**Data Science Workshop: Kaggle - Google QUEST Q&A Labeling**

**Introduction:**

Computers are really good at answering questions with single, verifiable answers. However, humans still maintain the edge over computers when answering questions about opinions, recommendations, or personal experiences.

The difficulty to build better subjective question-answering algorithms is due to a lack of data and good enough predictive models.

Our project is taken from a [Kaggle challenge](https://www.kaggle.com/c/google-quest-challenge) which is to build predictive algorithms for different subjective aspects of question-answering and to improve the understanding of complex question-answer content.

**Dataset description:**

The data includes questions and answers from various StackExchange properties. Our task is to predict the target values of 30 labels for each question-answer pair.

The data contains 11 features and 30 target labels. We are using two data files:

* train.csv: the training data - 11 features and 30 target labels, 6079 entries
* test.csv: the test set - 11 features, 476 entries

Each entry contains a single question and a single answer to that question, along with additional features. The training data contains rows with some duplicated questions (but with different answers). The test data does not contain any duplicated questions.

There are 3 label types defined as:

1. “question” labels: target labels predict question property features in the data.
2. “answer” labels: target labels predict an answer property feature.
3. “both” labels: there are five labels that refer to the answer fit to the question.

**Dataset analysis summary - characteristics and the nature of the datasets:**

**Features**  
All features are string type (discluding *qa\_id*).

*Names*, *URL* and *pages* have a high diversity and are not relevant for problem-solving.

*Category* feature is composed of 5 categories: 'LIFE\_ARTS', 'CULTURE', 'SCIENCE', 'STACKOVERFLOW', 'TECHNOLOGY'.

*Category* feature has an obvious liaison with *host* feature.

The *question\_title* and *question\_body* could potentially overlap.

**Labels**

All target labels are aggregated from multiple raters and can have continuous values in the range [0,1].

Labels could have dependencies (e.g. *question\_type\_instructions* and *answer\_type\_instruction* are highly matched).

**Problem formulation**

Our goal is to improve automated understanding of complex question-answer content, whereas the predictions must be in the range [0,1].

**point of focus of our project**

Given the subject chosen, NLP requires a special treatment regarding the fact we are coping with non-numeric values. therefore, our two points of focus are:

1. Question classification: we focused on making a classification for the questions, based on papersshowing the benefit of classified questions in the NLP process.

This is considered as preprocessing, making the data as optimal as we can for the BERT model to process. We used a model called “Adam\_qas” (to be specified later).

1. Preparation of word embedding: in order to make the best analysis, words from the vocabulary are mapped to [vectors](https://en.wikipedia.org/wiki/Vector_(mathematics)) of [real numbers](https://en.wikipedia.org/wiki/Real_numbers), we had to prepare the data in such a way, that the word embedding model (“BERT”) will make the most of it.

**Description of the solution**

Baseline:

As a baseline, we took a first place Kaggle participants model, ( called BERT): BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create a model for a question answering problem, without substantial task-specific architecture modifications.

**inputs**: question\_title, question\_body, answer.

**Result:** reached a loss function of 0.3084.

Preprocessing:

1. Correlation calculation: In order to distinguish relevant features:
   1. labels correlation: we found multiple labels with high correlation:
      1. *question\_type\_compare*, *question\_type\_definition*: 1.0
      2. *question\_fact\_seeking*, *question\_has\_commonly\_accepted\_answer*: 1.0
      3. *question\_type\_procedure*, *answer\_type\_procedure*: 0.99
      4. more correlation detailed in the notebook
   2. feature-label correlation: we noticed that there are 5 relevant features for the prediction:
      1. *question\_title*
      2. *question\_body*
      3. *answer*
      4. *category*
      5. *host*

All other features have a low correlation to the labels, hence they can be neglected.

