Kaggle Competition: Quora Question Pairs

(https://www.kaggle.com/c/quora-question-pairs/overview)

Abstract

In the *Quora Question Pairs* competition, we were challenged to tackle the natural language processing (NLP) problem, given the question pairs, classify whether question pairs are duplicates or not. This report describes our team's solution, which acheives top 10% (319/3394) in this competition.

Team Membors

- 羅右鈞 105062509
  - Text preprocessing: Automatic spelling error correction
  - Feature engineering: Latent semantic embeddings, word embeddings, similarity distances of various embeddings, syntatical features, etc.
  - Modeling: CNN, LSTM, gradient boosting machine.
- 林杰 102000039
  - Text preprocessing: Name entity replacement.
  - Data preprocessing: data augmentation.
  - Modeling: LSTM, gradient boosting machine, ensembling.

Dataset and Evaluation Metric

The dataset is released by *Quora*, which is a well-known platform to gain and share knowledge about anything. Let's have a glance on the dataset:

- Training data: 255027 non-duplicates and 149263 duplicates.

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24}[/math] is divided by 24,23?	0
4	4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0
5	5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?	1

- Testing data: 2345796 question pairs, no ground truth, need to be evaluated on the *Kaggle* platform.

	test_id	question1	question2
0	0	How does the Surface Pro himself 4 compare with iPad Pro?	Why did Microsoft choose core m3 and not core i3 home Surface Pro 4?
1	1	Should I have a hair transplant at age 24? How much would it cost?	How much cost does hair transplant require?
2	2	What but is the best way to send money from China to the US?	What you send money to China?
3	3	Which food not emulsifiers?	What foods fibre?
4	4	How "aberystwyth" start reading?	How their can I start reading?
5	5	How are the two wheeler insurance from Bharti Axa insurance?	I admire I am considering of buying insurance from them

The evaluation metric in this competition is log loss (or cross-entropy loss in binary classfication):

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

Main Tools

We use several popular NLP tools to help us acheive several necessary NLP tasks:

- SpaCy: An industrial-strength python based NLP tool, mainy provide some useful functionalities, such as tokenizing, part-of-speech tagging, dependecy parsing, entity recognition, etc.
- Textacy: Provide High-level APIs of *Spacy*, mainly for text normalization.
- Scikit-learn: Python-based machine learning library. We used it for extracting latent semantic features and some of its simple models in blending.

- Word2Vec, GloVe, FastText: Those are neural-based word embeddings, which represent a word as a real-valued dense vector. It captures nice syntatic and semantic properties in vector space. We used them for feature engineering and modeling.
- Keras: High-level deep learning library, which made us easy to implement complex deep learning models.
- XGBoost: An optimized distributed gradient boosting library. This offers another kind of non-deep-learning models for us to perform ensembling.

Method

Text Preprocessing

Here are some NLP tasks we've done in detail, we thought that some appropriate processings might help getting better performance, but we also found that "over-cleaning" data might lead to information loss:

- Text cleaning: Tokenizatiom, convert all tokens to lower case, remove punctuations, special tokens, etc.
- Text normalization: Restore abbreviations (e.g. "What's" to "What is", "We're" to "We are", etc.) using regular expression, replace number-like string with "number" and replace currency symbols with "USD" abbreviation.
- Name entity recognition (NER):
    - Perform name entity recognition with *SpaCy*, then replace recognized entities with predefined global name entity lookup table to reduce entity types. For example, "I am a Taiwanese living in Taiwan." to "I am a NATIONALITY living in COUNTRY."
    - Pros:
        - Make training more stable.
        - Data may have better generalizability. This may also help our models become better to capture sentence meaning.
    - Cons:
        - Risky in destructing data (imformation loss).
        - Since training data contains mislabeled examples, so NER can't really provide a better score.
    - Since there's no direct evidence to whether this method is applicable, so we decided not to take risks. But clearly we should include this method into

ensembling.

- Automatic spelling error correction:
    - First, check if a word is in the *Glove* (described below) vocabulary (~100M), if not, the word is considered to be mispelled. Second, find a list of good replacements for the misspelled word. Finally, choose the best replacement based on *SpaCy*'s smoothed log probability of the words.
    - Some special cases are avoided to be corrected in order not to destruct data too much, such as the word is a special term recognized by applying NER.

