Multi-Factor Duplicate Question Detection in Stack Overflow

Team Name: Hyperplane Scavengers

DataSet Curation

- Official MSR Dataset: Using techniques in paper, number of duplicates detected were only 2
- Finally resorted to using the Stack Exchange Data dump which contained data till 2021
- Postlinks.xml contained relationship for duplicate detection
- Postgres database for easy handling, sorting of data, visualizations and error analysis

Preprocessing

- Everything lowercase and the phrase Duplicate was removed from the title
- **<blockquote>** tag removed (since it contained the links to “duplicate”)
- All HTML tags removed but the data was preserved except **<code>**
- All whitespace converted to single space
- Punctuation except **#** removed, special case - C#
- Stemming
- Stopword Removal
- Tokenization

DupPredictor Model

4 Components: Title, Body, Topics (from Title + Body) and Tags

- Bag of Words and Cosine Similarity
  - Bag of Words Embedding for each component
  - Cosine Similarity for each component
- LDA
  - Get trend of the conversation, try to extract some meaning from the text
  - Topic modelling on Title + Body
  - Gensim Library with K = 100
- Greedy Composer
  - Random Restart Approach
  - Greedy Approach with params increasing in steps of 0.05
  - Weighted Sum and top K (K = 10, 20) are stored

Paper Results

Evaluation Metric

- Recall-Rate@K

$$recall - rate@k = \frac{N_{detected}}{N_{total}}$$

- $N_{detected}$ is the number of duplicate questions whose master was in top K
- N_{total} is the total number of duplicate questions

Research Question 1: How good is DupPredictor over its 4 individual components?

- Is our model good?
- Measure recall-rate@K, K = 10 and 20 of DupPredictor with its single components

- DupPredictor performs very good as compared to its other individual components and by a large margin

Algorithm	Recall Rate@10	Improvement (%)
DupPredictor	0.56	0.0
Title similarity	0.33	69.69
Description similarity	0.2	180
Topic similarity	0.08	600
Tag similarity	0.28	100

Algorithm	Recall Rate@20	Improvement (%)
DupPredictor	0.65	0.0
Title similarity	0.42	54.76
Description similarity	0.26	150
Topic similarity	0.14	364.28
Tag similarity	0.34	91.17

Research Question 2: Effect of Varying Number of Training Questions

- How many questions is enough for training due to the high compute?
- Trained the model over 2 training set sizes, the first 100 and 300 questions. Tested on the same testing set (300-400) for uniformity. Measure the recall-rate@10 and recall-rate@20
- General parameter trend remains the same

$$\alpha \geq \beta \geq \delta \geq \gamma$$

Trained on first	Alpha (Title)	Beta (Body)	Gamma (Topic)	Delta (Tags)
K = 10				
100	0.5	0.5	0.0454	0.45
300	1.0	0.6	0.3	0.6
K = 20				
100	0.4	0.5	0.13738	0.1992
300	0.9	0.85	0.3153	0.3

Trained on first	Recall@10 Training	Recall@10 Testing	Recall@20 Training	Recall@20 Testing
100	0.74	0.55	0.78	0.62
300	0.67	0.56	0.74	0.65

- Model trained on 300 performs better on test set so it must be generalizing better
- However performance difference is negligible
- Implies: Model is able to learn and tune its parameters even on a lesser amount of data and hence we can restrict training to 300 questions

Research Question 3: Does DupPredictor estimate the 4 weights of its 4 constituent components well?

- Is greedy random restart method accurate and well predicted?
- Randomly generated 50 sets of α, β, γ, δ

- Simplistic approach outperforms all randomly generated weights.
- If the random generated params follows the trend $\alpha \geq \beta \geq \delta \geq \gamma$ and $\alpha \approx 1$, then model performs best
- Worst performing give to little attention to title, or too much attention to topic

alpha	beta	gamma	delta	recall-rate@10	recall-rate@20
0.9	***0.85***	***0.3152***	***0.3***	***0.56***	***0.64***
0.89	0.0687	0.29	0.2593	0.53	0.62
0.8118	0.5148	0.6062	0.3224	0.53	0.61
0.2605	0.88	0.6911	0.08	0.36	0.45
0.1749	0.4582	0.7276	0.0998	0.36	0.45

Error Analysis And Basic Stats

Pruning the search space improves performance on multiple fronts

- 94/100 questions in the test set had at least one tag in common with their duplicates.
- The 6 cases where tags were not common were very subtle misses.

