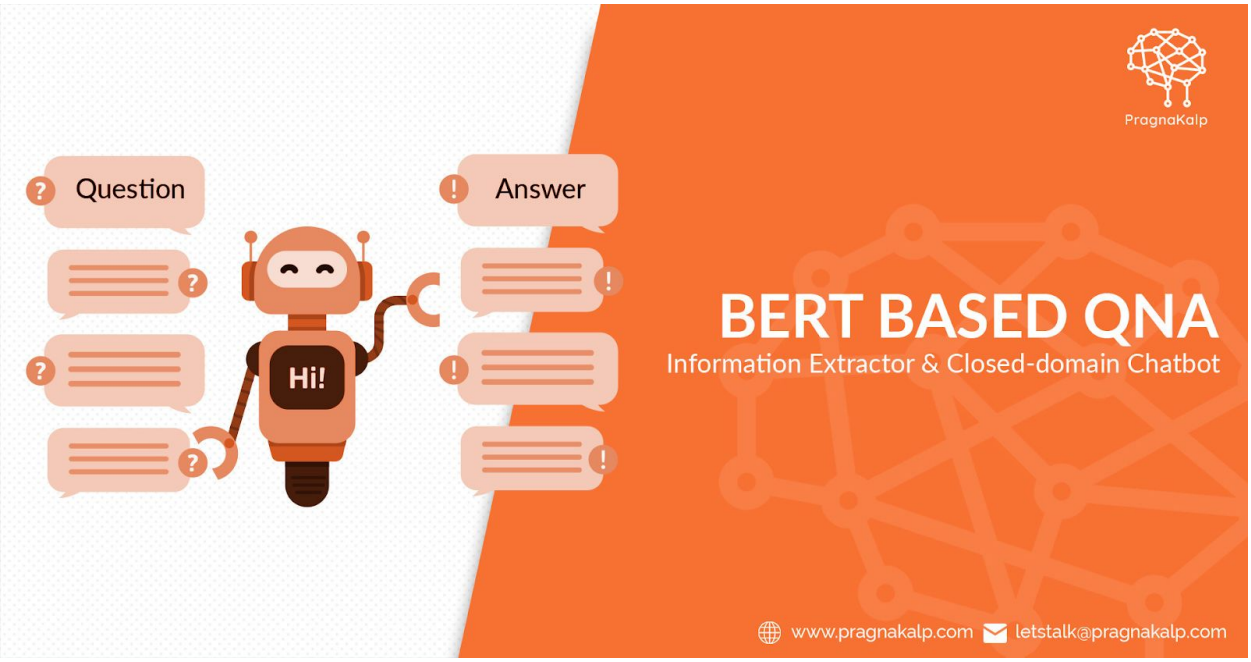
Udacity Machine Learning Nanodegree Capstone Project Report

Google Quest Q&A labelling

By Manav Nitin Kapadnis



Introduction

The main aim behind taking this project was to expand my knowledge in the Natural Language Processing Field,mainly in the Question and Answering system,so that I can create more projects such as Chatbots using NLU and Rasa.Moreover,I hadn't touched this field before,but now after completing such a complex project,I feel like I have accomplished something at the end of my Nanodegree. Let's begin the report,it consists of five Parts:

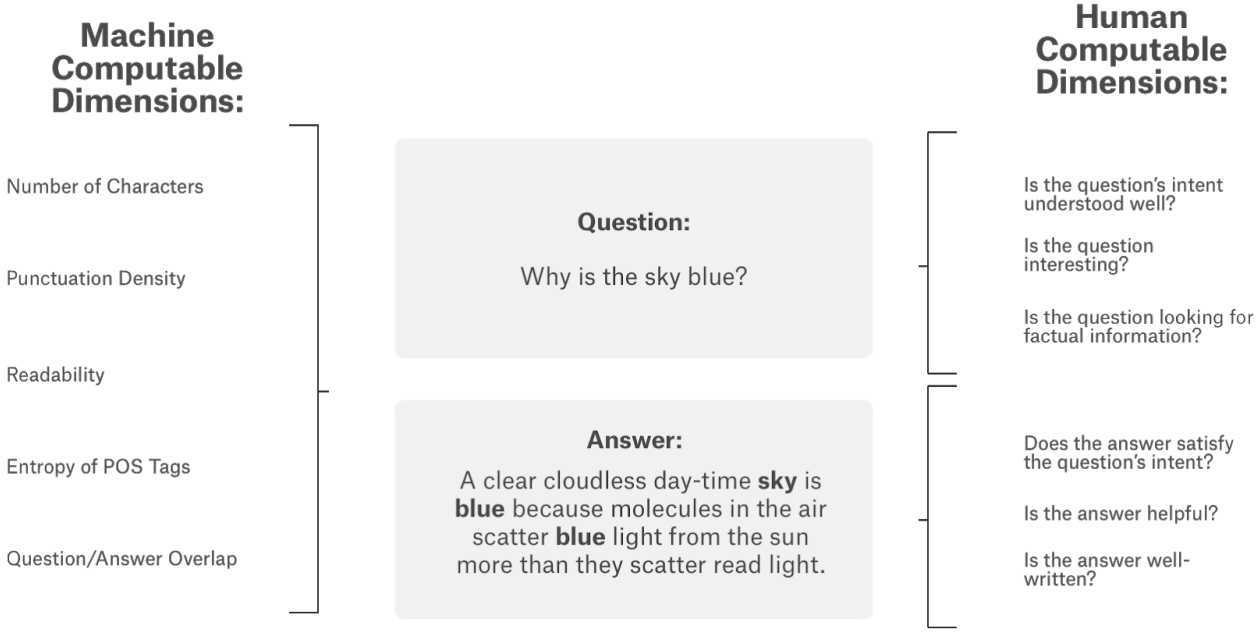
1. **Defining the Problem** - A formal definition and an explanatory one
2. **Analysing the Dataset** - Observations from EDA
3. **Explaining the Working Theory of BERT**
4. **Data Preprocessing**
5. **Implementation of Bert -** Explaining the functioning of Bert in this task
6. **Benchmark score -** This data is extracted from the Leader board of competetion

7. **Explanation Of Evaluation Metric -** It's a relatively complex metric so requires an explanation
8. **Results and Conclusion -** Final outcome of the report
9. **Comparison of our output with other people solution on Leaderboard.**

Defining The Problem

Computers are really good at answering questions with single, verifiable answers. But, humans are often still better at answering questions about opinions, recommendations, or personal experiences.

Humans are better at addressing subjective questions that require a deeper, multidimensional understanding of context - something computers aren't trained to do well...yet.. Questions can take many forms - some have multi-sentence elaborations, others may be simple curiosity or a fully developed problem. They can have multiple intents, or seek advice and opinions. Some may be helpful and others interesting. Some are simple right or wrong.



Unfortunately, it's hard to build better subjective question-answering algorithms because of a lack of data and predictive models. That's why the CrowdSource team at Google Research, a group dedicated to advancing NLP and other types of ML science via crowdsourcing, has collected data on a number of these quality scoring aspects.

In this dataset, you're challenged to use this new dataset to build predictive algorithms for different subjective aspects of question-answering. The question-answer pairs were gathered from nearly 70 different websites, in a "common-sense" fashion.