Feature Engineering

- Basic text features:
    - Length information: Character and word length.
    - Length difference information: Character and word level length absolute differences, log length absolute differences, length ratio, log length ratio.
    - Common word intersection count
    - Various string fuzzy matching scores
    - Non-ascii word count
    - Punctuation symbol count
    - Number in question count
    - Out of vocabulary word wount
- Syntatic features: We convert a sentence to POS tag sequence and dependency tag sequence. For example, “Can I recover my email if i forgot the password” to POS tag sequence “VERB PRON VERB ADJ NOUN ADP PRON VERB DET NOUN”, and to dependency tag sequence “aux nsubj ROOT poss dobj mark nsubj advcl det dobj”. We then compute basic features based on those sequences to generate a set of new features. Also, “ROOT” postion is also a new feature.
- Latent semantic embedding features: Character and word N-gram (N=1~3) count matrices, TF-IDF weight matrices with L1, L2-norm. We also applied SVD on TF-IDF weight matrices to perform dimension reduction.
- Neural embedding features: *Word2Vec* pretrained on *Google News*, *GloVe* and *FastText* pretrained on *Wiki*. We basically sum all of word embeddings and normalize it to unit vector to represent a sentence (so-called “sentence embedding”). In our

case, DL models perform better with embeddings pretrained on *Wiki* than *Google News*. We finally choose *FastText* embeddings as input features for our DL models, since *FastText* trains on larger *Wiki* corpus than *GloVe*.

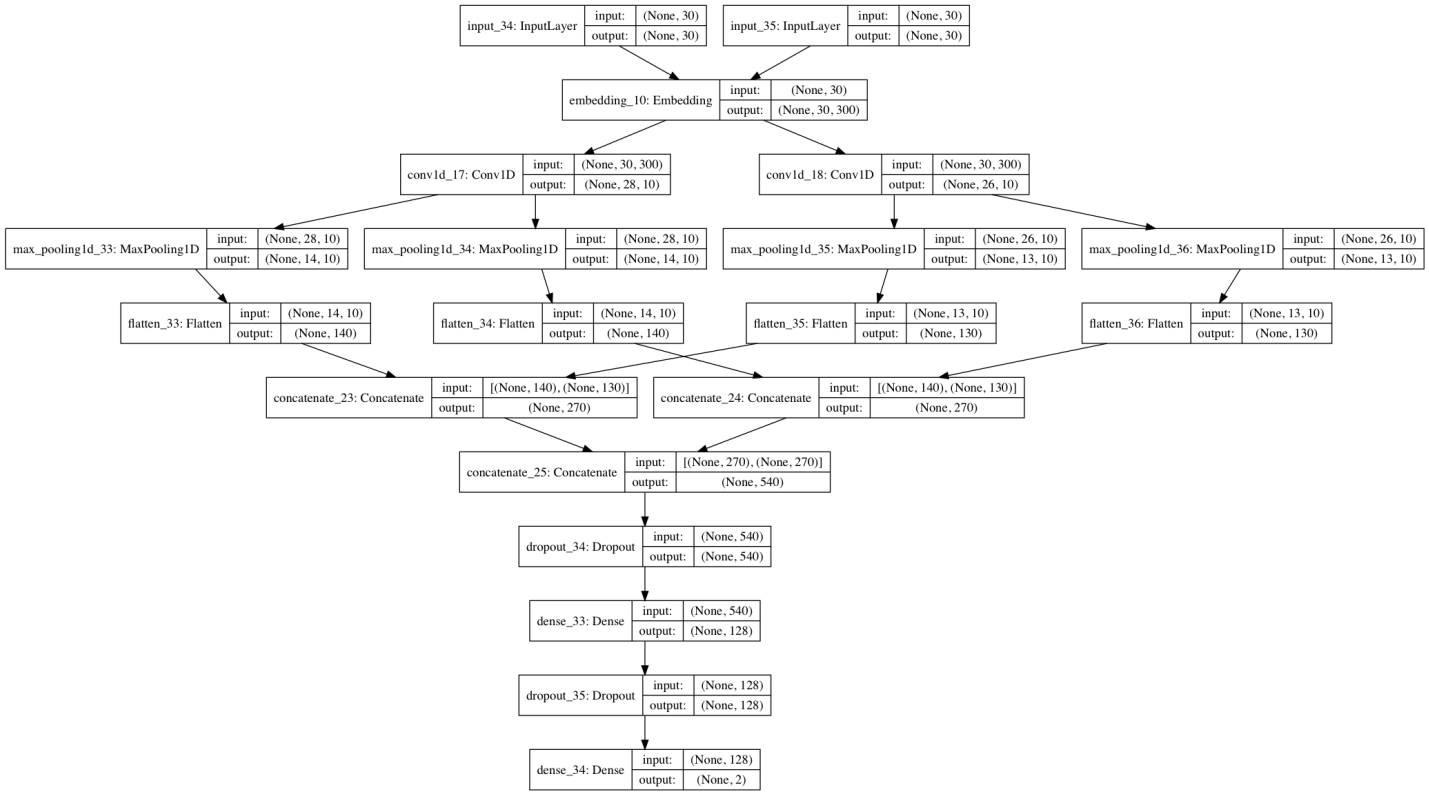