- For eg:

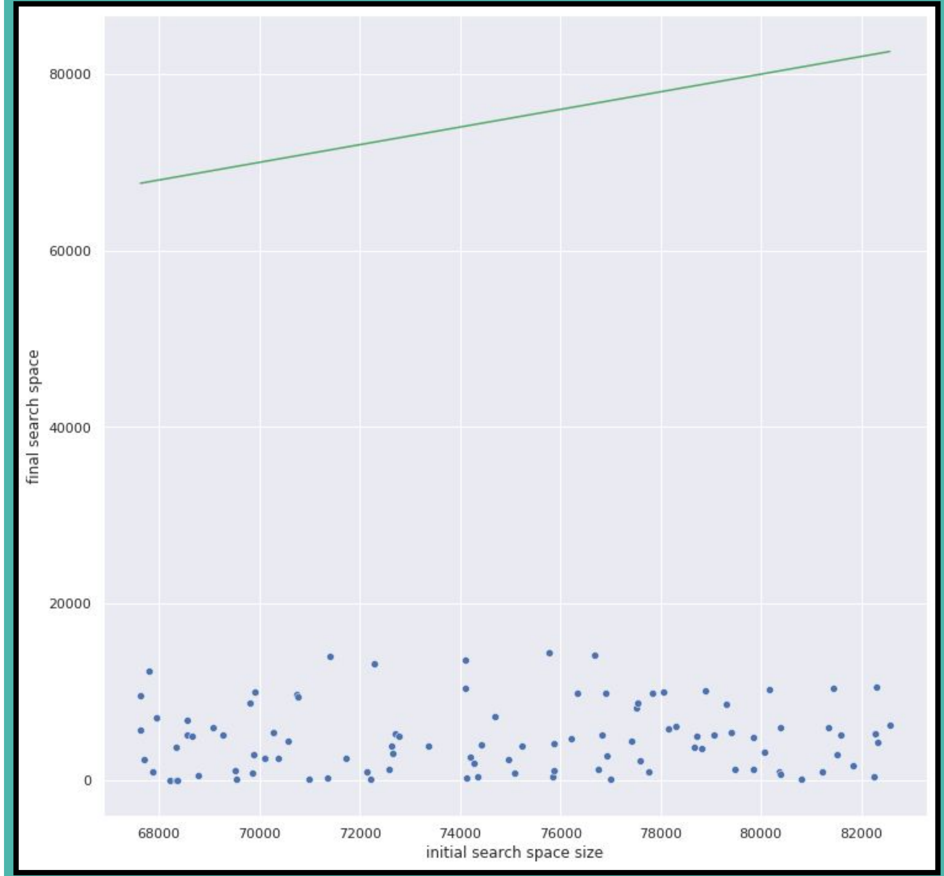
Tags of a question with ID 441910:

internet-explorer image firefox rendering

Tags of its duplicate ie 130161:

html css internet-explorer-6 png

- Figure on the right shows the massive reduction in search space size with 2 benefits:
    - Less time to evaluate candidates in search space
    - Higher rank for an actual duplicate

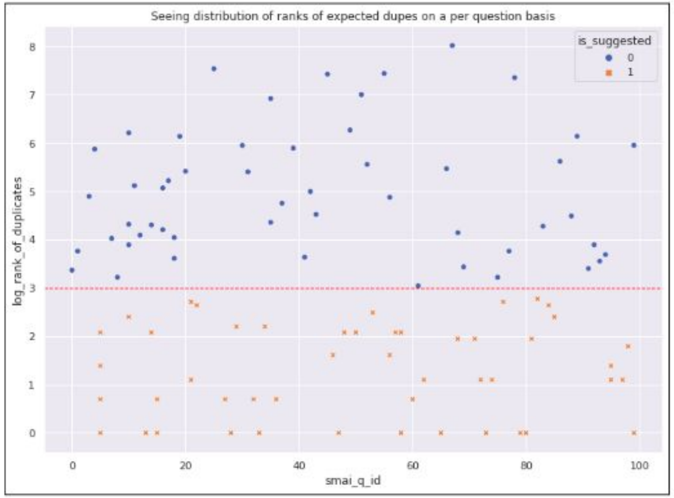


SUMMARY OF ABLATION STUDY SCORES (at Recall@20)

