CS3244 HW3 Set Question Answering

Zhu Liang (paradite)

November 1, 2016

Abstract

For this project, we have developed a classification model based on word2vec, to tackle the set question answering problem(Weston et al., 2015). The implementation use scikit-learn[1] and word2vec[2] libraries in Python.

1 Analysis of Data

From the training data, we know that the input data is sentences of varying length. This means we cannot directly feed the input into the SVM model without transforming it into a fixed dimension. Either padding or word embedding techniques is required.

Also, there can be multiple answers for one input question. This means we have two options for the model output format:

1. We can either treat the multiple answers as a single label and perform a multi-class single-label classification, i.e. only 1 output. Tthe single output then maps to 0, 1, or multiple answers. For example, the output can be "nothing", "milk" or "milk,football".

2. Alternatively, we can use a multi-class multi-label classification where our model outputs varying length of output, with each output corresponding to only 1 answer. For example, the output can be [], ["milk"] or ["milk", "football"].

2 Validation Setup

Since we can only submit a limited number of test results to the Kaggle platform everyday, it is necessary to develop a validation mechanism to test our model. For our project, we use the cross validation package from scikit-learn to perform 5-fold validation on training data as an estimate of the our-of-sample accuracy.

However, empirical results suggest that this was not a good estimate. Irregardless of the model that we use, there is a gap of 0.1 to 0.17 between the validation result and the actual test result reported from Kaggle. In some cases, while our model reports 0.94 accuracy from the validation result, the actual test

[1] `http://scikit-learn.org/`

[2] `https://github.com/danielfrg/word2vec`

accuracy is only 0.77. We suspect that the training data and the test data have been generated using different techniques to cause such a large discrepancy.

Nevertheless, there is a strong correlation between the validation result and the test result, improvement to validation results usually leads to improvement to test result (of a smaller scale). Hence, this setup is sufficient to gauge how well our model perform.

3 Pre-processing

3.1 word2vec

Word2vec model is a word embedding tool that allows us to transform a word into a vector with dimension d, where d is set during training of the model. We use word2vec model to transform our input sentences into a vector of fixed dimension. After we have trained the word2vec model, we use it in the following steps:

1. Retrieve each word's original vector, v of dimension d from the word2vec model
2. Apply weight to v to obtain $v_{weighted} = v * w_w * w_s$, where w_w is the weight of the word in the sentence, and w_s the weight of the sentence in the story. In both cases, the weight is simply the index of the word in the sentence or index of the sentence in the story, starting from 1.
3. Sum the vectors of all words in all sentences linearly to obtain $v_{story of dimension d}$

Using this approach, each story is transformed into a vector of dimension d (set to 100 in this project) ready to be taken as the input of the classification model.

We have tried to concatenate the vector of each sentence together to obtain a vector of dimension $d*l$ where l is the number of sentences in a story, however, we need to do restrict the maximum number of sentences l_{max} as well as add padding for the cases where there are few than l sentences. Empirical results showed that this perform worst than simply summing up the vectors.

One possible reason is that summing up the vectors preserves the semantic meaning of the sentences as the words of opposite meanings would cancel out each other in the vector form. This is especially helpful for this particular task as verbs of opposite meanings such as "pick up" and "drop" would have their vector representations canceling each other in the final input, reducing the noise and capturing the semantic information accurately.

We have also experimented with various weighting schemes such as power and log of the index of the word in a sentence, or index of the sentence in a story, besides assigning weight linearly according to the index. However, empirical result shows that linear weights yields better results.

3.2 Word Index Vector

In addition to the vector $v_{weighted}$ obtained from word2vec model, we also use another vector v_{index} to represent the story based on the ID of each word.

First we assign an index to each word in our vocabulary for training data. Here we make use of the assumption that same vocabulary is used in test data given by the teaching assistant in the course forum[3].

Then for each word, we obtain a vector containing 1 in the index of the word and 0 elsewhere. This is similar to the idea of a sparse matrix. Using this approach, we get separate vector representation of each word of dimension k, where k is the size of our entire vocabulary. Lastly we sum up the vector of each individual word to obtain a vector of dimension k to obtain v_{index}.

With both $v_{weighted}$ from word2vec and v_{index} from word index, we concatenate them together to obtain a vector of dimension $d + k$ as our final input vector. Results shows that compared to using pure word2vec vector, addition of the word index vector improved the test accuracy by 0.002. A more comprehensive experimentation result of improvements using each technique can be found in section 4.3.

3.3 Other techniques

We followed Weston et al. (2015)'s suggestion to filter irrelevant sentences with set of words that do not intersect the set of words in the question. This reduces the amount of noise in the input data, hence improving the test accuracy by 0.103.

We realized that the question sentence and the supporting fact sentences are of very different natures. And since we always have exactly one question and one set of supporting fact sentences, we can break down each story into 2 parts containing the supporting facts and the question sentence respectively. Then we can apply the transformation described above to each of the part and concatenate the result to obtain a vector of dimension $2*(d+k)$, which preserves more information about the story. Results showed that this improved the test accuracy by 0.0012.

