mentions negative comments regarding a specific store, the pipeline can automatically filter out that business from the document store.

Method #4: Locational Filtering. Lastly, the team implemented location based filtering. Each business review pair within the Yelp dataset included location details about the business, such as the address, city, state, longitude and latitude. It stands to reason that users are likely only searching for businesses within their immediate location, therefore, a configurable radius was applied to filter documents based off a user's location.

In practice, location information can be collected via end-users without direct input. For example, if the chatting interface is deployed via a website, the location details can be tracked by the user's public IP address. However, for this sample QA pipeline, the location information was entered by users via the chatting interface at the start of the conversation. Additionally, improvements can be made by filtering documents based off conversational messages from the user via NER, but this was outside of the scope of a simple location filtering setup.

3.4. Response Generation

Once the document store has been filtered, either through basic document retrieval or metadata filtering, the resulting documents are ready to be used for generating a new response for the user. The overall process is known as Retrieval-Augmented Generation (RAG), which describes the process of generating new content based off an existing set of retrieved documents. RAG combines the strengths of traditional information retrieval with

langrage generation models, allowing systems to create contextually aware responses [9]. There are two primary models associated with constructing a response within a QA pipeline, (1.) Readers and (2.) Generators.

As discussed previously, one of the primary advantages of document retrieval and metadata filtering is that it helps reduce the number of documents before passing it to the large language model (LLM). By eliminating irrelevant or redundant documents, the LLM can focus on more relevant content, improving both computational efficiency and the quality of the generated response.

3.4.1. Readers

Readers are a type of neural network that excels at information extraction. This class of models are capable of taking contextual data as inputs, and a query and extracting the semantic meaning in reference to the query by applying NLP and attention mechanisms [10]. Readers are trained to predict the start and end positions of an underlying sub-text within the context that answers the query; and such a model has an inherit role in a QA pipeline. For example, if given the context "The USA was founded in 1776", and a user asked "when was the USA founded?", the model would likely return the index pairs associated with the phrase "1776", as a direct text reference.

Readers are effective models when the pipeline should reflect direct references, instead of potentially inaccurate generated content. The research team applied reader models as a configurable architecture that extracted the answers from the document set and applied this as a response to the user.

3.4.2. Generators

A common alternative to the extractive reader architecture is the generator neural network. Generators are probabilistic generative models, which are capable of sampling unique and never-before-seen response. Generators are appropriate to tasks where the responses should resemble more human-like speech. Moreover, generators are conventionalist, making them effective at modeling and understanding dialog; such features make them a good fit for an interactive QA pipeline [11].

The team applied generators as an alternative configuration for the responder segment of the QA pipeline. Instead of returning just the exact string extractions, generators were used to summarize the documents and produce new unique responses. Ultimately, the decision to use a generator or a reader is largely determined by how pipeline admins want their responses to be structured.

3.5. Evaluation Metrics

An important process for any deep learning task is evaluating the different models and experiments conducted during the study. Due to the large nature of the QA pipeline, and the wide range of sub-tasks, there were multiple evaluation metrics applied. The primary evaluation metric was F1-Score, which is commonly applied for classification models, especially when dealing with imbalanced datasets.

Some more task-specific evaluation metrics were the inclusion of BLEU, and ROUGE, which are common for language modeling tasks. These metrics were applied to the responder models to evaluate how effective they are at generating responses. BLEU (Bilingual Evaluation Understudy) measures the precision of n-grams in the new generated text compared to the original reference answer. On the other hand, ROUGE (Recall-Oriented Understudy for Gisting Evaluation) focuses on model recall and compares the overlap of words between the output and references [12] [13].

3.6. Putting It All Together

Now that the deeper steps for the QA pipeline have been established, how will they all be integrated with one another to properly resolve a user query? At run-time when receiving a new user input, Q, the query will be passed to the initial metadata filtering models, Categorical Filtering and NER Filtering, along with the locational details of the user for Location Filtering.

Once the user query Q is tagged and has its named entities identified, the new metadata search, which consists of the categories, user query, named entities and locational details, will be passed to the retriever. The retriever will query the document store, based on the afforested retrieval algorithm, to get the top K documents, aka "business reviews". Those top K documents are then passed to the responder model, either a generator or reader architecture, which will produce an appropriate response for the user. The generator will also have access to the input query, the calculated query metadata, and a reference to all previous chat messages when responding.