Model description	Training Recall@20	Test Recall@20		Mean rank of highest ranked duplicate		Best rank of a duplicate		Worst rank of a duplicate		Parameters
		Without Pruning	With pruning	Without Pruning	With pruning	Without Pruning	With pruning	Without Pruning	With pruning	
K=20 with all components	0.74	0.65	**0.68**	175.84	99.9	1	1	2365	1359	[0.9, 0.85, 0.31, 0.3, 0.0]
K =20 with all components except Title Similarity	0.61	0.53	0.53	454	287	1	1	12366	4048	[0, 0.85, 0.35, 0.64, 0.0]
K =20 with all components except Topic Similarity	0.733	0.61	0.61	229	160	1	1	3691	3613	[0.2, 0.19, 0, 0.07, 0.0]
K =20 with all components except Body Similarity	0.707	0.61	0.62	214	120	1	1	4058	1972	[1.0, 0, 0.45, 0.6, 0.0]
K =20 with all components except Tags Similarity	0.67	0.51	0.62	540	122	1	1	25818	3177	[0.9, 0.8, 0.1, 0, 0.0]
K=20 with all components + Jaccard Coeff included	**0.763**	0.65	**0.67**	**105**	**78**	1	1	2054	1197	[0.45, 0.35, 0.25, 0.29, 1.0]

Some insights based on the table above and the detailed analysis in the next section

- As per the ablation study, the most significant parts for detection seem to be **"title" and the "tags"**.
- Pruning the search space is able to improve results and is quicker computationally
- The most significant increase in results after pruning was in the case where tag similarity was discarded. This was as pruning based on tags does a job ALMOST similar to tag similarity.
- With moderators strictly enforcing titles to be precise and accurate, "title similarity" seems to be a great indicator.
- TAGS become important as it helps in **restricting search to relevant languages and technologies.** Eg: a question regarding retrieval queries on MySQL is likely to be similar in body and title when compared to the same question wrt MongoDB. However, the two questions being different tagged is what is allowing the model to successfully differentiate between the two.
- In the adjoining figure, x: question id ; y : rank of one of the expected duplicates. Since a question can have more than one expected duplicate, there may be multiple dots on the same vertical line. The red line depicts the threshold (20) beyond which questions are not suggested.

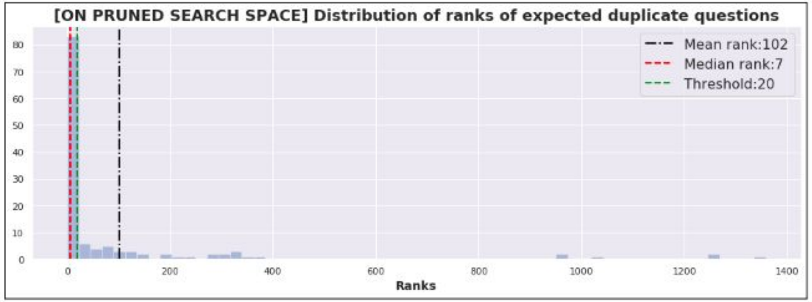


ALL 4 components (title + body + tags + topics) involved

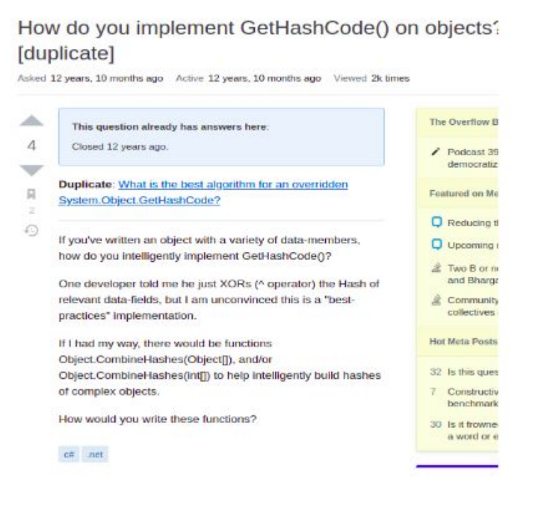
In general, the 0.68% accuracy seems decent.

