Piazza Search Project Report

Julia Aoun jcaoun@umich.edu
Chimmuanya Iheanyi-Igwe ciheanyi@umich.edu
Mariam Mahmoud mariamhm@umich.edu
Melina O'Dell melodell@umich.edu
Sydney Swider swiders@umich.edu

1 Project Description

1.1 Current Problem

The motivation for building an improved Piazza search engine is the frustration we have personally experienced, both as students and TAs, when trying to search for posts related to our questions or our students' questions on Piazza. Piazza is a tool that all UMich CS students interact with on a daily basis for asking questions, studying, and interacting with other students. But, the existing search functionality is poorly designed and continues to hinder students and instructors ability to quickly find posts and questions.

Currently, Piazza returns query results ordered by the recency of the post, and only if the post includes the exact matches with individual words in the query, ignoring the meaning of specific combinations of words. It also does not account for misspelled words, which can lead to an especially frustrating experience when trying to search for related topics as only exact matches with those misspellings will be retrieved. Trying to search for an existing post that the user has seen before is unnecessarily difficult. For example, if an IA wants to refer a student back to a previous post from another student, the search functionality is so ineffective that it does not help with finding it. This results in the TA having to search through posts manually. The consistent low quality of Piazza's search results is disappointing for all users.

Over multiple semesters (and even within the same semester), students often ask similar questions about specific projects, exams, or commonly difficult homeworks. Piazza does not currently offer a way to compile a list of "Frequently Asked Questions" based on this redundancy. We believe that would be extremely beneficial for students, as they can easily find and reference the answers to common questions without waiting for the instructors to rewrite the answer. It also benefits instructors because the FAQ could be automatically generated, instead of instructors having to manually update it during the semester. This helps to reduce the overall workload, while also limiting the number of duplicate questions instructors may see around certain topics.

We would like to save students and instructors time, effort, and frustration when searching Piazza by building a smarter search functionality to mitigate these problems.

1.2 Proposed Solution and Approach

This project is focused on creating a better search feature for Piazza. With improved algorithms and a user interface, both students and instructors will be able to pass in a query and retrieve any and all posts relevant to their search. In addition to this, we use K-means clustering to generate a "FAQ" page for each topic based on the most popular questions asked about that topic during previous semesters. We hope the availability of this topic summary reduces duplicate question load when projects and assignments are released throughout the semester.

We scraped Piazza question and answer data for one course and implemented three models with increasing levels of specificity before evaluating their precision and recall: The boolean model, the vector space model with weighting scheme `tfc nfx`, and the vector space model with weighting scheme `nxx bpx`. We also decided to make two versions of the inverted index used for these calculations: One with bigrams and one with unigrams. We analyzed the differences between the precision and recall for each model for each type of inverted index to see which provided better results. Evaluation of the ground truth for the dataset was done manually by identifying relevant documents for each of the training queries.

For demonstration purposes, we built a simple UI (Similar to the current Piazza interface) to interact with the search feature. We used scraped Piazza course datasets to create databases, which allows users to select from different courses to search for the query. We also built an API to call the search functions and populate the ranked results based on the user-inputted query. Unlike Piazza, our UI sorts relevant documents based on similarity, rather than recency of the post. This UI also displays the FAQ topics generated for the selected course.

2 Related Work

There has been lots of work using NLP techniques to classify and compute the similarity between forum posts, including coding-specific platforms. This work is relevant to our project, as the EECS Piazza data we are working with is dominated by code-related questions and answers.

2.1 Document Summaries

Work by Abric et al. (2019) analyzed questions on Stack Overflow, a well-known social platform for asking and answering questions related to software development. They used TF-IDF, cosine similarity, and other NLP techniques to analyze similar posts, determine if they were duplicates, and assess the quality of these duplicates against each other. This is similar to the work we are trying to do with the FAQ section. We will use similar weighting schemes and text analysis methods to compare Piazza questions and answers, conglomerating the most similar and relevant for each project into a summarized collection.

Generating relevant and comprehensible summaries from extracted text data is known to be a challenging problem. There have been many proposed summarization algorithms to optimize automatic labeling of topics, such as the one designed in work by Wan and Wang (2016). The FAQ generation implementation uses similar but simpler document summarization techniques, grouping together similar data entries and extracting the most relevant and encompassing words as "summaries." We use the K-means clustering algorithm to group related Piazza posts together (as well as incorporating manual classifications provided by Piazza's existing folder feature). Rashid et al. (2020) investigates a similar use for K-means and reported success with categorizing and topic assignment to data. By combining these techniques, we hope to achieve sufficient topic clustering and summarization to demonstrate the potential for improving class forum data.

