Predicting Answer Quality on Community Q&A Websites

Tin-Yun Ho, Ye Xu

December, 2015

1 Introduction

More than ever before, internet users obtain needed information from community-based Q&A websites such as quora, stack exchange, and other forums. A critical component of making these sites work is ensuring that the most high quality answers are pushed to information-seekers first, and also ensuring that writers are encouraged/informed on how to provide high quality information. In the longer-run, identifying what makes a good answer can also eventually be used to help create effective AI-based Q&A systems by allowing them to better rank and weight potential answer choices.

In light of this, we decided to create a system that can take any given question post's answer raw text and metadata and push the highest quality answers to the top of the page in ranking. We formulated the problem in two ways, as a pointwise and a pairwise ranking problem using binary response variables and evaluated using mean average precision (MAP). For our classification algorithms we used regularized logistic regression and regularized linear/gaussian kernel support vector machine classification. For feature selection we used regularization parameters as well as forward search.

2 Related Work

The most relevant work we found came from the Semeval Competition of 2015 task 3 [Nak+15]. Their challenge was to effectively classify posts in community based Q&A sites (this time a Qatar lifestyle discussion forum) into whether they answer the question, potentially provide useful information, or are flat out unhelpful. Each competitor posted their papers online, and we reviewed the winning entries as well as some followup papers posted after the competition. A strength that the top papers shared [Tra+15; Hou+15; Nic+15] was that they went far beyond just using relevant word counts/inclusion as features, including significantly more metadata such as word length/number, sentence length/number, xml/content tags, etc. They also incorporated a variety of different measures of textual similarity between the question and answer such as tfidf cosine similarity, word2vec cosine similarity, measures of whether the question and answer come from the same topic distribution, and more, all strengths which we adapted to our problem.

We also found several weaknesses with the winning papers. The first and most obvious one was that they by and large treated the text itself as a bag of words (and at best a bag of phrases), as opposed to actually teasing out semantic meaning from statements and determining whether the meaning of those statements actually truly answered the posed questions from a semantic perspective. The second was that there was no mention of the use of interaction terms, even though one would expect that the appearance of certain words would only be helpful when answering certain topics or types of questions. The latter problem we solved through the addition of interaction terms, but the former problem we couldn't come up with a good way to solve. Finally, one major difference between our project and the winning papers was that they were provided with authoritatively labeled data, while for us, score on StackExchange is at best a noisy indicator of quality, hence motivating us to create our own response variable (explained below).

Later papers we reviewed published after the competition branched out in a variety of ways. One was with the usage of different types of neural networks [Zho+15]. Another was to try identifying patterns of dialogues between answers in response to a given question [Bar+15]. Both approaches improved performance, but as the former was not a major focus of CS 229 and the latter was not as applicable to StackExchange's data (since most questions only had 1-2 answers and little dialogue), we did not apply these approaches.

3 Experimental setup

3.1 Collecting Data

For data, StackExchange periodically provides anonymized dumps of all user-contributed content on their entire network. This includes, for each of their domains, every single question and answer posted as well as their major attributes such as Id, ParentId (to identify the parent question of an answer post), creation date, score (upvotes minus downvotes), view count (only for questions), title and body full text, tags, favorite count, owner's user id, tags, answer count (for questions), accepted

answer id, comment count, and more. We used the data dump released on August 15, 2015. We are currently starting just with the "Ask Different" dump of posts related to apple products which has 90,000 answer posts. [Inc15]

3.2 Creating an Appropriate Response Variable

For the response variable, we sought an objective metric of answer quality. We initially tried using pure score as our metric, but found that it was highly correlated to a variety of factors irrelevant to quality. For example, the natural log of score is correlated to age of the post (r = 0.31), a natural log estimate of the number of views of the answer post ln(parentViews * postAge/parentAge) (r = 0.24), age of the parent question post (r = 0.22), natural log of the ratio between post age and parent age ln(postAge/parentAge) (r = 0.21), etc. This makes intuitive sense–an answer post which has existed for a longer time and has had more people view its original question post is likely to have a higher score regardless of quality.

Thus, we used ordinary least squares (OLS) to create a prediction of the response variable ($\hat{y}$) based on these non-quality-related features for each answer post. Then by subtracting the predicted response variable from the real response variable ($y_{new} = y - \hat{y}$), we get a measure of the part of the response variable that is currently unexplained by the above non-quality-related features and thus more likely to purely measure quality.