The what you read was a formal explanation of the problem,roughly the problem translates to this,you are given a question title,a question ,an answer,the source of answer,and a question id,which was not useful and so was dropped while preprocessing.On the basis of these information,you have to predict the probabilities of thirty different labels,which are namely -

Target labels

1. question_asker_intent_understanding
2. question_body_critical
3. question_conversational
4. question_expect_short_answer
5. question_fact_seeking
6. question_has_commonly_accepted_answer
7. question_interestingness_others
8. question_interestingness_self
9. question_multi_intent
10. question_not_really_a_question
11. question_opinion_seeking
12. question_type_choice
13. question_type_compare
14. question_type_consequence
15. question_type_definition
16. question_type_entity
17. question_type_instructions
18. question_type_procedure
19. question_type_reason_explanation
20. question_type_spelling
21. question_well_written
22. answer_helpful
23. answer_level_of_information
24. answer_plausible
25. answer_relevance
26. answer_satisfaction
27. answer_type_instructions
28. answer_type_procedure
29. answer_type_reason_explanation
30. answer_well_written

This could be easily(not so easily) implemented by the help of Bidirectional Encoder Representations from Transformers or BERT .

Analysing the Dataset

Some of the Observations made by analyzing the Data are as follows:

- The train and test size are fairly small enough,(which was one of the reasons why the competition organizers allowed the use of external data)
- The train dataset contains merely 6079 data points,whereas the test set has around 476 predictions which need to be submitted.The reason why i consider this dataset small is because of the modern standards and the complexity of problem,due to which my expectation was around 10000 to 15000 train data should have been given.
- The dataset contained zero missing values,which was very impressive
- The train as well as test dataset hosts were mainly dominated by two websites namely, stackoverflow and stackexchange
- The questions were mainly categorised into five different categories namely Technology , stackoverflow,culture science and life arts.
- There were a lot of duplicate questions,their order is given in the picture below:

question_title	
What is the best introductory Bayesian statistics textbook?	12
What does mathematics have to do with programming?	11
Important non-technical course for programmers?	11
How to prevent the "Too awesome to use" syndrome	9
Another instructor is pushing me out of the classroom right after my class ends	7
No sound in Ubuntu except at log in	7
How do I deal with a slow and undedicated colleague in the team?	7
What are the benefits of owning a physical book?	7
House rules to make the cloister less of a game winning tile in Carcassonne?	6
Making sure that you have comprehended a concept	6
hide javascript/jquery scripts from html page?	6
What is the best place to start Warhammer 40k?	6
Is pretending to want to trade before playing a monopoly card objectionable?	6
Does "so far, so good" carry a negative connotation?	6
Good travel games for two players, especially for playing on trains?	6
Effects of nuclear explosions in space?	6
Is there any performance difference between ++i and i++ in C#?	6
When should a supervisor be a co-author?	6
Isn’t the FAQ label obsolete by now?	6
Should I tell other interviewers where else I've interviewed?	6
CASTING attributes for Ordering on a Doctrine2 DQL Query	5

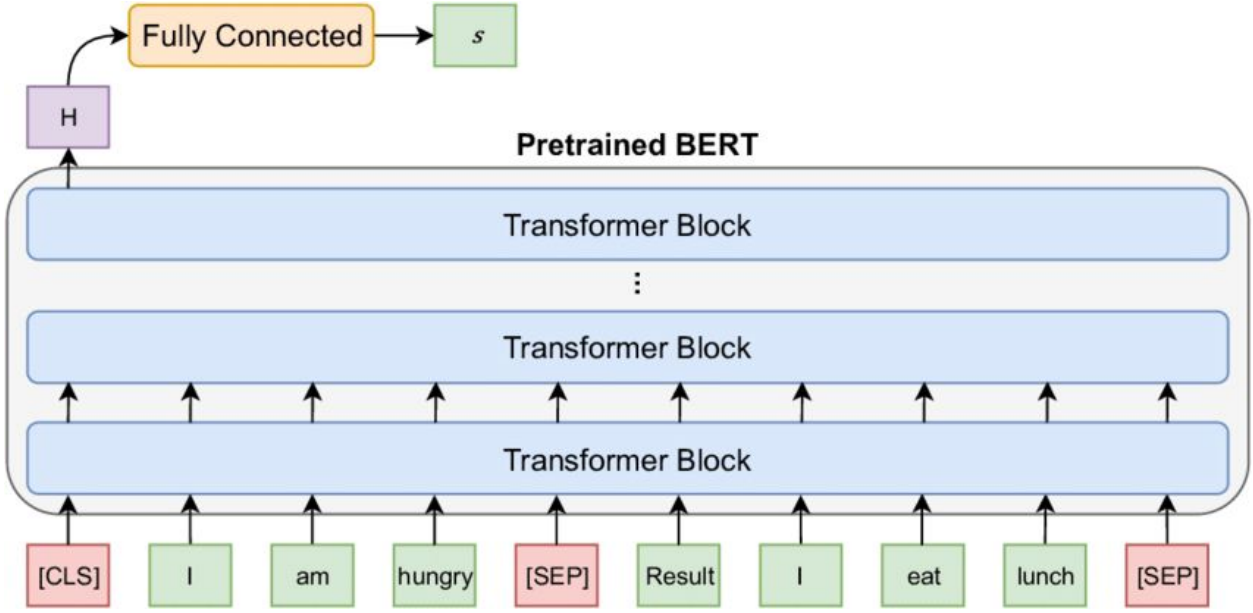
- There were a lot of repeating differently ordered questions too,those can be seen in the EDA notebook named as Quest EDA.