2.2 Similarity Measures

Similar classification techniques were also used for creating AnswerBot, a tool for summarizing the answers to similar or duplicate Stack Overflow questions given a query (Cai et al. (2019)). More examples of this can be observed in work by Distante et al. (2015) relating and suggesting similar topics based on forum content. We will be using similar approaches as these groups did to process and analyze the data we scrape from Piazza. Vector space models and TF-IDF are known to work well on social media data, as shown in work by Venekoski et al. (2016) classifying social media texts. They tried out multiple weighting schemes, a comparison that we also intend to make, to compare the effectiveness of each scheme against each other for this specific type of data.

3 Data Collection and Annotation Methods

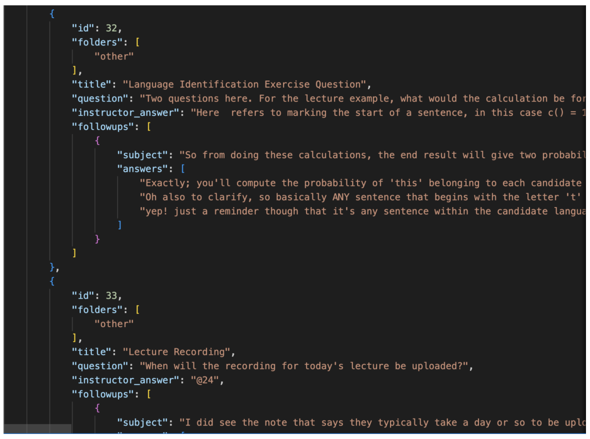
To scrape post data from Piazza courses, we wrote a Python script that makes calls to the Piazza API and collects the following information from every Piazza post in the specified dataset:

- Post ID
- Folders the post was tagged with
- Post title
- Post question
- Student answer (if one was given)
- Instructor answer (if one was given)
- All follow ups

The script takes in a range of post IDs, gathers all the data, and writes to a JSON file (named `<class_name>_posts.json`). Figure 1 shows an example JSON file with this semester's EECS 486 Piazza data.

This raw data serves as input to a computation pipeline that preprocesses the text and builds an inverted index. We use case-transformation and tokenization to preprocess the post data before generating the inverted index. The inverted index contains term frequency and inverse document frequency values for all terms in all documents to be used for computing similarity scores.

After scraping all the posts, we manually created 50 test queries on various topics in the class



```
{
    "id": 32,
    "folders": [
        "other"
    ],
    "title": "Language Identification Exercise Question",
    "question": "Two questions here. For the lecture example, what would the calculation be for
    "instructor_answer": "Here  refers to marking the start of a sentence, in this case c() = 1
    "followups": [
        {
            "subject": "So from doing these calculations, the end result will give two probabil
            "answers": [
                "Exactly; you'll compute the probability of 'this' belonging to each candidate
                "Oh also to clarify, so basically ANY sentence that begins with the letter 't'
                "yep! just a reminder though that it's any sentence within the candidate langua
            ]
        }
    ]
},
{
    "id": 33,
    "folders": [
        "other"
    ],
    "title": "Lecture Recording",
    "question": "When will the recording for today's lecture be uploaded?",
    "instructor_answer": "@24",
    "followups": [
        {
            "subject": "I did see the note that says they typically take a day or so to be uplo
```

Figure 1: Sample JSON data from the W23 486 Piazza

with varying degrees of specificity. Once we gathered the test queries, we had two members of the group manually search through all of the posts to determine which were relevant to each query. We stored this information in a spreadsheet to calculate and plot the precision and recall of the system. We focused on posts that shared key words with the query and used our domain knowledge to determine if posts with the same key words as the query were actually relevant or if they would potentially answer the question. Each query, on average, had between 15-25 relevant posts related to it, with more relevant posts for the general queries and fewer posts for specific queries. These classifications serve as the ground truth for the IR system.

4 Method Description

4.1 Building the Corpus

To add documents to the corpus, we scraped public Piazza posts from the EECS 370 Winter 2022 semester, which included approximately 5000 posts. We used sample questions from the EECS 370 Spring 2022 semester as test queries because the dataset was smaller, including only 500 questions. Each post on Piazza is tagged with a certain topic (also called a "folder"). We took a portion of the questions from each topic to include variety in the 50 queries. Once we chose the best 50 queries. After gathering the queries, we manually went through the documents with the same topic of the query and decided whether or not that document was relevant to the query or not. These human judgements are used as the ground truth for the dataset when we calculate precision and recall.