Here, we use the unregularized version of OLS implemented in sklearn, which seeks to solve the optimization problem $min_w||Xw - y||_2^2$, where X is an $m \times n$ matrix of real numbers, with each row representing a data point and each column representing the features (that we wish to control for), w is a $n \times 1$ matrix of real numbers with each row representing the weight for its respective feature (to be estimated in the optimization problem), and y is a $m \times 1$ matrix with each row entry equivalent to the raw score for that respective data point. Finally, the norm here used is l2. Combined with the squared term, it equates to $Z^T Z$ where Z is the term inside $||\cdot||$. [Bui+13]

For the original y variable, we set $y = \log(score + 1)$ if score is zero or positive, and $y = \alpha score$ if score is negative, where α is a positive constant that varies depending on the range of scores for a given domain. The natural log smoothing helps us deal with the long tail of positive scores and also enables our aforementioned features to have much more explanatory power/correlation to the dependent variable.

For specific features, we chose post age, parent age, parent view counts, order of answer post, as well as each of their respective natural log transformations. Next, by running the ordinary least squares algorithm, we obtained adjusted scores ($y_{new} = y - \hat{y}$) that were almost completely uncorrelated with these non-quality-related features (r = 0.00 for each), normally distributed around a mean of 0, and still leave a lot to be explained ($R^2 = 0.15$ between y and $\hat{y}$). Finally, we translate this value into a very simple and evenly split classification for each sample where helpful posts with score greater than mean are class 1, and unhelpful posts with score less than mean are class 0.

3.3 Pre-processing and Feature Selection

We began by extracting all the question answer posts and metadata from the data dump, converting from xml to python dictionary format. Next, we created the following metadata and text-based feature matrices:

- **Topic tags:** log total and binary existence by type for parents (questions) of each answer post
- **XML tags:** log total, raw count and log count by type, tfidf score, binary existence by type for each answer body text
- **Question metadata:** log # of characters, words, sentences, sum of question words, binary existence of question words
- **Answer metadata:** answer log number of characters, words, sentences
- **Word2vec representations:** Word2vec representation of each question and answer post's body text
- **Raw text data:** lower case unigram and bigram binary existence, raw and logged counts, tfidf scores for answer body
- **Q&A similarity:** Word2vec, Tfidf cosine similarity between question and answer body text vectors
- **Interaction terms:** 2nd degree interaction terms between a dimension-reduced matrix of word Tfidf frequency features and a dimension-reduced matrix of topic metadata features

For word2vec representations, we used the pre-trained implementation provided through the Spacy toolkit [Co15], which is a dependency-based implementation of word2vec based on [LG14]. Each word in the vocabulary $w \in W$ is associated with a vector $v_w \in R^d$ where d is the embedding dimensionality. The entries in the vectors are latent, and treated as parameters to be learned. The probability of a given pair of words existing in the data is measured as $P(D = 1|w_i, w_j) = \frac{1}{1+\exp(-v_{w_i} \cdot v_{w_j})}$, and the probability they don't exist in the data is $P(D = 0|w_i, w_j) = 1 - P(D = 1|w_i, w_j)$. Given a large corpus of pairs of words observed in the data $(w_i, w_j) \in D$, as well as a list of randomly generated pairs of words (assumed not in the data) $(w_i, w_j) \in D'$, we then maximize log likelihood (using stochastic gradient descent):

$$\arg\max_{v_w} \left(\sum_{(w_i,w_j) \in D} \log P(D = 1|w_i, w_j) + \sum_{(w_i,w_j) \in D'} \log P(D = 0|w_i, w_j) \right)$$

Implementations differ based on how they decide whether a given pair of words counts as a positive example. Traditional implementations use a context window of k words: if two words appear within k words of each other in the raw text, then the pair is a positive example. The dependency implementation considers a pair of words a positive example if one word is syntactically either the immediate head of or modifier of the other word (appearing either immediately above or below each other in the grammatical parse tree derived from the raw text). The advantage of this is that it goes beyond just capturing topical similarities between words to also removing spurious correlations between words that just happen to be close to each other in the text, as well as capturing functional similarities. Finally, to represent a given question/answer post, we take the average of the word2vec vectors of its individual words, effectively giving us a fixed, compact (300 variable) numerical representation for every post in our corpus.