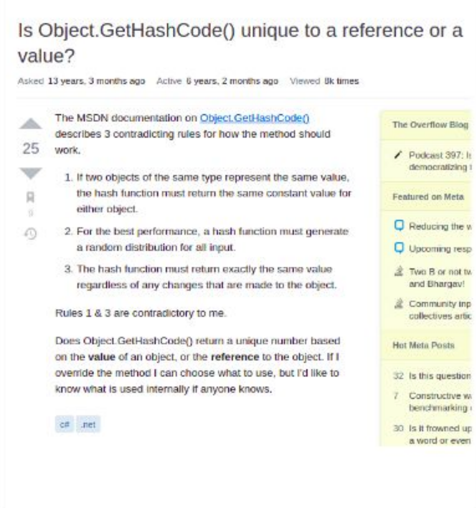
Params: [0.9, 0.85, 0.3125, 0.3] i.e. high importance to title and body text

- qid for whom we want a duplicate: 441529
- Link to qid: LINK
- **Actual best candidate:** 263400
    - scores: [0.51, 0.11, 0.5 , 0.4]
    - final score: 0.85 | rank 68
- **Chosen best candidate:** 34505
    - Scores: [0.5, 0.5, 0.7, 1]
    - final score: 1.4287 | rank : 1
- We see that the q_chosen beats q_expected in all 4 criteria.
- **Conclusion:** The model is misled by the use of common words like **"objects" and "hash function"** in q_asked and q_chosen. Even though q_expected is clearly a better match; but the model fails to capture the fact that the q_asked is actually discussing an algorithm and naively allots q_chosen as the higher score (despite it being very different) just because of common words. This can be improved by considering synonyms or word embedding such that **phrases like "intelligently built" etc can be linked to words like "algorithm"**.



question asked	Expected best	Chosen best



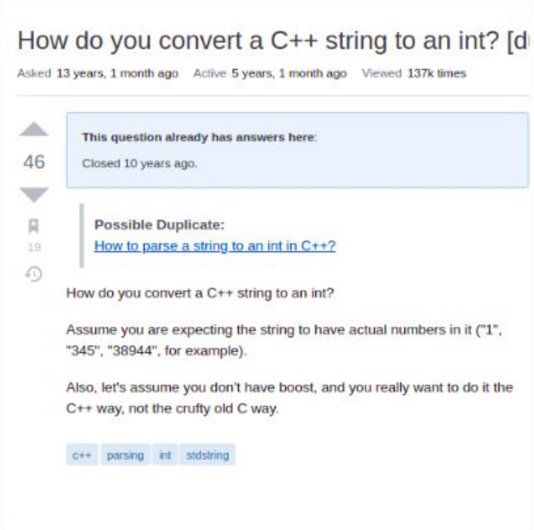


ALL components (Except Title Similarity) involved

In general, the 0.53% accuracy WITHOUT using the title is reasonable.

Params: [0 , 0.85, 0.35, 0.64] i.e. without the title, the content now shifts attention to TAGS.

- qid for whom we want a duplicate: 508191 | Link to qid: LINK
- **Actual best candidate:** 200090
    - scores: [0.67   , 0, 0.95 , 0.5]
    - final score: 0.65 | rank 62
- **Chosen best candidate:** 275733
    - Scores: [0.4, 0.129, 0.88, 0.7]
    - final score: 0.87 | rank : 1
- q_chosen beats q_expected in “body_score” which has a high weightage of 0.85.
- **Conclusion:** Due to removal of title, q_expected loses out. Also, because the body in q_asked is almost entirely empty (since we remove the code before inputting to model), there are hardly any words in the “filtered body” of q_asked to give any meaningful insights for matching. However, since the coefficient for the body is high, q_chosen overtakes q_expected. The model where title similarity exists is able to CORRECTLY detect this test case.

question asked	Expected best	Chosen best
		

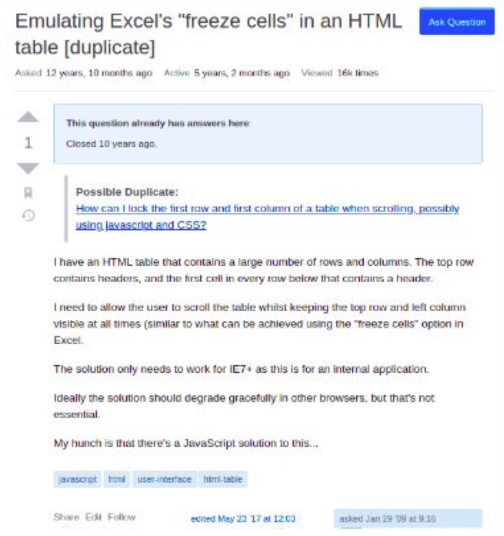
ALL components (Except Body Similarity) involved

In general, the 0.62% accuracy WITHOUT using the body shows that BODY does not play a major role in helping the detector..

Params: [1, 0, 0.5 , 0.6] with emphasis on TITLE and TAGS.