4.2 Preprocessing and Spellcheck Implementation

The initial preprocessing follows a similar methodology from Assignment 1. We scrape text data from Piazza, taking the questions, student and/or instructor answers, and follow ups. We then tokenize the text, using the Python contractions library to break up any contractions. We remove stopwords from the tokenized text, using the base stopword list from the course as well as specific words we manually decided did not provide any helpful context to the posts when inspecting intermediate test results.

In addition to this preprocessing, we created and applied a spellcheck to the documents and queries. The spellcheck process consists of building a dictionary using a starter set of words from the `nltk words` set, and a set of every word that appears more than 5 times in the corpus. Choosing 5 as the cutoff was partially arbitrary. We looked through the frequency of different words in the data and observed that using values below 5 would include too many misspelled words, which we did not want to add to the dictionary of "correct" words. This allowed us to identify misspelled words as those that were not present in the dictionary. The spellcheck program then finds the word in the dictionary that has the highest similarity using difflab's SequenceMatcher.

Initial preprocessing along with spellcheck made the search feature more resistant to typos and allowed for more meaningful links between similar words.

4.3 Building the Inverted Index Using a MapReduce Pipeline

To build the inverted index for perform weighting scheme calculations, we designed a pipeline (sequence) of MapReduce programs. These programs calculate the term frequency and inverse document frequency for each term given an input corpus of documents (In our case, the JSON list of Piazza post data) and output an inverted index, also represented as JSON. JSON data is very easy to work with for calculations, making it the best choice for our data representations. MapReduce programming is best suited for processing and performing calculations on large amounts of data (In fact, Google invented and uses MapReduce to do batch data processing for their search engine, as noted in the original MapReduce paper, Dean and

Ghemawat (2004)). Although we do not have access to a real distributed system for this project, which would have allowed us to take advantage of the power of parallel programming, we will implement the inverted index creation this way as a proof-of-concept. Given the resources, these inverted index programs could be scaled to massive amounts of input data and perform these computations in a reasonable amount of time.

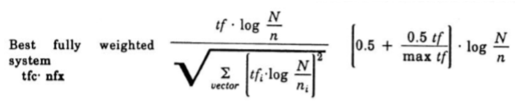
4.4 Models and Evaluation Methodology

Our evaluation methodology consists of precision and recall metrics, based on the relevant judgements for each document we collected for each of the queries. We had two teammates evaluate which documents from the Winter 2022 dataset are relevant to each query. These judgments may include some bias, given that we had only two (human) teammates judging the dataset for the test queries, but we attempted to minimize this discrepancy by comparing judgements between the two teammates (as opposed to only following the judgment of one). Both teammates double-checked a random subset of the other's relevant documents to validate each other's judgments.

We used three different models to evaluate relevant documents for the test queries. We compared their precision and recall values in order to choose the "best" model to use with the user-facing search feature demonstration. Each model was evaluated using both unigram and bigram inverted indices.

Boolean: With the boolean search model, we will return all documents that have a word in common with the query.

`tfc nfx` (TF-IDF): Our second iteration will implement standard TF-IDF, using `tfc` and `nfx` weighting for documents and queries respectively (See Figure 2).



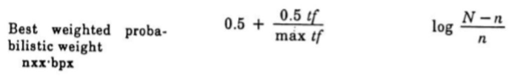
Best fully weighted system tfc· nfx	$\frac{tf \cdot \log \frac{N}{n}}{\sqrt{\sum_{vector} \left[tf_i \cdot \log \frac{N}{n_i}\right]^2}}$	$\left(0.5 + \frac{0.5\, tf}{\max tf}\right) \cdot \log \frac{N}{n}$

Figure 2: The Salton-Buckley `tfc nfx` weighting scheme

`nxx bpx`: Our last iteration will implement the weighting `nxx bpx` weighting for documents and queries respectively (See Figure 3).

4.5 Using K-Means Clustering for FAQs

Aside from implementing a better Piazza search functionality, we believed that creating an addi-



Best weighted probabilistic weight nxx·bpx	$0.5 + \frac{0.5\, tf}{\max tf}$	$\log \frac{N-n}{n}$

Figure 3: The Salton-Buckley `nxx bpx` weighting scheme

tional FAQ generation tool would be beneficial for both students and instructors. This tool computes similarities between documents, clusters relevant documents together, and extracts keywords from each cluster to summarize each subset of documents. This can be used to create an FAQ post for each project or assignment that will allow users to conveniently find posts that are informative of broad topics.