- Embedding distance features: We also compute various distance metrics, such as cosine, cityblock, canberra, euclidean, minkowski, braycurtis distance for both *Word2Vec* and *FastText* sentence embeddings, and also word mover distance only for *Word2Vec*.
- Embedding skewness and kurtosis: Skewness is a measure of lacking of symmetry. Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution.
- LSTM representation: Deep learning is well-known with its ability of representation learning, so we extracted the LSTM layer output of model as a new feature. But it gives very little improvement and significantly increases risk in over-fitting problem.
- Other “magic” features (or “competition-oriented hacking features”):
    - Word correction count: Should be a strong feature, since some of typos of data are machine-generated and have some pattern amoung them. But it took way too much time to compute, so we only tried on training data but end up giving up due to the requirement of expensive computation on testing data.
    - Question ID (qid): Originally, qid is only an identifier for specific question. But one of the participants found that the qid is actually a time dependent serial number (by searching on Quora directly). Furthermore, the duplicate label ratio becomes lower and lower as the time pass. The hypothesis is that Quora itself also tried to reduce duplicated question in this period of time. So, the qid becomes a strong feature.
    - Numbers of question occurrence: One suggested that the duplicated questions tend to be the same group of questions. In other words, the more time a question appears in the training set, the more probable such question pair is duplicated regardless of what question is paired with it.