For tfidf score, we used the sklearn implementation [Bui+13]. Tfidf stands for term-frequency times inverse document-frequency, where term-frequency is one plus the log of the natural count of occurrences of a word in any given post and inverse document frequency stands for 1 over the number of answer posts a word has appeared in within the entire corpus plus 1 to normalize/prevent zero division. The advantage over raw counts is that this formulation discounts words that appear frequently in all posts (and thus likely carry less information, such as 'the' or 'a'). Finally, each of these scores is normalized by dividing by the l2 norm calculated over all the values for that feature.

For both word2vec and tfidf-based similarity, since for both models each post is represented as a fixed-length vector, the natural way to calculate similarity between different posts is with the cosine similarity metric: $\frac{v_1 \cdot v_2}{||v_1||||v_2||}$ representing how close in pointed direction two posts are to each other in vector space.

Next, for reducing feature matrix dimensionality to some target k before creating interaction terms, we use the sklearn implementations of PCA and truncated Singular Value Decomposition (SVD) [Bui+13]. PCA involves selecting the top k eigenvectors of the matrix $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T}$ (corresponding to the top k eigenvalues by value), where the $x^{(i)}$ values from the original X matrix have already been scaled (had mean subtracted and then divided by standard deviation). These top k eigenvectors correspond to the k vectors for whom when each scaled x is projected onto them, the mean squared projection distance to origin is the greatest (and thus these eigenvectors maximize the capture of variance from the original dataset in a lower dimension). Then to compute any x in this new reduced dimension space, we just need to take Ux, where the ith row of U corresponds to the transpose of the ith greatest eigenvector derived above.

Truncated SVD is similar to PCA and can be shown to be equivalent to PCA if we were to actually subtract feature-wise means from each feature value beforehand [Bui+13]. The practical impact of this is that it can be applied to very large dimensional sparse matrices without densifying them (since it leaves the zeroes as they are), and is thus more appropriate for our large tfidf feature vectors (which otherwise wouldn't fit in memory if densified). More specifically, this algorithm creates an approximation of a given feature matrix X of the form $X \approx U\Sigma V^T$ where $X \in R^{m \times n}$ as before, $U \in R^{m \times k}$, Σ is a diagonal matrix in $R^{k \times k}$ and $V^T \in R^{k \times n}$. $U\Sigma^T$ then becomes our new reduced representation of the X matrix.

Finally, we created interaction terms because we hypothesized that specific unigrams/bigrams are likely only predictive depending on the post topic. However, if we exhaustively covered all of the potential interaction terms we would have ended up with a feature matrix far beyond what our memory could handle. Thus, we used dimension reduction as a way to improve computational tractability. Furthermore, we didn't create every interaction term, but only the second-degree interaction terms between matrices of the form $x_i y_j$ where x_i would come from the first matrix and y_j would come from the second. In total, we created 5 question topic features (using PCA) and 300 word features (using Truncated SVD), taking interaction terms left us with 1500 interaction terms.

Together these features still number in the millions so wittling them down will be important to preventing overfit. To improve computational tractability, we ran forward search on groupings of features (for example one grouping would be the xml tag counts, another would be the topic binary tag features). Specific steps are to first initialize F as the empty set. Next run a for loop where for each feature grouping in the list of groupings, let F_i equal $F \cup$ grouping i, then obtain average of test error from 5-fold cross validation using regularized logistic regression with C = 1.0 (explained below) on F_i, then set F to be the F_i with lowest error obtained from this process. Finally, output top feature subset obtained from all cycles.

The following features were included in the optimal subset (in this order): the word tfidf matrix dimension-reduced to 500 features (we used this to represent the larger tfidf matrix since calculations were taking too long), the xml tag tfidf matrix, the word2vec representation of the answer post, question body text metadata, word2vec representation of the question post, and the log sum of xml tags. To our surprise, interaction terms did not make the cut.

4 Methods: Algorithms and Evaluation

4.1 Pointwise Approach

We first implemented a pointwise approach to our ranking problem, where we treat each individual question-answer pair as a training example, then try to learn which class (good answer or bad one) the question-answer pair belongs to. We used two primary algorithms, regularized logistic regression and regularized support vector machine classification with a linear kernel (linear SVC) and gaussian kernel (Gaussian SVC) as implemented in sklearn ([Bui+13]), primarily because unlike bernoulli naive bayes, multinomial naive bayes, or gaussian discriminant analysis, they do not make as significant requirements on the

distribution/structure of their inputs and have been known to perform well on text classification tasks (especially regularized logistic regression, as Andrew Ng mentioned in lecture 5).