Before generating the actual clusters, we find an optimal number of clusters between 2 and 20, incremented by 2 (2, 4, 6, 8...). We do this by running the K-means algorithm to generate each different number of clusters and graph the sum of squared error for each (See Figure 6 in the "Results" section). Finding the elbow point on the graph gives an estimate for the optimal number of clusters, which we use when generating the FAQ.

Once the clusters are generated, we extract keywords from each cluster using `sklearn's get_feature_names_out` function. This allows us to find potential keywords by grouping all dimensions of our TF-IDF matrix by cluster. Our dataframe then consisted of a row for each cluster and a column for each Piazza post in the dataset. We choose the best words from the potential keywords by finding the words that appear in the posts with the highest TF-IDF score in the cluster.

We further split clusters into folders, taking advantage of Piazza's existing manual classification system, known to users as "folders" (ex. cluster 0 homework 1, cluster 0 homework 2...). We finally generate a text file with all of the clusters and keywords per cluster.

5 Results

5.1 Spellcheck

When building the dictionary for spellchecking, we set the cutoff for the number of times that a word had to be seen within the corpus in order for it to be considered correct to 5. We added all words that appeared more frequently to the dictionary, assuming that they were "correct" because of repeated appearance. 4.6% of all words were seen with a frequency of less than 5, and therefore classified as misspelled words. The spellcheck function takes in a word or multiple words, classifies each word as correct or incorrect, identifies the correct word for a misspelled word, and returns the correctly spelled word(s). This function was a key part of the preprocessing we performed on the dataset.

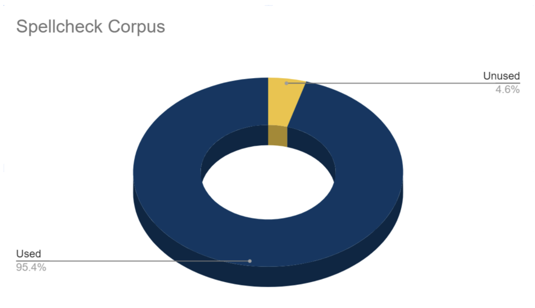


Figure 4: A pie chart showing the fraction of words that were classified as "misspelled" (in yellow)

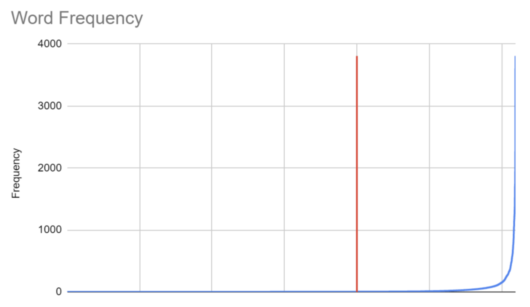


Figure 5: Plot of frequency of words. The vertical red line shows cutoff for fraction of “correct” words used in dictionary

5.2 FAQ

Creating the FAQ data generally worked well, with the clustering algorithm outputting several clear clusters that had good descriptions.

Figure 6 shows the sum of squared errors, with the elbow of the graph at number of clusters = 8, leading us to determine that 8 was the optimal number of clusters for the EECS 370 W22 Piazza dataset.

This clustering generated 8 groupings of keywords (Table 1). Figure 7 shows a scatter plot representation of the clusters, demonstrating the similarity between any two posts and providing a visual of the effectiveness of the clustering algorithm on this dataset. More broadly, this algorithm seems to have high success on online forum data.

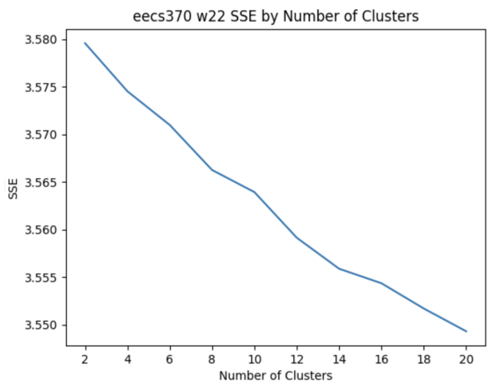


Figure 6: The sum of squared errors for each cluster count, with a significant elbow at 8 clusters