  It is worth mention that those magic features end up helping a lot. Though we call those features “magic”, in fact, we think those magic features basically capture some hidden phenomenon behind the data, which make them become strong features in this task.

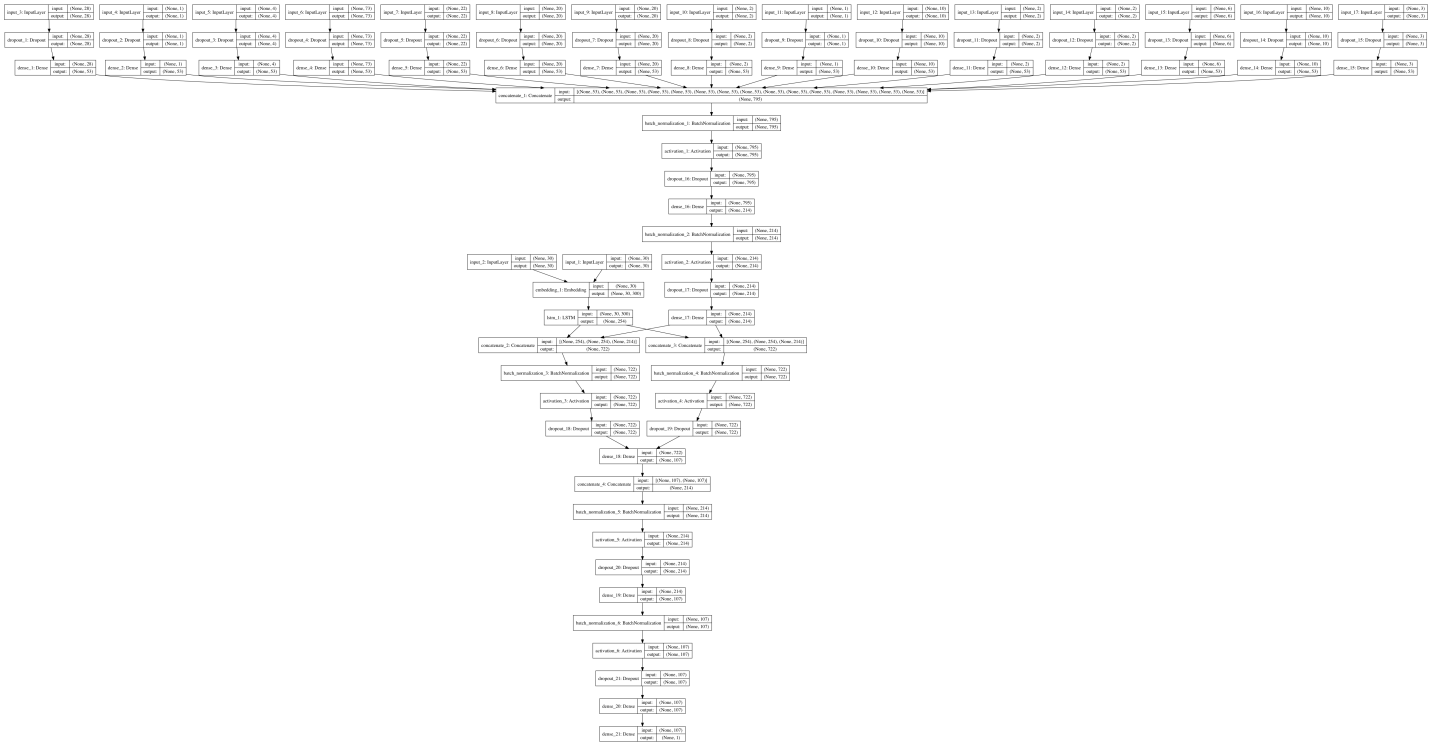
Modeling

We basically tried dozens of model architectures and parameter tunings. Here we only report the models that works for us.

- Glove embeddings with 1D-CNN. Our CNN contains 2 different size of filters (3, 5). Filter number is 10. Basically, we've also tested larger number of filter and larger size of filter, but the performance did not improve much. This model was one of our first early trials, but we found that LSTM was much better in encoding questions, so we sticked to LSTM until the end of the competition. It is worth mentioned that after the competition, we've found one team used 1D-CNN successfully with some interesting designs, such as using global average-pooling layer instead of local max-pooling layer after the convolutional layers, and computes absolute distance and element-wise multiplication after the pooling layers. They also fed externel handcrafted features like we did in our LSTM (described below), which mainly improves model performance a lot.



- FastText embedding with single layer LSTM. On the other hand, we have features described in *Feature Engineering* with fully-connected (FC) layers individually. Finally, we merge those 2 models' outputs and feed it to the final deep FC layers.



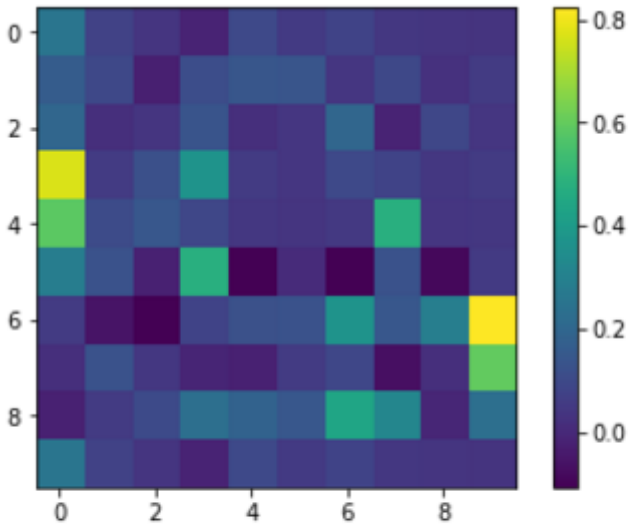
- *XGBoost* with all features described in *Feature Engineering*. We exclude sentence embeddings for training simply due to the expensive computation caused by the large dimensionality. Moreover, *XGBoost* performs well enough only with those handcrafted features.

- Other models we've tried:

    - Bidirectional LSTM: People usually replace LSTM with Bi-LSTM since its strong performance. In our case, it doubles the training time but with a very slight improvement. (Although an improvement is still precious for data scientists.)