The final generated response is passed to a context evaluator to determine if the response is adequate, and if not, a new response will be sampled. Ultimately, the pipeline will return a new message from the "bot", which the user can continue to interact with; and this pipeline exhibits conversational tendencies by retaining historical information. An advanced breakdown of this process can be reviewed in Figure 1, which visualizes the steps.

4. EXPERIMENTS

4.1. Exploratory Data Analysis

Before the team started to work on forming the datasets for model training, the contents of the Yelp dataset needed to be explored to see the kind of data the team would need to process, and what data would be unnecessary. While parsing through the data, the team was able to gain insight on where the business data was mainly located, with states such as Pennsylvania, Florida and Louisiana being more than half of the data. In terms of businesses categories, many businesses categories

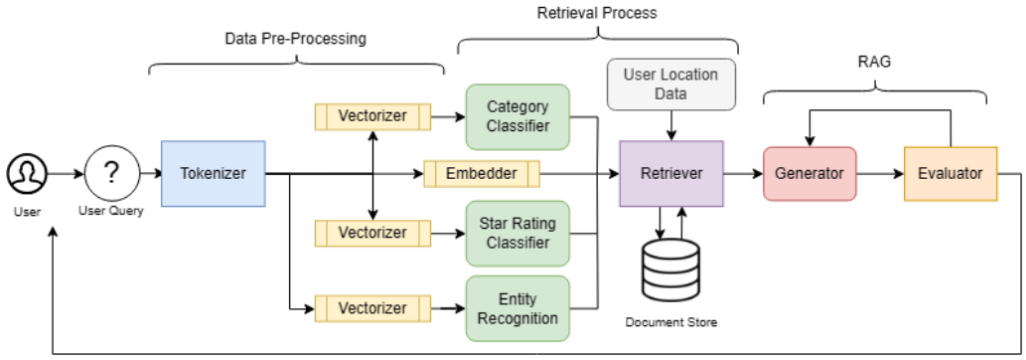

Fig. 1. Overall QA Pipeline

were repetitive, with many being a sub-set of a more descriptive input. For example, the most common tag was restaurant, with other tags like diner being less popular subsets of the parent, and these less-popular labels could be classified under their more generic parents. When the team looked at the review rating distribution, it was shown that there were significantly more 5 and 4 star reviews than 3,2, and 1 stars combined. Ultimately, these EDA results helped to guide the team when developing alternative datasets and models by providing key insights. In particular, the results found in the EDA helped to generate the new datasets mentioned in section 4.2, by visualizing the imbalance within categorical fields.

4.2. Applied Datasets

While the raw Yelp dataset is a treasure trove of information regarding businesses and their reviews, it is not in the correct format to properly accomplish the entire QA pipeline. Many of the sub-tasks require modified datasets to train effective models. To accommodate this, the team generated multiple new datasets more geared towards a specific task, with the Yelp dataset as the base.

Dataset #1: Business Categories. To train a model for business category classification, the original Yelp dataset must be slightly tweaked; specifically regarding the number of classes, and how often they appear. The raw dataset is highly imbalanced, with over 1,000 total classes, each with a varying amount of occurrences. This heavily imbalanced nature could cause a model to be ineffective, as it would be unable to generalize for all classes. Additionally, the internal class imbalance could cause the model to over fit/classify specific classes within the data. Therefore, the new Business Categories dataset was generated as a subset of the original categories, where only the top 20-100 classes were selected, and each class was sampled to have an equal distribution of occurrences, with a total of 40K samples.

Dataset #2: Question Answering. Effectively training a question answer responder requires a dataset that resembles the format of a QA pair, aka includes both a question and an answer. In the case of the Yelp dataset, there is no existing 1-to-1 relationship between a question and its respective answer, only an inherited business review. Therefore, to effectively train a responder an appropriate dataset is needed. The new Question Answering dataset comprised a sample of 20K entries, each including a question, answer, and original related text. To produce such a dataset the team used a combination of manual labeling and synthetic data generation. While manual labeling requires real people to label the dataset, synthetic data generation applies pre-trained LLMs to generate entries from the text, in this case providing the LLM with the review, and metadata, and prompting it to generate a new question/answer pair. [14]. The model applied to generate the new questions was the GPT-4-turbo model, but alternative models could produce an equally effective dataset. Each researcher reviewed a subset of the LLM-generated entries to help enforce standards.