Working Theory Of Bert Model

Bert was proposed by Google AI in late 2018 and since then it has become state-of-the-art for a wide spectrum of NLP tasks.

It uses an architecture derived from transformers pre-trained over a lot of unlabeled text data to learn a language representation that can be used to fine-tune for specific machine learning tasks. BERT outperformed the NLP state-of-the-art on several challenging tasks. This performance of BERT can be ascribed to the transformer’s encoder architecture, unconventional training methodology like the Masked Language Model (MLM), and Next Sentence Prediction (NSP) and the humungous amount of text data (all of Wikipedia and

book corpus) that it is trained on. BERT comes in different sizes but for this challenge, I've used *bert_base_uncased.*



The architecture of bert_base_uncased consists of 12 encoder cells with 8 attention heads in each encoder cell.

It takes an input of size 512 and returns 2 values by default, the output corresponding to the first input token [CLS] which has a dimension of 786 and another output corresponding to all the 512 input tokens which have a dimension of (512, 768) aka pooled_output.

But apart from these, we can also access the hidden states returned by each of the 12 encoder cells by passing output_hidden_states=True as one of the parameters.

BERT accepts several sets of input, for this challenge, the input I'll be using will be of 3 types:

- **input_ids**: The token embeddings are numerical representations of words in the input sentence. There is also something called sub-word tokenization that BERT uses to first breakdown larger or complex words into simple words and then convert them into tokens. For example, in the above diagram look how the word 'playing' was broken into 'play' and '##ing' before generating the token embeddings. This tweak in tokenization

works wonders as it utilized the sub-word context of a complex word instead of just treating it like a new word.

- **attention_mask**: The segment embeddings are used to help BERT distinguish between the different sentences in a single input. The elements of this embedding vector are all the same for the words from the same sentence and the value changes if the sentence is different.
  Let's consider an example: Suppose we want to pass the two sentences "I have a pen" and "The pen is red" to BERT. The tokenizer will first tokenize these sentences as:
  ['[CLS]', 'I', 'have', 'a', 'pen', '[SEP]', 'the', 'pen', 'is', 'red', '[SEP]']
  And the segment embeddings for these will look like:
  [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
  Notice how all the elements corresponding to the word in the first sentence have the same element 0 whereas all the elements corresponding to the word in the second sentence have the same element 1.

- **token_type_ids/segment ids:** The mask tokens that help BERT to understand what all input words are relevant and what all are just there for padding.
  Since BERT takes a 512-dimensional input, and suppose we have an input of 10 words only. To make the tokenized words compatible with the input size, we will add padding of size 512–10=502 at the end. Along with the padding, we will generate a mask token of size 512 in which the index corresponding to the relevant words will have 1s and the index corresponding to padding will have 0s.