1. Distribution: *Category* is divided into five discrete classes, the baseline model did not use this fact, we assumed this could help improve the model.
2. *host* and *Category* features could be combined in order to find sub-category, thus adding a new feature.
3. Null handling: no zeroes, questions mark or Null values were given as data.

Step 1:

At first, we predicted a sole label, in order to learn the model and for a better understanding. After diagnosing which labels are both interesting and well distributed, We examined correlations (with the aim of finding a label that could potentially help predict other labels). From our research, the two most interesting labels were: ‘answer\_helpful’ and ‘answer\_relevance’. After analyzing correlation (0.37) and distribution we chose ‘answer\_helpful’ as the label to predict.

**inputs**: *question\_title*, *question\_body*, *answer*.

**result**: we got to the loss function of 0.207.

**note:** the baseline predicted all the labels, whereas we focused on a single one.

Step 2:

After succeeding in predicting one label with a (relatively) good function loss, we decided to move forward to predict all 30 labels instead of a single one.

Data engineering:

As indicated in “preprocessing Item 3”, *host* and *category* have a good potential to be merged, thus extraction of *host* and *category* was made to *sub\_category*.

**inputs**: *question\_title*, *question\_body*, *answer*, *sub\_category*.

**result:** loss function 0.3059.

Step 3:

While examining the data and understanding the nature of labels, we came to the conclusion that categorization of the questions could help us improve the model. hence looked for a compatible classification theoremand chose the most suitable model called “Adam\_Qas” that implements the theory. Based on the “Adam\_Qas” API, we designed an API to fit our needs (adam\_script.py).

Using the “Adam\_Qas” model, we classified questions into predefined categories. This classification would help us in our further processing (using the BERT model). the goal was to extract three new features from the data, thinking those could help to optimize BERT results, the three features were:

* 1. *q\_class*: a feature that defines the question type, divided into six classes.
  2. *q\_keywords*: a feature holding the essential keywords.
  3. *query*: a feature containing a query ready to be sent to Wikipedia. the query was supposed to be a tool for the comparison process of the Wikipedia answer and the given answer in order to define few labels indicating the answer quality (assuming the Wikipedia answer is the best answer, and using comparing sentence similarity methods, such as Word2Vec, Smooth Inverse Frequency, Cosine Similarity, etc.) the model is still in progressand thus we couldn’t manage to use this feature yet.

The adjustments and implementation of the “Adam\_Qas” model were done in “Google Colab” due to the long running time and GPU requirements.

BERT model uses a 512 char vector as input. We primarily chose a naive approach, where we divided the vector to two: the baseline(1st 256 Char), *q\_title, q\_body,* and *answer* and the second half for the “Adam\_Qas” output and *sub\_category* features.

**inputs**: *question\_title*, *question\_body*, *answer*, *sub\_category*, *q\_class*, *q\_keywords*, *q\_query*.

**result:** loss function 0.3124.

Breakpoint:

We realized that the 512 char vector is an input bottleneck. Therefore, we decided to try several divisions in order to maximize the number of relevant chars in the input vector.

This is why we calculated min, max and average length for each optional feature input in order to divide correctly.

Step 4:

Based on the BERT API, we designed an API to fit our needs (amen\_script.py).

Version 1: as said above, *query* given from “Adam\_Qas” is unnecessary and *question\_body* is sometimes redundant due to the *question\_title* and can be very long, the first input vector split that we tried was: all features except *query* and *question\_body*. excluding *answer\_body*, all the features had enough space in the vector. The remaining space was for (all or part of) *answer\_body*.

**inputs:** *question\_title*, *answer*, *sub\_category*, *q\_class*, *q\_keywords*.

**result:** loss function of 0.2981.

Version 2: Another version of this step was running the first version without *question\_title*, thinking the *q\_keywords* feature is sufficient.