Dataset #3: Entity Recognition. In order to effectively train the named entity recognition model, the team needed to a dataset data on both reviews with the Yelp businesses tokenized as well as the labels for each token. Some of the reviews upon inspection of the full Yelp dataset were shown to not include the name of the business in them. To address this, the team had sent a subset of the dataset as a batch to OpenAI through the use of the GPT-3.5-Turbo model; instructing the LLM model to generate an artificial review that keeps all of the detail of the original reviews whilst also injecting the name and location information of the business into it. These artificial reviews are then processed through a pre-made spaCy, an open-source software library for natural language processing [15] to create NER tags for each word in the review. At the same time, specific words, like the business names, were highlighted as an ORG (organization/company) tag. In total, the team was able to produce a full dataset of 10,000 records split with 70 percent for training and 15 percent for both validation and testing.

Dataset #4: Sentiment Analysis. As the team discovered in the exploratory data analysis (EDA), the original dataset is imbalanced regarding the number of star rat-

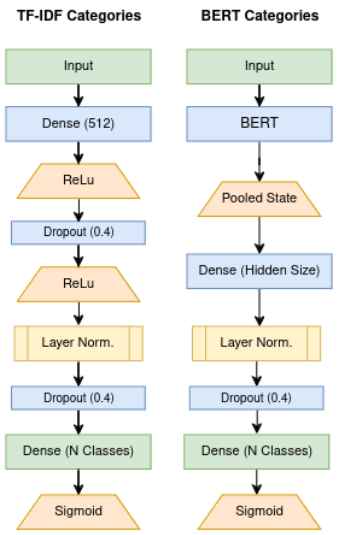

Fig. 2. Category Prediction Model Architectures

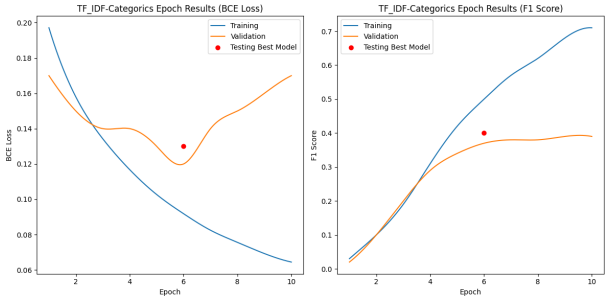

Fig. 3. TF-IDF-Categories Results

ings. For building a dataset to train a sentiment analysis dataset, the team used 50 chunks with 20K records, and within the data, the team added a new column. The new column was based on the review sentiment, broken down into the following labels: 0, 1, or 2; 2 if the review had at least 4 stars (positive), 0 if the review was 1.5 stars or less (negative), and everything in between was assigned 1 (neutral). The number of reviews was then normalized resulting in a balanced dataset of 53,872 records.

4.3. Categories Prediction

The first task undergone by the research team was identifying the categories associated with business reviews. The solution was to train a machine learning model, relying on deep learning and neural networks, to predict what multi-label classes were associated with text samples. The primary applied dataset was the custom Business Categories dataset. To represent the categorical output classes as a numerical feature a multi-label binarizer was fit to the text. Additionally, the input text was vectorized, but the vectorization technique applied depended on the model type, either TF-IDF, or token encoded.

4.3.1. TF-IDF-Categories

When tackling the categories prediction model the team initially started working with TF-IDF features, calculated via a TF-IDF vectorizer. The model trained was a simple neural network, which was a combination of many fully-connected layers, along with a range of activation functions, and normalization. The final classification head was a basic logits prediction layer, and logits were then converted to probabilities via sigmoid when applicable. Figure 2 describes the model architecture in detail.