Data Preprocessing

Since most of the text cleaning and processing was done by **BertTokenizer ,**I didn't do much extra cleaning apart from that.The BertTokenizer automatically performs the functions of lower casing the word,removing the stopwords and unnecessary punctuations.So the first step was to load the Bert Tokenizer from the bert-base-uncased-vocab.txt file.

The next step was to create ***input_ids, attention_masks,*** and ***token_type_ids*** from the input sentence.

The `return_id` function takes in the question and answer sequence and makes sure that the maximum number of tokens is equal to the value of **length(i.e max_len=384 which I had fixed manually)** . The steps for fixing the number of tokens are as follows:

- If the input sentence has the number of tokens >384, the sentence is trimmed down to 384.

- To trim the number of tokens, 189 tokens from the beginning and 189 tokens from the end are kept and the remaining tokens are dropped.

- For example, suppose an answer has 700 tokens, to trim this down to 384, 192 tokens from the beginning are taken and 192 tokens from the end are taken and concatenated to make 384 tokens. The remaining [700-(384)) = 316] tokens that are in the middle of the answer are dropped.

- The logic makes sense because in a large text, the beginning part usually describes what the text is all about and the end part describes the conclusion of the text.

After performing the above tasks the return_id function returns [input_ids_q, input_masks_q, input_segments_q, input_ids_a, input_masks_a, input_segments_a], where

q stands for question and **a** stands for answer.

Implementation of BERT

The implementation of Bert for this task was similar to that for a normal text classification,but here instead of the three, inputs,attention mask and segment vectors,the BERT had to be given six inputs mainly two sets of (input, attention mask and segment) ,one for question and one for answer text.

BERT can handle a maximum sequence length of 512,but we only required a max len of 384 numbers of vector representation for a single input.

Now,in order to create model,a bert-base-uncased model was used for the tokenizer as well as for the the pre trained model,which was later fine tuned by changing the final layer to having 30 outputs with a softmax activation,apart from these Dropout of 20% and two more layers of GLobal average pooling 1 D were also used in the fine tuning of model.

The cross validation splits was of five folds but was trained on only three of them to save time.

The optimizer used was Adam optimizer with a learning rate of 2*10^(-5).The model was trained for two epochs,even then the training time was 1 hr,which was pretty big for a beginner like me.

Benchmark score Observed from the Leaderboard

There were 1571 participants in the competetion,and there scores ranging from as low as -0.00153,which were obviously flukes or manual answer entering attempts.

For a score between 0.2 to 0.3 ,your rank was about 1400 to 1300

For a score between 0.3 to 0.32 your rank was about 1299 to 1200

For a score between 0.32 to 0.34 your rank was about 1199 to 1000

For a score between 0.34 to 0.36 your rank was about 999 to 580

For a score between 0.36 to 0.37, your rank was about 579 to 198

For a score between 0.37 to 0.38, your rank was about 197 to 136

For a score between 0.38 to 0.39, your rank was about 135 to 99

For a score above 0.39 you lie in the top 100,which is a really impressive performance.The highest score was attained by well established data scientists or researchers who have many years of experience in this field,which was 0.489 on public leaderboard and 0.43 on private leaderboard.

All the above scores are for private Leaderboard which usually has lesser value than Public Leaderboard,in order to adjust these scores for Public leaderboard,you can add 0.05 to 0.07 to these range border scores

Explanation of Evaluation Metric

Submissions were evaluated on the mean column-wise Spearman's correlation coefficient. The Spearman's rank correlation is computed for each target column, and the mean of these values is calculated for the submission score.

Spearman's Correlation between two variables is equal to the Pearson correlation between the rank values of those two variables; while Pearson's correlation assesses linear relationships, Spearman's correlation assesses monotonic relationships (whether linear or not). If there are no repeated data values, a perfect Spearman correlation of +1 or −1 occurs when each of the variables is a perfect monotone function of the other.