Table 1: K-Means clusters of keywords from EECS 370

#	**Keywords Extracted from W22 EECS 370 Piazza Data**	**Category**
0	"getting, simulator, run, project, spec, output, autograder, file, error, code"	AG + Project errors
1	"sure, look, autograder, code, ag, passing, failing, case, cases, test"	Test Cases
2	"miss, access, entry, tlb, level, memory, size, cache, table, page"	Cache Hits/Misses + Memory
3	"value, data, beq, register, add, instruction, lw, pc, stage, resolved"	LC2K Instructions
4	"wrong, failing, incorrect, tests, case, code, output, autograder, cases, test"	Test Cases
5	"wrong, tests, error, output, code, autograder, failing, case, cases, test"	Test Cases
6	"confused, nan, address, number, need, lecture, problem, use, bit, question"	Lecture Questions + Problems
7	"address, bits, symbol, relocation, set, size, memory, block, table, cache"	Memory + Addresses

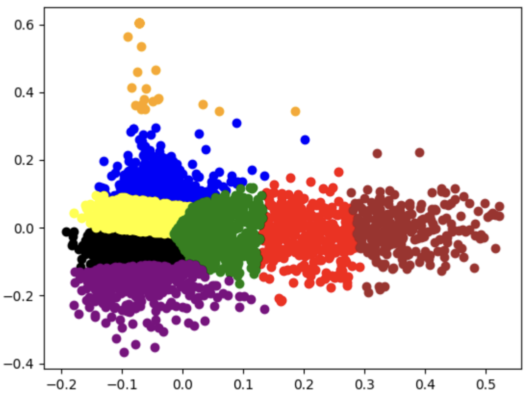


Figure 7: Scatter plot representation of the clusters, demonstrating similarity between any two posts

5.3 Precision and Recall

The maximum number of posts returned for any one query by any weighting scheme was just under 3000. Therefore, the first set of precision and recall metrics displayed in this graph are for the top 10 to 3000 on increments of 10 documents returned (10, 20, 30, 40, ... 3000). We plot the precision and recall values for each of the 6 weighting schemes: unigram `bxx bxx`, bigram `bxx bxx`, unigram `nxx bpx`, bigram `nxx bpx`, unigram `tfc tfx`, bigram `tfc tfx` in Figures 8 and 9.

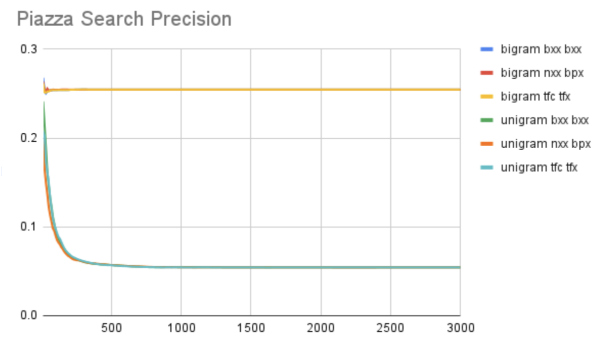


Figure 8: Precision for the top 3000 most relevant posts from W22 EECS 370 Piazza data

The bigram weighting schemes did not perform as well in recall, but had a higher precision than the unigram. The bigram generally returned much fewer documents than the unigram. Within the unigram and bigram classes, though, all of the weighting schemes performed similarly.

More interesting behavior can be observed in the graphs at the 300 mark and below. We rescale the graphs in Figures 10 and 11.

The difference in performance between each of

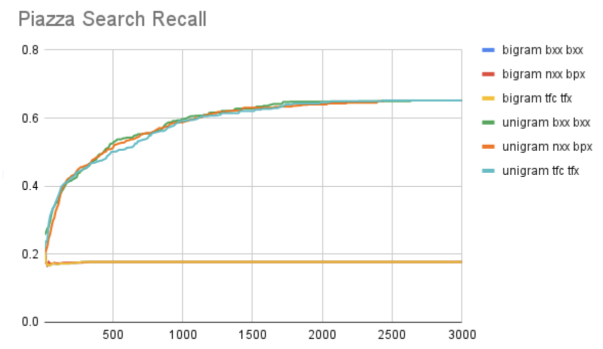


Figure 9: Recall for the top 3000 most relevant posts from W22 EECS 370 Piazza data

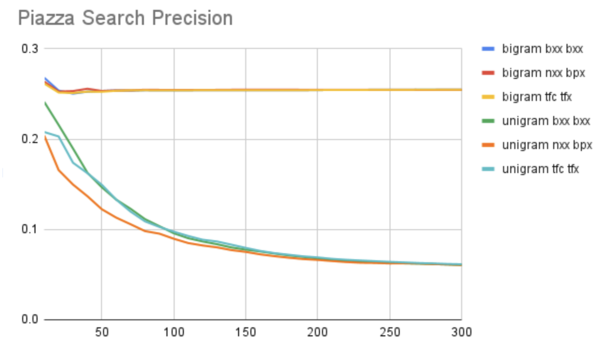


Figure 10: Precision for the top 300 most relevant posts from W22 EECS 370 Piazza data