When training the model, the team applied a batch size of 32 features, with a training iteration of 10 epochs, and a learning rate of 1e-4. The underlying results of this model were overall underwhelming, with the best validation F1 Score of 0.42, with a precision and recall score of 0.55 and 0.47 respectively. Based on the training curve, seen in Figure 3, the model largely overfits the training data. The team experimented with altered model architectures, but due to the inherent lack of semantic meaning the team ultimately opted to change their approach.

$$\text{BCE} = -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right] \quad [16] \quad (1)$$

$$\text{Focal Loss} = -\alpha_t (1-p_t)^{\gamma} \log(p_t) \quad [17] \quad (2)$$

4.3.2. BERT-Categories

The main model trained was a BERT based neural network for classification, denoted as BERT-Categories. BERT, is a multi-purpose, decoder based LLM, which is applicable to a large range of tasks when trained on a downstream dataset [11]. The base architecture of the BERT model was the component responsible for capturing the semantic meaning. Once the input is processed by BERT, the hidden state, which encompass the encoded meaning of the entire input, is passed to an output head. The output head applied a Dense layer with N total output nodes, where N is the number of classes in the dataset. The final output from the model was a sigmoid set of logits, representing the probability distribution.

The input for the model was a tokenized sequence based on the original text, which was collected by applying the generic BERT tokenizer, with a max length of 512 tokens. Due to the inherit nature of multi-label classification, BCE was chosen as the initial loss function for the model. Along with the loss, the primary evaluation metrics were the F1 Score, Hamming Loss, Precision and Recall, as they provide valuable insights for multi-label tasks on imbalanced datasets.

First Iterations. The first model implementation of this type, trained with a total epoch size of 15, and with a learning rate of 1e-4, produced a loss curve as seen in Figure 4. Based on the curve, the model is over-fitting even with such a small number of epochs, and as a result, the epoch size for future models was kept small. When

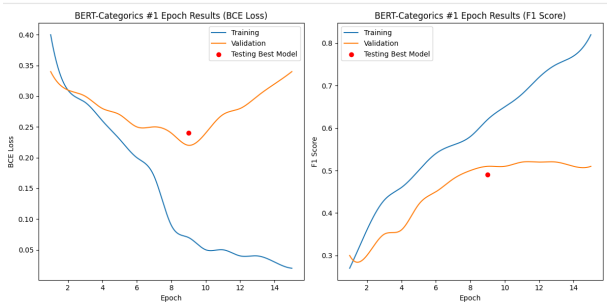

Fig. 4. BERT-Categories #1 Results

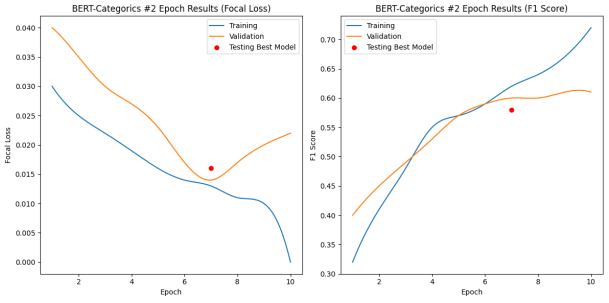

Fig. 5. BERT-Categories #2 Results

evaluating, an F1 score of 0.49 was reported, along with a BCE loss of 0.24 at the best model checkpoint. These results showed that a category classification model was possible, but, would require fine-tuning to perfect.

Second Iterations. During the second iteration of training for the BERT-Categories model the team experimented with a wide range of changes to improve model performance. Specifically, the team experimented with new loss functions, alternative model structure, varying learning rates (LR) and LR schedulers. The loss function was swapped from BCE to Focal Loss. Focal Loss helps to exhibit improved performance by down-weighting the loss assigned to well-classified items, allowing the model to focus more on difficult, misclassified classes [17]. Moreover, a dynamic warm-up learning rate scheduler based on the total number of epochs (10) and the training steps, was applied. It has been noted that training step warm-up schedulers help to boost BERT performance during earlier training stages [18]. Overall, this model produced significantly better results compared to the first iteration, as seen in Figure 5 and Table 1.