**inputs**: *answer*, *sub\_category*, *q\_class*, *q\_keywords*.

**result:** loss function of 0.2986.

Step 5:

We decided to divide the process into three sub-processes, where each subprocess runs the BERT model on different input and predicts its own labels:

1. question relevance: the input contains the question features (such as *question\_body*) and predicts only the question labels (such as *question\_type\_instruction*)
2. answer relevance: the input contains answer relevant feature (such as *answer*) and predict only the answer labels (such as *answer\_well\_written*)
3. question and answer relevance: the input contains the features whose relevance is for both question and answer (that are predicted based on both question and answer) and predicts the matching labels (e.g. *answer\_relevance*).

In order to have a loss function reference for this updated model, we had to have a new baseline loss function for each sub-process.

**results**:

|  |  |  |  |
| --- | --- | --- | --- |
|  | question | answer | both |
| baseline | 0.3064 | 0.2716 | 0.2849 |
| Step 5 model | 0.3231 | 0.2681 | 0.2841 |

**Findings and statistical evaluation**

Baseline: 0.3084

First step: using only one label to predict: 0.207

Second step: using the “sub-category”: 0.3059

Third step: using the “sub-category” and “Adam\_Qas” model: 0.3124

Fourth step: version 1: 0.2981

version 2: 0.2986

Fifth step: table above

**Insights and applications**

We can gladly say we improved the first place solution at that time (Jan. 20).

Extracting *sub-category* instead of *host* and *category*, proved a better implementation.

Using “Adam\_Qas” naively didn’t improve the results at first, we assumed this was a result of an inefficient division of the input vector.

Dividing the vector according to statistical analysis gave a better result using “Adam\_Qas” as well as neglecting the *query* (“Adam\_Qas” feature).

In the fourth step, we assumed *q\_keywords* and *q\_classe* would compensate for the omission of *question\_title*, but found this assumption to be incorrect, as version 1 had a better result than version 2. hence concluded, *question\_title* has relevant information which “Adam\_Qas” cannot extract.

Surprisingly, we found in step 5 there was a deterioration for the question part. The significant difference between the baseline and our improvement was that we omitted the *answer* input, hence, we can conclude the *answer* holds an important role in predicting the question labels.

**Related work**

We joined the Kaggle competition but decided to use an external source and modify it for our purpose. the challenge rules don’t allow it, hence we couldn’t submit our notebook as a legitim one for the competition. We intend to publish our work as a contribution to “Adam\_Qas” and BERT repositories, after approval.

**Citations:**

1) Medlock, B.W., 2008. *Investigating classification for natural language processing tasks* (No. UCAM-CL-TR-721). University of Cambridge, Computer Laboratory.

2) [NLP: Question Classification using Support Vector Machines [spacy][scikit-learn][pandas]](https://shirishkadam.com/2017/07/03/nlp-question-classification-using-support-vector-machines-spacyscikit-learnpandas/)

3) Huang, Z., Thint, M., & Qin, Z. (2008, October). Question classification using head words and their hypernyms. In *Proceedings of the 2008 Conference on empirical methods in natural language processing* (pp. 927-936).

Kaggle notebooks:

our baseline: [Bert-base TF2.0 (minimalistic) III](https://www.kaggle.com/khoongweihao/bert-base-tf2-0-minimalistic-iii)

our supplement kaggle [notebooks](https://xaxok44829.wixsite.com/kagglenotebooks)

useful articles:

[How Well Sentence Embeddings Capture Meaning](https://white.ucc.asn.au/publications/White2015SentVecMeaning.pdf)

[Comparing Sentence Similarity Methods](https://nlp.town/blog/sentence-similarity/)

<https://github.com/seatgeek/fuzzywuzzy#usage>

[Text Similarities : Estimate the degree of similarity between two texts](https://medium.com/@adriensieg/text-similarities-da019229c894)

code repository:

Adam\_Qas: [ADAM - A Question Answering System](https://github.com/5hirish/adam_qas)

BERT: [bert\_tokenization](https://www.kaggle.com/akensert/bert-tokenization)