Regularized logistic regression seeks to solve the minimization problem $\min_{w,b} \frac{1}{2}w^T w + C\sum_{i=1}^{m} \log\left(1+\exp(-y^{(i)}(w^T x^{(i)} + b))\right)$, where decreasing the size of the C term means that the relative penalty for having larger coefficients versus reducing error on classifying the data becomes higher. This corresponds to the original logistic regression formulation where we seek to maximize the log joint likelihood of the data where the data is distributed according to the logistic function, with the addition that here we also assume that the w values are themselves distributed normally with a mean of 0, so maximizing $\log P(w)$, and thus some function of $-\frac{1}{2}w^T w$, the same as minimizing $\frac{1}{2}w^T w$, becomes part of our objective. Similarly, note that maximizing the log of the logistic function $\log \frac{1}{1+\exp(-z)}$ is equivalent to minimizing $\log(1+\exp(-z))$ if we expand the log term. Note that in this formulation an l2 norm is used as the penalty, but it can also be replaced by an l1 norm.

Support Vector Machine (SVM) seeks to solve the primal minimization problem $\min_{w,b,\zeta} \frac{1}{2}w^T w + C\sum_{i=1}^{m} \zeta^{(i)}$, where decreasing the size of the C term also implies that the relative penalty for larger coefficients versus misclassified samples becomes higher. It requires the constraints: $y^{(i)}(w^T \phi(x^{(i)}) + b) \geq 1 - \zeta^{(i)}$ and $\zeta^{(i)} \geq 0$, $\forall i$. This corresponds also to the original formulation of SVC where we seek to identify the hyperplane that maximizes the margin between all data points and the hyperplane, with the adjustment that instead of requiring complete linear separability, we impose a penalty of $C\zeta^{(i)}$ if point i is classified with a margin of less than 1, or 1 - $\zeta^{(i)}$ where $\zeta^{(i)} \geq 0$. In this formulation an l2 norm is used as the penalty. Next, using Lagrangians and KKT conditions we can derive the dual formulation as follows:

$$max_\alpha W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

With the constraints $0 \leq \alpha_i \leq C$ holding $\forall i$ and $\sum_i \alpha_i y^{(i)} = 0$. And finally, the term in the angular brackets is the kernel, which we use both a linear formulation for (of form $x^T z$) as well as a gaussian formulation for (of form $\exp(-\gamma||x-z||^2)$) allowing us to implicitly map to an infinite dimensional feature mapping of x.

4.2 Pairwise Approach

We also implemented a pairwise approach to our ranking problem, where we train a model to recognize, given a pair of answers to a given question, which answer is better. This requires transforming the data set so that the new answer pair feature vectors are just the difference between the two individual answer feature vectors with different class labels from the same question. The new class label for the answer pair {+1, -1} then indicates which feature vector should be ranked higher.

Concretely, assuming there are three answers to a question, the corresponding training set of the question can be denoted as $\{(y_1, x_1), (y_2, x_2), (y_3, x_3)\}$, where $y_i (i = 1, 2, 3)$ is the class label of the question-answer pair defined in 3.2 and x_i (i = 1, 2, 3) is the corresponding feature vector. The new answer-pair based set of feature vectors then becomes $\{(x_1 - x_2), (x_2 - x_3), (x_1 - x_3)\}$. Finally, the label for $(x_1 - x_2)$ is +1 if $y_1 > y_2$, -1 if $y_1 < y_2$. If $y_1 = y_2$, which means these two vectors are in the same class, we don't include them in the training set. [HGO00]

Finally, we use RankingSVM with linear/gaussian kernel and pairwise logistic regression with l1/l2 norms. Both use the same optimization functions described in 4.1, but with the newly transformed features and labels described above.

4.3 Evaluation method

To evaluate our ranking model's performance, we treated it as an information retrieval (IR) system that seeks to return results with the good answers (class 1) being ranked (and thus appearing) before the bad ones (class 0), reflecting a real life situation where we would expect a community Q&A website to place the best results first on the page. Accuracy is then based on comparing the ranking our IR system returns to the ground truth ranking, with a focus on where the class 1 answers are located in the ranking. Especially for binary response variables, research literature commonly uses Mean Average Precision (MAP) for this purpose, so we do so as well.