Despite improved performance, this model is still not performing at its peak, with the F1 Score only being 0.58. However, this may be due to poor training data that results in poor performance. As mentioned in section 4.2, the underlying text that is used to classify inputs is the review text, while the categorical labels are from the business data set. It is possible that joining the two datasets does not encompass true categorical samples, as the review text may be unrelated to the business tags. For example, in the case of review "y0gKAew2ek6JKPJOH_Hf-w", the business tags are ["Hair Salons", "Day Spas", "Nail Salons", "Eyelash Service", "Massage, Beauty & Spas"], yet the review, "Overpriced services. Management acts like Brandon is South Beach. Mediocre. Rude staff. Do yourself a favor –stay away" is unrelated to the specific tags for this business. These inadequacies within the dataset may cause poor performance, as the model cannot learns.

An alternative dataset that better comprises the categories, and their text, such as business descriptions, may result in a better overall model. However, in the grand scheme of the pipeline, inaccuracy with category classification is not a large concern, since filtering is inclusive, not exclusive. Therefore, a semi-inaccurate model that predicts more classes, but with less accurate results, still produces effective overall responses.

Model Type	F1	Prec.	Recall	HAM
TF-IDF Categories	0.40	0.57	0.33	0.08
BCE Bert Categories	0.49	0.62	0.42	0.05
Focal Bert Categories	0.58	0.65	0.53	0.03

Table 1. Categories Prediction Evaluation Breakdown

4.4. Sentiment Analysis

For the sentiment analysis model, a simple sequence classification model with an initial BERT base was implemented. With the dataset being balanced for neutral, positive, and negative reviews, training would help to ensure higher performance results. In terms of training, the model was trained through 7 epochs with evaluation and train batch sizes of 16. During training, it was clear that the model was overfitting to the training data as the training loss was steadily decreasing to a loss of 0.27, while the validation loss initially decreased, but began increasing to a final value of 0.88. BERT is a large model, which may have overfit due to the simplicity of the task.

When applying the model to the testing dataset, the team saw that the model performed well, as shown in table Table 2. The model was able to detect negative sentiment effectively, and decent result for the alternative classes. The model received sub-par precision in neutral sentiment and sub-par recall for positive reviews. Additional model fine-tuning may lead to better results. However, the model performed quite well and proved to be an effective addition to the pipeline.

	Precision	Recall	F1-Score
Negative (-)	0.82	0.86	0.84
Neutral	0.56	0.67	0.61
Positive (+)	0.74	0.56	0.64
Macro Avg.	0.71	0.69	0.69

Table 2. YERT-SENT Metrics on Testing Data

4.5. Entity Recognition

During training, the team started with a simple BERT model for token classification and made slight modifications to fine-tune it. With the use of spaCy, the team was able to take the artificial reviews created and extract information such as the business names, locations, and other entities that would be found in reviews or metadata locations of the full dataset.

The input for the model was the artificial reviews, alongside the list of entity dictionaries created through spaCy that contained information such as the start characters, end characters, and labels. When gathering the number of labels for the label2id field of the pre-trained BERT-base-case model, the team discovered that the spaCy pre-processing to create referential entities was able to grab 18 kinds of labels [19], excluding non-entities, aka common words that would not fit into a category. Examples of some of the categories that were discovered are (including business names), geopolitical entities, locations, people (a certain person that worked at the business or mascot for example), works of art, languages, quantities, and more. The reviews were tokenized and either padded/truncated to their full length with offset mappings provided. Once the data was mapped to the appropriate format, the model was trained with training and validation batches of 32 for 15 epochs. The team had to apply 16-bit floating point precision through the use of CUDA cores to train a model effectively. From there, the team decided to save the model as YERT-NER (Yelp-Bert-Ner) and evaluate based on precision, recall, and f1-score, using a test dataset.

Metric	Metric Score
Precision	0.717
Recall	0.692
F1 Score	0.699

Table 3. Evaluation Metrics from YERT-NER

Overall, the YERT-NER model developed produced decent results as shown in Table 3, specifically regarding precision and F1 score. However, the recall is slightly lower than expected; and this may be the result of the model missing entities in the testing dataset. These scores can be explained by the extra addition of the business names that the model may not be the most familiar with. This can be the result of local family businesses in the dataset or the dataset being too small due to the limited batch sizing, a limitation placed by the GPT-3.5-turbo model used to generate the dataset. In practice, this entity model was able to grab enough of the information the team would need for the QA pipeline to get contextually accurate results.