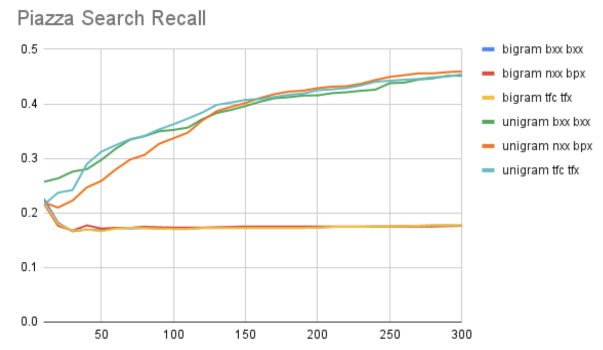


Figure 11: Recall for the top 300 most relevant posts from W22 EECS 370 Piazza data

the weighting schemes is more noticeable at this granularity. The `tfc tfx` weighting scheme for the query and the document appears to perform the best. We overlay these curves in Figure 12 for convenience.

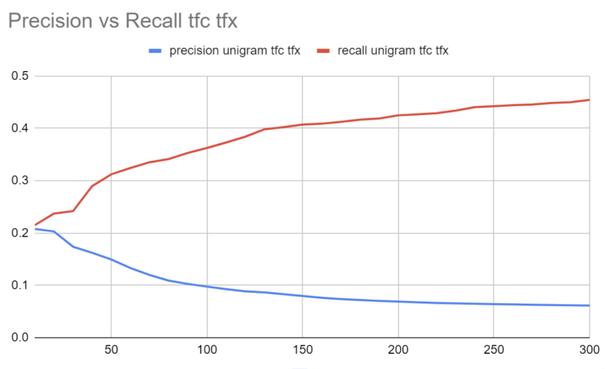
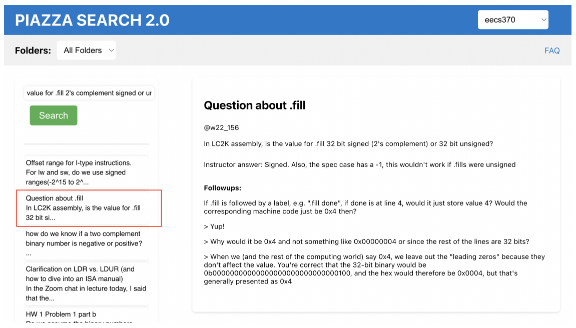


Figure 12: Precision and recall for the `tfc tfx` weighting scheme for the top 300 most relevant posts

The precision and recall curves diverge significantly after 30 documents returned. Therefore, we identified 30 as the optimal number of posts to return per query. It is worth noting that this number could change with each dataset.

5.4 User Application

We built a simple API and user interface for interacting with the improved search feature. The technical details for this web app are not relevant for information retrieval analysis, but we found it was easiest to test results with a user-friendly interface (see Figure 13).



value for .fill 2's complement signed or ur

Offset range for I-type instructions.
For lw and sw, do we use signed ranges(-2^15 to 2^...

Question about .fill
In LC2K assembly, is the value for .fill 32 bit si...

how do we know if a two complement binary number is negative or positive?
...

Clarification on LDR vs. LDUR (and how to dive into an ISA manual)
In the Zoom chat in lecture today, I said that the...

HW 1 Problem 1 part b

Question about .fill

@w22_156

In LC2K assembly, is the value for .fill 32 bit signed (2's complement) or 32 bit unsigned?

Instructor answer: Signed. Also, the spec case has a -1, this wouldn't work if .fills were unsigned

Followups:

If .fill is followed by a label, e.g. ".fill done", if done is at line 4, would it just store value 4? Would the corresponding machine code just be 0x4 then?

> Yup!

> Why would it be 0x4 and not something like 0x00000004 or since the rest of the lines are 32 bits?

> When we (and the rest of the computing world) say 0x4, we leave out the "leading zeros" because they don't affect the value. You're correct that the 32-bit binary would be 0b00000000000000000000000000000100, and the hex would therefore be 0x0004, but that's generally presented as 0x4

Figure 13: Piazza Search 2.0 UI

6 Conclusions

6.1 Main Contributions

This project makes it easier and more effective to search for relevant Piazza posts across multiple semesters.