    - Treating question pairs as an image: Compute word embeddings similarity distances between each of the words of one question and each of the words from the other sentence. Since some duplicate questions seems to have positional locality (i.e. "same short phrases", "similar descriptions with some words being replaced"), so we can treat it as an image. This method has approximately the same performance compared to either LSTM or CNN approaches and has

advantage in visualizing, but the downside is the training loss is very unstable during training, which made us finally decide to give it up.




```
Q1: How do you get your best friend to confide in you?
Q2: How do I get a best friend?
Duplicate: 0
```

- All of the models mentioned above resulting with log-loss about 0.30~0.32 (without handcrafed features but only neural word embeddings) and accuracy about 85%, which makes us think again if the model has already sufficiently used all the information we provided and starts to become overfitting. So, we starts to focus on feature engineering instead of trying more fancy deep learning models.

Data Augmentation

A simple and helpful approach but risky in some cases, for example, in this competition, the porpotion of the positive example is 37% in our training data. But if we upsample our positive examples to 50%, it might be problematic in terms of mismatching the actual class label distribution in testing data (or similarly, in the real world), which may introduce a positive bias to our model, especially the tree-based model like *XGBoost* we used in this competition. So finally, we mainly use class label reweighting technique (described below) to address imbalanced data problem.

- Can we simply upsample by forming two identical questions together as a new training example?

  NOT EXACTLY. There doesn't exist any of such kind of examples in the dataset. It may improve generalizability for "human", but not for this dataset, hence for this competiton.

- Does two duplicate pairs (A,B) and (B,C) imply that (A,C) pair is also duplicate?

NO. There exists 1% mislabeled examples in the dataset. Both methods destruct the original data distribution. There exists some (A,C) pairs that are labeled as non-duplicate pair.

- Can we change the order when concatenating question pair embeddings horizontally as a data augmentation technique (e.g. `x1 = concate(q1, q2, axis=1)` to `x2 = concate(q2, q1, axis=1)`) when using neural networks?

  YES. This works. Since `x1` and `x2` might yeild different results when feeding into the FC layer, we used such technique to balance our network training. Unfortunately, during the competition, we forgot to switch the externel handcrafted features' order of the question pair (described in *Feature Engineering*) when feeding into the FC layer along with question pair embeddings, which might effect badly to our model's performance.

Class Label Reweighting

The participants of this competition had noticed that there is a different porpotion of positive examples between the training data (~37%) and the testing data (~17%). That is, we can probably get different evaluation score on our validation set and testing data. In order to address this problem, we assigned different class weights when training our models.

Note that we could not know the porpotion of the positive examples in the testing data in advance, but we can simply estimate it by making a constant prediction at the training data with the mean of `0.369` . After submitting to *Kaggle*, we got a score of `0.554` . Using the log-loss formula, we can derive the porpotion of the positive example in the testing data is `0.174` . Finally, we can then reweight positive class label to `0.174/0.369` and reweight negative class label to `(1-0.174/(1-0.369)` .

Training and Ensembling

We split ~400,000 training data into ~360,000 examples as training set and ~40,000 examples as validation set.

We further split the original ~40,000 validation set into ~28,000 training set and ~12,000 validation set for training our ensembling models.

But if we want to do ensembling, we shouldn't train each ensemble model on the same 360,000 training set, which makes the ensembling models overfit to the 360,000 training set very rapidly and losses generalizability.

Pros:

- Overcomes the problem of all ensembling models works too well on original training set since we don't use any of the original training set.
- Uses out-of-bound samples to do training, which has better generalizability.

Cons:

- The training and validation set in blending are too small, implying that it is risky in generalizability and tends to overfit on small validation set. But eventually it works.

We also did traditional stacking with LSTM directly, but we found that this overfits extremely and has no generalizability, since the validation loss was about only ~0.11, but ~0.16 on the *Kaggle* private log loss.

Submission Score

We finally achieves top 10% (319/3394) in this competition.

	Private log loss	Public log loss
LSTM	0.15625	0.15380
Stacking with XGBoost	0.17086	0.167081
Blending with XGBoost	0.15489	0.15172

(Note: The public log loss is calculated with approximately 35% of the testing data. The private log loss is based on the other 65%. The *Kaggle* competition's ranking score is based on the private log loss.)