Specifically, for each question and for every position of a good answer in the ranked sequence of answers sorted by model score, we calculate an average precision (AP). Next, we average AP across all test set questions. Mathematically,

$$MAP(Q) = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{1}{m_i}\sum_{k=1}^{m_i} Precision(A_{ik})$$

where $|Q|$ denotes how many questions in the test set, m_i denotes how many answers in the i^{th} question A_{ik} denotes a set of ranked answers from the top answer until k^{th} answer

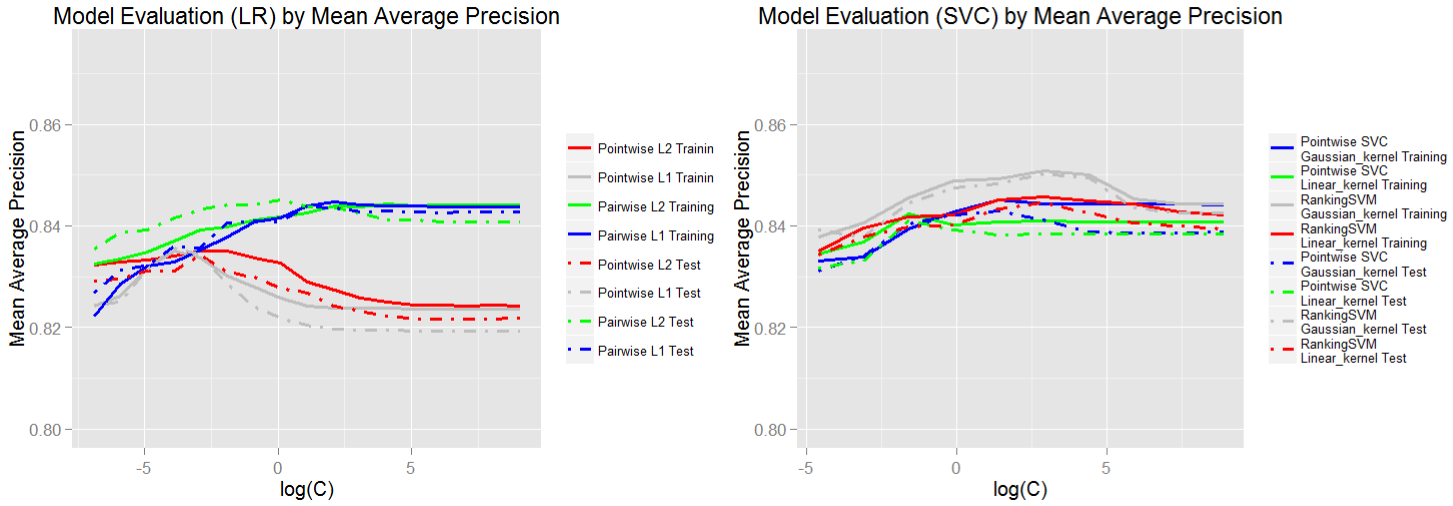
In order to carry out this evaluation, we needed a dataset organized by questions, where each question had at least two answers and these answers weren't all good or all bad. Thus, we removed questions with only one answer and also deleted questions whose answers are all good (or bad). This left us with 21,779 questions and 63,685 child answers. Next, for each individual experimental run, we randomly selected 20% of the questions from the whole dataset as a test set, and applied 5-fold cross validation (CV) to the remaining 80% of the data: We fit algorithms on the training set, used the validation

set to choose hyper-parameters based on MAP performance, then averaged the optimal hyper-parameters obtained on each fold of cross validation, and applied them to the test set to get an estimated MAP. We repeated the experimental strategy multiple times then aggregated the estimated MAP for each algorithm across all experiments.

Finally, in each experiment, we applied both the pointwise approach and pairwise approach (both approaches include logistic regression with L1/L2, SVM with linear/Gaussian kernel) to the same training set and estimate hyperparameters from the same validation data. For logistic regression and SVM with linear kernel, the hyperparameter is C (please see section 4.1). For SVM with Gaussian kernel, the hyperparameters are C and γ. We used grid search to find the optimal hyperparameters. Our C candidates are from 0.0001 to 1000 with equal steps in logarithm. γ candidates are 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10.