- qid for whom we want a duplicate: 490973 | Link to qid: LINK
- **Actual best candidate:** 296020
    - scores: [0.15, 0.6, 0.7 , 0.4]
    - final score: 0.8 | rank 42
- **Chosen best candidate:** 150606
    - Scores: [0.36, 0.26, 0.65, 0.7]
    - final score: 1.11 | rank : 1
- Conclusion: The question wants an answer on how to FREEZE cells in EXCEL using HTML/CSS. Q_chosen gives an answer which is **NOT EVEN REMOTELY related** to EXCEL. Clearly, the model fails to capture the "excel" factor and hence, blunders. The BODY of q_expected and q_ask match to a large extent to give a score of 0.6 . But since coeff for body score is ZERO in this model, the test case gets misdetected by the model.

question asked



Expected best

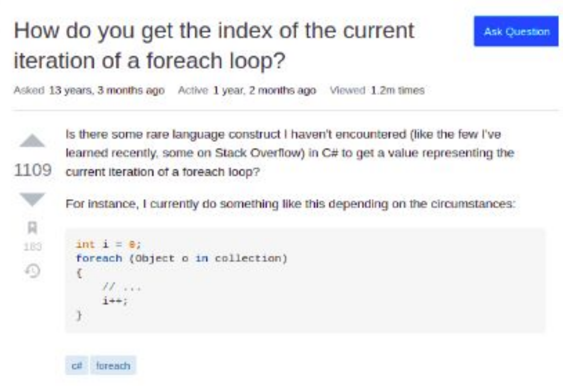
Chosen best

ALL components (Except TAGS Similarity) involved

In general, the accuracy without using tags is 0.51%. But in the pruned search space, tags are automatically taken care of and hence, leads to an accuracy of 0.62% (very minor drop) when tags are not considered.

Params: [0.9 , 0.8 , 0.1 , 0] with emphasis on TITLE and BODY.

- **qid for whom we want a duplicate:** 521687 | Link to qid: LINK
- **Actual best candidate:** 43021
  - scores: [0.5, 0.08, 0.35 , 1]
  - final score: 0.77 | rank 28
- **Chosen best candidate:** 141108
  - Scores: [0.8, 0.25, 0.64, 0.4]
  - final score: 0.999 | rank : 1
- **Conclusion:** q_ask is concerned with foreach in C#. Q_expected is related to foreach in C# but q_chosen is related to foreach in PHP. Both of them are not filtered as both of them have the "foreach" tag common with q_ask. Q_expected has exactly the same tags as q_asked but q_chosen is predicted (even though it **deals with a totally different language**). This is due to the zero value of coefficients of TAGS_SCORE.

<table>
<tr><th>question asked</th><th>Expected best</th><th>Chosen best</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
</table>

Further Explorations

- Attempted to further improve the best results obtained.
- Mainly experiment with 2 techniques
  - Increasing the number of features while calculating the composer score
  - Further improving the predictions with the help of NLP

Jaccard Coefficient

- Composer is treating each of its similarities as separate entities
- Use Jaccard to get a holistic view
- combined the processed title, body and tags as BOW

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- Included a fifth parameter, Jaccard similarity
- Model predicted jaccard to be very important
- Though training recall increased from 0.74 to 0.76, testing recall didn't increase.
- Conclusion: Title, body and tags are disjoint

BERT embeddings

- Reranking smaller pruned rank list (by DupPredictor)
- Top 30 predictions
- Actual masters not in top 30 appended at the end
- Computed bert embeddings for title and body text

$$BertScore = \alpha \times CosineSim(BERT(Q1_{Title}), BERT(Q2_{Title})) + \beta \times CosineSim(BERT(Q1_{Body}), BERT(Q2_{Body}))$$

- Improved recall-rate@20 - from **0.65 to 0.88**
- For **29** duplicate, improved richness by including **32** more correct candidate questions
- Able to include **2** questions in top 20 - initially **not even in top 30**

Website

Live Demo

Limitations

- Size of training and testing set
  - Huge number of duplicate questions
  - Had to compute large number of pairwise scores
    - Training set - 60,000 candidate questions per duplicate - 8 hours time
    - Test set - 80,000 candidate questions per duplicate - 5 hours time
  - Including more duplicate became tough
- Estimation of parameters
  - Able to compute majority of parameters, but greedy approach costs 4 hours compute time per training.
- Bert Training
  - Embedding creation for each post was compute heavy
  - Approx 20 Hours with 14GB RAM. Laptops started hanging
  - Resorted to the re-ranking approach - reduced search space

Thanks