Intuitively, the Spearman correlation between two variables will be high when observations have a similar (or identical for a correlation of 1) rank (i.e. relative position label of the observations within the variable: 1st, 2nd, 3rd, etc.) between the two variables, and low when observations have a dissimilar (or fully opposed for a correlation of −1) rank between the two variables.

The Spearman correlation coefficient is defined as the Pearson correlation coefficient between the rank variables.[3]

For a sample of size *n*, the *n* raw scores X_i, Y_i are converted to ranks $\mathrm{rg}_{X_i}, \mathrm{rg}_{Y_i}$, and r_s is computed as

$$r_s = \rho_{\mathrm{rg}_X, \mathrm{rg}_Y} = \frac{\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)}{\sigma_{\mathrm{rg}_X} \sigma_{\mathrm{rg}_Y}},$$

where

ρ denotes the usual Pearson correlation coefficient, but applied to the rank variables,

$\mathrm{cov}(\mathrm{rg}_X, \mathrm{rg}_Y)$ is the covariance of the rank variables,

σ_{rg_X} and σ_{rg_Y} are the standard deviations of the rank variables.

Only if all *n* ranks are *distinct integers*, it can be computed using the popular formula

$$r_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)},$$

where

$d_i = \mathrm{rg}(X_i) - \mathrm{rg}(Y_i)$ is the difference between the two ranks of each observation,

n is the number of observations.

Identical values are usually[4] each assigned fractional ranks equal to the average of their positions in the ascending order of the values, which is equivalent to averaging over all possible permutations.

Results and Conclusion

```
Epoch 1/3
811/811 [==============================] - 483s 595ms/step - loss: 0.4020
Epoch 2/3
811/811 [==============================] - 484s 596ms/step - loss: 0.3685
Epoch 3/3
811/811 [==============================] - 484s 596ms/step - loss: 0.3549
validation score =  0.3873238360765599
Epoch 1/3
811/811 [==============================] - 482s 595ms/step - loss: 0.3974
Epoch 2/3
811/811 [==============================] - 481s 594ms/step - loss: 0.3680
Epoch 3/3
811/811 [==============================] - 481s 593ms/step - loss: 0.3536
validation score =  0.3994787522858457
```

The picture you see above is the the validation score after running the code for two epochs and for three cross validation score each.In the output the score shown is 0.399 in the validation set,but while submission I guess the score might be between 0.39 to 0.40, I forgot to submit my predictions,because here in this competition you have to submit via the kernel itself rather than manual submission and I forgot to do that and closed my kernel,only to realise later the submission wasn't done,but no problem,as this competition was already over ,I guess there was no loss in my mistake.I also wanted to implement RoberTa and DistillBert,but due to college assignments and homeworks,I got time only for BERT Implementation.There was no point in training for more than 2 epochs since the bert already has such complex calculations,running for more than two epochs would lead to overfitting and that would lead to a low test score

Our model performs exceptionally well on the data,but the score which we see in the picture ,may not give exact same score when we submit the submission file.Moreover,the score in the picture is a Public leaderboard score which is not counted for competition medals.A rough estimate of our score is around 0.36 to 0.38 considering the trends of ups and downs seen on the leaderboard scores and roughly comparing them to my scores.I would have ranked in the top 500 people participating in this competition,which is really good considering my first attempt.A perfect estimate would be that i would lie in the top 25% percentile of the performers.

In Conclusion,I would like to say that the score can be increased a bit more by using a different model namely BART or AlBert,this was seen in the first position solution.I will try this in future.I would like to thank Udacity for giving me this opportunity to learn and implement a full fledged project on NLP on my own,which has increased my knowledge in this sector many folds.Thankyou Udacity and Amazon for this Nanodegree Program!!!