5 Results and Discussion

We found that the optimal C are quite different between pointwise approach and pairwise approach. Figure 1 shows MAP vs. different C for applying different models to training and test set. For example, optimal C of logistic regression (L1) is 2.98 from pairwise approach, but 0.02 from pointwise approach (see Figure 1 (a)). Furthermore, the optimal γ of SVM Gaussian kernel is 0.0001 in most cases. To simplify the visualization of hyperparamter tuning, we fixed $\gamma = 0.0001$ and visualized trends of MAP across different C in Figure 1.



(a) Logistic Regression: MAP vs. C

(b) SVC: MAP vs. C

Figure 1: Outputs versus hyperparameter choice

As the image shows, all the methods we implemented do not have obvious overfitting issue when we chose appropriate hyperparameters. Overall, the Pairwise approach did better than pointwise approach in terms of MAP. Between the two pairwise (or pointwise), logistic regression with L2 norm outperformed L1 norm, and RankingSVM with Gaussian kernel outperformed linear kernel. For logistic regression, this may be because L1 tends to delete insignificant features while L2 may shrink some coefficients close to zero but won't delete them altogether; for SVM, the Gaussian kernel takes interactions among features into account which we hypothesized earlier should be important. However, if we consider training time, logistic regression only took 4 to 7 minutes, pointwise SVM took 2 to 4 hours, RankingSVM took even longer (8+ hours) since the training size of the pairwise approach is much larger.

6 Conclusions and Next Steps

As variance was not our main challenge, our next step will be to reduce bias. The most obvious way to fix bias is with better features. Currently our features are largely bag of words (not taking into account syntactic ordering), don't try to uncover semantic meaning, and are just based on the raw text itself, not using any third party authoritative source to determine whether posts accurately answer questions or not. Another direction to consider is, especially as we scale this approach to larger data sets with millions of posts (Stack Overflow), trying to use different neural network architectures that can more effectively capture the nonlinear interactions between features. We began to do this with the Gaussian kernel but can likely do more, seeing as the boost from linear to Gaussian kernel on SVC was only around 1%. Finally, doing any of this will require moving to a much faster and more parallelized architecture, as time was a major bottleneck for us even with the current feature set/models!

References

[HGO00] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. "Large Margin Rank Boundaries for Ordinal Regression". In: MIT Press, Jan. 2000. Chap. 7, pp. 115–132. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=65610.

[Bui+13] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[LG14] Omer Levy and Yoav Goldberg. "Dependency-Based Word Embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 302–308. URL: http://www.aclweb.org/anthology/P14-2050.

[Bar+15] Alberto Barrón-Cedeño et al. "Thread-Level Information for Comment Classification in Community Question Answering". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 687–693. URL: http://www.aclweb.org/anthology/P15-2113.

[Co15] Syllogism Co. *spaCy*. 2015. URL: https://spacy.io.

[Hou+15] Yongshuai Hou et al. "HITSZ-ICRC: Exploiting Classification Approach for Answer Selection in Community Question Answering". In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, Colorado: Association for Computational Linguistics, June 2015, pp. 196–202. URL: http://www.aclweb.org/anthology/S15-2035.

[Inc15] Stack Exchange Inc. *Stack Exchange Data Dump*. Aug. 2015. URL: https://archive.org/details/stackexchange.

[Nak+15] Preslav Nakov et al., eds. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, Colorado: Association for Computational Linguistics, June 2015. URL: http://www.aclweb.org/anthology/S15-2.

[Nic+15] Massimo Nicosia et al. "QCRI: Answer Selection for Community Question Answering - Experiments for Arabic and English". In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, Colorado: Association for Computational Linguistics, June 2015, pp. 203–209. URL: http://www.aclweb.org/anthology/S15-2036.

[Tra+15] Quan Hung Tran et al. "JAIST: Combining multiple features for Answer Selection in Community Question Answering". In: *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*. Denver, Colorado: Association for Computational Linguistics, June 2015, pp. 215–219. URL: http://www.aclweb.org/anthology/S15-2038.

[Zho+15] Xiaoqiang Zhou et al. "Answer Sequence Learning with Neural Networks for Answer Selection in Community Question Answering". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 713–718. URL: http://www.aclweb.org/anthology/P15-2117.