4.6. Applied Document Retrieval

When experimenting with different implementations of document retrieval, the team largely opted to apply pre-trained models. Specifically, two models were chosen for their effectiveness and ease of use: (1) BM25 and (2) DPR. BM25, a classical probabilistic retrieval model, has long been recognized for its solid performance in information retrieval tasks [20]. The simplicity of BM25 comes from the fact that it uses human-understandable features to encode documents within the store. On the other hand, DPR, named after Dense Passage Retrieval, leverages neural networks to map documents and queries into dense vectors, enabling context-rich retrieval [21].

To enable DPR, the ElasticSearch documents require an additional field, known as an embedding. Embeddings are vector representations of text, generated by neural networks, transformers, and this is what allows DPR to capture the semantic meaning. When calculating embeddings for the base pipeline, the ”BAAI/bge-large-en-v1.5” model was used to embed the documents.

Unfortunately, the team lacked a dataset for fine-tuning a custom document retrieval model, yet, the off-the-shelf models were still highly effective in practice. Both of these models were enabled as configuration operations when running the training pipeline.

4.7. Fine-tuned Responders

The last part of the pipeline is producing a proper response for the user; and as mentioned in section 3.4 there are two main approaches, readers and generators. The team opted to apply the QA dataset to a series of pre-trained LLM's for QA, and also fine-tune a response model via custom training specific for the Yelp data.

Readers. The reader evaluated was a pre-trained Deepset Roberta model, originally trained on the SQUAD dataset. This off the shelf reader was capable of extracting revenant information from the input context, which in this case was a review, and provided an applicable answer. Table 4 shows the performance breakdown for each model, with the reader architecture performing quite well, with an F1 score of 0.56. In practice, it would be feasible to fine-tune this base model on an alternative dataset, which could lead to better results. However, the focus of the QA pipeline was geared more towards generators as they offer more human-like responses.

Generators. As mentioned previously, generators are the favored response model for the QA pipeline, as they exhibit speech patterns more applicable to real conversations. This is largely due to the fact that these model types solve the Casual Language Modeling task, meaning they are trained to sample new statements. This is particularity useful for a QA pipeline which resembles a conversational input/output.

The first generator evaluated was the off the shelf Google-T5 small model, which at this point was not

Model	Type	F1	BLEU	ROUGE
Deepset Roberta	Read.	0.56	0.40	0.61
T5-Small	Gen.	0.32	0.19	0.36
Fine-T5-Base	Gen.	0.62	0.42	0.65

Table 4. QA Evaluation Breakdown

fined-tuned for the Yelp dataset. The small model was chosen, as opposed to the "large" alternatives, because due to these models larger size, the team experienced CUDA memory issues. The small model provided a good balance between performance, and compute requirements. The basic Google T5 small model performed the worst compared to other responder types, as seen in Table 4. Yet, the model provides a good baseline to compare other generators. Additionally, this was to be expected, as the model performs best when fine-tuned.

The team then fine-tuned an additional T5 model on the Yelp dataset to asses the performance improvements between the two models. This model was trained for a total of 20 epochs, with model performance being evaluated every 5 epochs. Ultimately, this model was capable of adjusting to the new QA dataset, and learned to produce more accurate responses. This fine-tuned model produced the best results out of the responder models tested, and was primarily applied to the QA pipeline. The final evaluation scores for this model were F1-Score of 0.62, BLEU score of 0.42 and a ROUGE score of 0.65.

It is important to note that the QA pipeline also passed previous messages within the chat history as context to the generators at run-time. This allowed them to be more accurate and effective at long-term discussions. Wile this did not impact their training and evaluation, it would give generator based models an edge over readers. This is just one of the advantages that generators have over their reader counterparts, which is not demonstrated in the evaluation results in Table 4.

4.8. Context Evaluation

The final step in the pipeline process is evaluating the outputted response for contextual relevance toward the original input user query. By evaluating the response before returning to the user, the pipeline can produce better results, by re-sampling a new response if necessary. To accomplish this, the researchers implemented a pre-trained LLM specifically for context evaluation.