6.2 What Worked Well

Scraping: We were very successful in getting all of the posts that we needed from Piazza. We were able to download data for all posts from a specific Piazza course and use it to generate an inverted index, correct spelling, and perform K-means clustering to group similar posts together. We were also able to remove links to other posts and replace them with the answer given in the other posts, enhancing the content base by including posts that do not directly have content due to linking.

Retrieving Relevant Posts: The chosen models were good at returning a lot of posts, with the most number of posts returned for a given query for any weighting scheme being just under 3000. This is very different behavior from the real Piazza, which only returns a post if all of the words in a query appear in the text. From the perspective of the user, it is better to have some posts that aren't perfectly accurate or precise and have the option to sort through the returned posts, than having none at all.

Spellcheck: Spellcheck was fun to implement and addresses the severe sensitivity to misspellings the current Piazza search functionality has. It was quite slow, taking up to 3 seconds to check each word, but when run on a small scale (such as only the search query), it worked well and provided a more pleasant user experience, even with a slight performance cost.

FAQ: The general idea of implementing the FAQ generation with K-means worked well. We successfully obtained clusters of posts (which we saw similarity in after manually scanning the results). We were able to obtain visual verification of successful clustering through scatterplots, and we saw that certain keywords from EECS 370 were properly extracted. Overall, we are pleased with the outcome, but recognize some of its flaws in the following section.

6.3 What Didn't Work Well

Runtime of Spellcheck: As mentioned previously, the runtime for the spellchecking was slow and not feasible to run on the entire corpus when building the dictionary, without access to more computational power. For our application, this could have been due to the data structure used for the dictionary (A built-in Python dictionary), or more likely, due to the search space being so large. Ways to improve this are discussed in the "Future Work" section.

Evaluating Document Relevance: Another aspect of this project that could be improved was evaluating the relevant documents for each of the test queries. Since one of the group members had more expertise on the EECS 370 content, it is possible that the results could be slightly biased and that the definition of a "relevant document" was different between group members. This may or may not have affected the precision and recall calculations. Additionally, the manual search for relevant documents was performed using the current Piazza search functionality, which we know to be unreliable in many cases.

FAQ Clusters: As seen in the results, some of the general categories across clusters are similar. This could be because the clusters were too granular or that there were too many clusters. If we were able to find a way to better predict keywords with more meaning and less generality, we could potentially have better clusters. Additionally, once we have the clusters, we would have to reconstruct the FAQ by looking through the posts that were classified in each cluster. Therefore, gathering information to find the FAQs for a dataset using solely clustering is not incredibly informative. The FAQ needs to be reconstructed using additional data after receiving the output from the clustering algorithm as a starting point.

6.4 Future Work

A very interesting area of future work would be improving the spellcheck runtime by limiting the search space of potentially correct spellings for each word. As mentioned above, the program takes up to 3 seconds to check each word. In the future, it would be good to speed this up either by using a faster data structure or by figuring out a way to limit the search space more, not requiring us to look through all possible words. This could be achieved by precomputing similarity scores between different words in the dictionary, then only searching the space with high similarity scores to that initial dictionary word. Additionally, the dictionary will be different for each class, so it would need to be rebuilt for each unique dataset. For example, the words "lw" and "sw" are commonly used within EECS 370 posts referring to assembly instructions. These words don't show up in any English dictionary, but we would not want to flag them as misspelled in this context. These exceptions must be accounted for.

We could also further refine the weighting schemes to measure the similarity between a query and a document. The inspiration to experiment with a bigram inverted index was the phrase "virtual memory" in the EECS 370 data. This query initially had a lot of false positives and very low precision and accuracy scores with the unigram inverted index, because both the words virtual and memory were seen often throughout posts independently, but the two together have a different, more specific meaning. For this query, a bigram inverted index worked better because it only returned posts that had "virtual" and "memory" together. In most other cases, however, bigrams were far too restrictive, returning only a fraction of the posts that unigrams were, resulting in poor recall and precision scores. Improvement would involve identifying certain words that should be bigrams, like "virtual memory," "crash consistency," or "control hazards," using these as full terms when computing relevance scores.

If launched as its own application, the UI could be built to replicate the current Piazza. For faster runtime, we could have a precomputed matrix of relevance scores for common words to each of the posts so that we don't need to search the inverted index to compute relevancy scores for every query request.