We tried to use Principal component analysis (PCA) to reduce the input dimension. However, apart from speeding up the training process, it does not improve the accuracy or generality of the model. When number of components is set to a low number with sum of explained variance from all components less than 1, we noticed that both validation result and test result dropped. This suggests that PCA is not useful in reducing noise in this task.

4 Models and Experimental Results

4.1 Support Vector Machine (SVM)

One model that we used from the beginning is SVM. We have done extensive experiments with SVM for parameter selection.

First we tried out different kernels. After experimenting with linear, rbf and polynomial kernels, we concluded that rbf kernel is most effective and time-efficient for this task due the high dimension of the input data. Despite polynomial kernel being favored for natural language processing (NLP) tasks in

[3] https://ivle.nus.edu.sg/v1/forum/board_read.aspx?forumid=7df82300-8101-475e-8b97-c7b6af32d59e&headingid=00000000-0000-0000-0000-000000000000&postid=00728933-b72e-4fa8-8875-d6f2cf65e62f&lpreferrer=&lastpost=1&currpage=last

general, it fails to terminate within reasonable amount of time even after performing PCA to reduce $2 * (d + k) = 164$ dimensions into 20 dimensions.

We also performed standard grid search for parameter γ and C in our SVM model with rbf kernel. Results showed that the validation result is highest when $\gamma = 0.01$ and $C = 100$.

4.2 Meta Models

The SVM implementation in scikit-learn allows multi-class classification but not multi-label classification by default. We use OneVsRestClassifier to wrap our base model in order to output multiple labels for one input. This yields an increase of 0.0016 improvement in the test accuracy.

We also tried Ensemble methods such as bagging and Ada Boost. The scikit-learn documentation recommends bagging for complex models whereas Ada Boost for weak models [4].

From our experiment, bagging method did not improve the validation and test performance. When the percentage of subset is reduced for sampling, both validation and test accuracy dropped. Hence, there is no incentive to use bagging method for our model.

As for Ada Boost, it failed to terminate within reasonable amount of time, presumably due to the complexity of our model.

Both bagging and Ada Boost were based on the base SVM model. We also tried the implementation of random forest provided scikit-learn completely independent of the SVM model. From our result, it yields almost identical result as our SVM model in significantly less time. Hence, it is a good alternative base model for this task.

4.3 Table of Results

Below is the summary of the result. Validation Accuracy is reported by 5-fold validation using training data. Test Accuracy is reported by Kaggle.

[4] `http://scikit-learn.org/stable/modules/ensemble.html`

Base Model	Input	Meta Model	Validation Accuracy	Test Accuracy
SVM polynomial	word2vec, no weight, no noise filter	-	(Did not terminate)	(Did not terminate)
SVM rbf, C = 1	word2vec, no weight, no noise filter	-	0.640	0.607
SVM rbf, C = 1	word2vec, no weight, with noise filter	-	0.820	0.710
SVM rbf, C = 1	word2vec, power weight, with noise filter	-	0.830	0.731
SVM rbf, C = 1	word2vec, log weight, with noise filter	-	0.820	-
SVM rbf, C = 1	word2vec, linear weight, with noise filter	-	0.900	0.751
SVM rbf, C = 1	word2vec, linear weight, with noise filter, separate qn/ans	-	0.910	0.753
SVM rbf, C = 100	word2vec, linear weight, with noise filter, separate qn/ans	-	0.930	0.767
SVM rbf, C = 100	word2vec+word index, linear weight, with noise filter, separate qn/ans	-	0.935	0.769
SVM rbf, C = 100	word2vec+word index, linear weight, with noise filter, separate qn/ans	Multi-label	0.940	0.770
SVM rbf, C = 100	word2vec+word index, linear weight, with noise filter, separate qn/ans	Multi-label, bagging	0.935	0.769
SVM rbf, C = 100	word2vec+word index, linear weight, with noise filter, separate qn/ans	Multi-label, Ada Boost	(Did not terminate)	(Did not terminate)
Decision Tree	word2vec+word index, linear weight, with noise filter, separate qn/ans	Random Forest with 100 trees	0.948	- *
Decision Tree	word2vec+word index, linear weight, with noise filter, separate qn/ans	Random Forest with 50 trees	0.941	0.771

"-" indicates no data available for that combination.
"*" indicates potential over-fitting.

5 Files Submitted

- train-word2vec.py - Script to train word2vec model, requires training file text8[5]

[5]http://mattmahoney.net/dc/text8.zip

- hw3-qasets.py - Main script, including training, testing, parameter selection and validation of model
- README.pdf - This file, report on the project

6 Statement of independent work

[Zhu] I, A0093910H, certify that I have followed the CS 3244 Machine Learning class guidelines for homework assignments. In particular, I expressly vow that I have followed the Facebook rule in discussing with others in doing the assignment and did not take notes (digital or printed) from the discussions.

References

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., & Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.