The ContextEvaluator pipeline takes an input query, a context set, along with a response, and calculates a score measuring how accurate/relevant the response is to the query, about the context. The threshold for a response's accuracy was a configurable parameter within the pipeline, yet, the team noticed a threshold slightly above average produced an effective trade-off between correctness and compute time. It is possible to train a custom model capable of the same functionality as the pre-trained evaluator, however, easily accessible off-the-shelf models offer the same performance, without requiring time and computing training.

5. CONCLUSIONS

The results presented in this research study show the ease of use and effectiveness of implementing a custom QA pipeline for real-world datasets. Many recent advancements in the fields of NLP and Deep Learning have resulted in large improvements and ease of access to scalable and efficient models. These models apply to a wide range of tasks and can be further fine-tuned and improved for specific datasets or tasks. Additionally, this paper has demonstrated how training sub-models help narrow down answers and enhance the pipeline's response process. By applying the techniques and practices studied and suggested in this paper, any person, or organization, can create a QA pipeline with little overhead and complexity.

6. FUTURE WORK

The team showed that building an effective QA pipeline is possible and feasible for a wide range of datasets. However, that does not mean that improvements cannot be made to what has already been built. The team recommends that future research delve deeper into fine-tuning the models trained, and look outwards to alternative architectures. For example, while effective, the basic generator architectures trained in section 4.7 are not perfect, and alternative architectures may prove fruitful. GPT-based models, such as the ones by OpenAI, may provide superior results, but at the cost of computing power. Additionally, further investigating the applicability of this QA pipeline on alternative datasets may shed light on the generality of the implementation. Ultimately, these developments may lead to a more effective and better QA pipeline; which is more applicable to everyday usage.

7. REFERENCES

[1] Yelp, "Yelp dataset challenge," `https://www.yelp.com/dataset`, 2018.

[2] Devendra Singh Sachan, Mike Lewis, Dani Yogatama, Luke Zettlemoyer, Joelle Pineau, and Manzil Zaheer, "Questions are all you need to train a dense passage retriever," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 600–616, 06 2023.

[3] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, Eds.

2020, vol. 33, pp. 9459–9474, Curran Associates, Inc.

[4] Clinton Gormley and Zachary Tong, *Elasticsearch: A Distributed, RESTful Search and Analytics Engine*, O'Reilly Media, Sebastopol, CA, 2015.

[5] Alec Radford, Jong Wook Kim, Chris Hallacy, Alane Lim, David H. K. Lee, Dario Amodei, and Ilya Sutskever, "Learning transferable visual models from natural language supervision," *Proceedings of NeurIPS 2021*, 2021.

[6] Robert Salton and Michael J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, 1983.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Proceedings of NeurIPS 2017*, 2017.

[8] James Hammerton, "Named entity recognition with long short-term memory," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 172–175.

[9] John Smith, "An introduction to retrieval-augmented generation," *Journal of AI Research*, vol. 15, pp. 120–135, 2023.

[10] Xinya Du, Jun Hou, Wen tau Yih, Hongyu Wu, and Jianfeng Gao, "Deep learning for question answering," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP 2016)*. 2016, pp. 414–424, Association for Computational Linguistics.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2019.

[12] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the association for computational linguistics*, 2002, pp. 311–318.

[13] Chin-Yew Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[14] Yingzhou Lu and et al., "Machine learning for synthetic data generation: a review," *arXiv preprint arXiv:2302.04062*, 2023.

[15] Kamal Khumar, "Text summarization using spacy," *Analytics Vidhya*, 2020.

[16] Claude E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[17] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, Bharath Hariharan, and David Dollár, "Focal loss for dense object detection," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2018.

[19] Neha Heda Hemlata Shelar, Gagandeep Kaur and Poorva Agrawal, "Named entity recognition approaches and their comparison for custom ner model," *Science & Technology Libraries*, vol. 39, no. 3, pp. 324–337, 2020.

[20] S. E. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[21] Vasily Karpukhin and et al., "Dense passage retrieval for open-domain question answering," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 675–684.