For the FAQ portion of the project, we could try different clustering algorithms to see if they perform better and produce better clusters. One new implementation that we briefly looked into was the DBSCAN clustering algorithm. This algorithm doesn't require an input of the number of clusters to split the data into like K-means does, but rather clusters based on distance. Starting from any unclustered point, the algorithm finds all points that are within a certain inputted distance ε of the current point, and classifies all these points together as one cluster. The algorithm then runs in a breadth-first search fashion, visiting each of the newly labeled points within the cluster, finding all points from the new point within distance ε, and adding those as well to the cluster. Once the search has visited all points within a cluster and there are no more other points within a distance ε, the algorithm marks this cluster as complete and jumps to the next unvisited point as the starting point for the next cluster. This method continues until all points have been sorted into a cluster. This implementation would save us the added complexity and runtime of trying K-means on different numbers of initial clusters, as well as the uncertainty associated with deciding on the optimal number for the given dataset. Instead, we would have the tunable parameter of the distance ε in the DBSCAN algorithm.

7 Individual Contributions

7.1 Mariam

Mariam worked on gathering the dataset from Piazza and helped with k-means clustering for FAQ generation. She solely worked on gathering data by calling the Piazza unofficial API, formatting all results into a JSON file to be easily used by the inverted index pipeline, and ensuring all edge cases were satisfied (posts with/without student/instructor answers, posts with/without follow ups, etc.). She also worked on the K-means clustering for FAQ generation (see relevant files). She wrote the K-means algorithm, helper functions to display clusters, and graphing to find optimal number of clusters based on sum of squared error and the elbow method.

7.2 Julia

Julia worked on evaluating the precision and recall of our system, preprocessing, and helped with the API. She came up with 50 test queries based on EECS 370 content, helped manually evaluate the relevance of all posts within EECS 370's W22 Piazza which allowed us to develop a relevance judgment file, and eventually helped to calculate the precision and recall of our system with respect to this particular dataset.

7.3 Chimmuanya

Chimmuanya worked on spell check and implementing the different weighting schemes to return the relevant documents given a query. She updated and resegmented code from our second assignment to work for multiple query and document weighting schemes. She also changed the code to build the inverted index using the results of the MapReduce pipeline. Chimmuanya also helped Sydney with testing and evaluating the results of spellcheck, and integrated it with the rest of the preprocessing steps.

7.4 Melina

Melina developed the inverted index pipeline and worked on the UI. She was familiar with the scalability of parallel programs in the context of search engines from previous course work and felt like taking the initiative to write a MapReduce pipeline to generate the inverted index as proof of concept, even though it technically did not run any faster locally. Melina was also familiar with frontend development and worked on creating a user interface similar to Piazza, but featuring the improved search feature, for the purposes of demonstration.

7.5 Sydney

Sydney worked on spell check, brainstorming the implementation of K-means, and evaluating precision and recall of the system. Regarding K-means, Sydney had worked on a project before that utilized the clustering algorithm, and had taken a class that further went into the specifics and applications of the algorithm. This background helped brainstorm the implementation and use case of the algorithm in this project. Additionally, Sydney helped Julia manually evaluate the relevance of all posts within EECS 370's W22 piazza which allowed us to develop a relevance judgment file, and eventually helped to calculate the precision and recall of our system with respect to this particular dataset.

All members participated in writing the project report and designing the project poster during collaborative work sessions.

References

Durham Abric, Oliver E Clark, Matthew Caminiti, Keheliya Gallaba, and Shane McIntosh. 2019. Can duplicate questions on stack overflow benefit the software development community? In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 230–234. IEEE.

Liang Cai, Haoye Wang, Bowen Xu, Qiao Huang, Xin Xia, David Lo, and Zhenchang Xing. 2019. Answerbot: an answer summary generation tool based on stack overflow. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1134–1138.

Jeffrey Dean and Sanjay Ghemawat. 2004. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA.

Damiano Distante, Alejandro Fernandez, Luigi Cerulo, and Aaron Visaggio. 2015. Enhancing online discussion forums with topic-driven content search and assisted posting. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management: 6th International Joint Conference, IC3K 2014, Rome, Italy, October 21-24, 2014, Revised Selected Papers 6*, pages 161–180. Springer.

Junaid Rashid, Syed Muhammad Adnan Shah, and Aun Irtaza. 2020. An efficient topic modeling approach for text mining and information retrieval through k-means clustering. *Mehran University Research Journal Of Engineering Technology*, 39(1):213–222.

Viljami Venekoski, Samir Puuska, and Jouko Vankka. 2016. Vector space representations of documents in classifying finnish social media texts. In *Information and Software Technologies: 22nd International Conference, ICIST 2016, Druskininkai, Lithuania, October 13-15, 2016, Proceedings 22*, pages 525–535. Springer.

Xiaojun Wan and Tianming Wang. 2016. Automatic labeling of topic models using text summaries. pages 2297–2305.