Project Amalgam

Githire Brian Wahome

Minerva Schools at KGI

Introduction

Amalgam Description

Amalgam is a study and analytics tool intended to standardize and help guide students as they go through study materials. Standardization in this regard implies having all study material being displayed on one page, using common fonts whenever possible, on a single tab. Amalgam combines all resources under one web page composed of collapsible widgets. Standardization is, in fact, the inspiration behind the name of the software, Amalgam, which means a mixture or a blend (Definition of AMALGAM. (n.d.)). Amalgam notebooks are a mixture of resources in one place organized in a coherent and easy to follow manner by their creators. The 'help guide' aspect manifests in the similarity statistics. In the context of Minerva Schools, at least, readings come with a study guide. This is usually a shortlist of the expected outcomes of the class and concepts one is expected to understand when doing pre-class readings. In a lot of classes, however, main readings tend to be long and dense, making them hard to follow even with a study guide. Experts suggest that on average, it takes at least three passes, with the third pass alone taking up to 6 hours to read and understand an average 28-page research paper (HowtoReadPaper.pdf. (n.d.)). This is not a feasible approach for most students as they usually have multiple such readings per class and multiple classes to read for. There is simply not enough time to read everything hence the popularity of skimming and the need for study guides prepared by tutors to assist students in narrowing down areas they should focus on[1]. Study guides, while helpful, tend to be too broad and general and in a lot of instances, poorly developed. In a lot of instances, they simply pinpoint the entire sections or chapters students should focus on. They rarely detail how to read supplementary readings which are sometimes more accessible for students or provide a hint on

[1] **#gapanalysis, #rightproblem:** We identify a gap where there is not enough time to read papers efficiently and there are no tools designed to help students skim or analyze the relevance of documents. After identifying this, we design solutions in light of this problem to address that gap.

where to read when a student wants to delve into more detail related to the recommended sections. In some instances, the recommended sections themselves are too long, and a page-level focus might be necessary.

That said, the relevance of said sections or pages can be estimated by various metrics with some being based on conceptual similarity[2] (Mirković, J., & Altmann, G. T. M. (2019)). Some others are more straightforward and based on diachronic content similarity (Bernecker, S. (Ed.). (2008)). It is harder to automate the inference of concepts from various resources and compare them. The same cannot be said for content, that is easy for computers to do. Amalgam takes advantage of this fact and analyzes the similarity of content within documents be they study guides, outcome descriptions, etc. The resources to which others will be compared are hereby referred to as base resources. Supplementary resources, which might be research papers, assignments among others, are hereby referred to as support resources. The core of it all is a simple similarity learning problem where page by page analytics are done by the software when resources are added and depending on their similarity to the base resources, the user can decide whether or not to prioritize them post-inspection. This is why Amalgam is being said to 'help guide' and not instruct the user. There needs to be a human to perform a sanity check on the generated statistics. More detail on this is in subsequent sections.

Some other problems Amalgam aims to solve include the elimination of distractions by eliminating the necessity of opening multiple tabs when doing readings. It also aims to minimize distraction that may come in the form of targeted advertisements and pop ups. Research shows that on average, college students (Indirectly inferred from the data as those aged 18-23) on average have 12 tabs open per browsing session

[2] **#CP194-Similarity:** Understand the differences between various similarity measures and what they tell you about the similarity of the resources being determined as well as how to interpret them.
#context, #CP194-Relevance: Similarity may indicate relevance but it does not imply relevance though in a lot of instances it will. Similarly, relevance does not necessarily imply similarity. It is key to distinguish this. This is similar to correlation and causation. One does not imply the other regardless, they tend to follow similar patterns. Relevance is very conceptual and context-dependent which makes it a very hard measure to track but, were to come up with improvements to Amalgam's relevance analysis, a good place to start would be similarity analysis. We also very extensively discuss the role of context in keeping meaning and why our preprocessing loses it. It is paramount to practically everything we discuss.

(How many tabs do people use? (Now with real data!)). This number, of course, goes higher when they are doing research, say for an assignment. With more than 70% of websites tracking user activity, this poses not only a security threat but leaves people vulnerable to aggressive marketing and advertising campaigns which can be distractions[3] to the education process. The old saying goes, 'education is a shy bride, be present or she will let you be'[4]. It is necessary that attention is maintained when studying. By having these resources compiled in a similar web structure with offline access enabled, Amalgam minimizes the need to load up multiple tabs. The layout also follows a waterfall structure which is similar to digital books with some sections being collapsible. More on the logic behind the design is discussed in later sections.

[3] **#utility**: We define the ability of Amalgam to make resources available offline, speed up the retrieval of resources and even the skimming process, thus defining Its utility relative to the base case where no guide is given.
#scienceoflearning: We are designing Amalgam to minimize distraction, reduce targeted and assist in skimming, all techniques that are designed to facilitate more efficient learning based on the science of learning.
[4] **#analogies:** We use multiple analogies to define functionalities within Amalgam from the wolf references for scores to the bride example to illustrate how one needs to focus to learn.

Inspiration

The structure of Amalgam was initially intended to resemble that of Jupyter notebooks (Project Jupyter | Home. (n.d.)). Some design features both on the frontend and the backend are, in fact, directly inspired by Jupyter. The viability of Jupyter notebooks as study tools and not just analytics tools/ Integrated Development Environments was demonstrated to me and fellow students in Professor Scheffler's class on Modelling, Simulations and Decision Making (CS166). While hosting technical exercises in notebooks was not an idea he pioneered, he perfected it by combining technical guides and reading summaries. These were seamlessly integrated into a single notebook such that students did not need to leave the document when doing technical pre-class work as well as most of the required readings. This sure took a lot of work but the resultant document flowed well and allowed full engagement with the students both during preparation for classes and during the classes themselves.

Jupyter notebooks excel when it comes to technical exercises requiring python programming but it does not work well for other classes. In fact, a lot of non-Computer Science students absolutely loathe the software. This inspired the Amalgam team of one, myself, to create an alternative that simplifies content creation, carrying over the convenience of Jupyter in combining study resources into one document. Amalgam also aimed to eliminate the technicalities of creating notes in Jupyter using markup languages.. Amalgam initially aimed at facilitating the creation of text resources but evolved to include analytics tools like neural networks and similarity algorithms that not only analyze added resources but also help the reader have an idea of the similarity layout within documents and study guides. This is discussed in more detail in the later parts of the document.

Some of the bits that made Jupyter notebooks and Google Colab (Google Colaboratory. (n.d.).) (a cloud version of notebooks, so successful) great were adapted in the implementation of Amalgam, for example, the ability to share notebooks as JSON files, user locking to limit unauthorized editing, the ability to view notebooks natively through any browser and the availability of the software as a downloadable package for use offline.[5]

Similarity Learning Problem setup

Outside lab work and field studies, most studying, revision and academic research is usually set up nicely as a supervised learning problem (Supervised Learning—An overview | ScienceDirect Topics. (n.d.)). Students and professors alike are usually short on time and tend not to be able to thoroughly read all the material available within assignments, text books, research articles among others. This is why quick search mechanisms are a must have for most modern day document readers and browsers. Most of these search mechanisms, however, only allow for the exact matching of words and short phrase querying. Amalgam automates the process of painstakingly searching all instances of words by doing bulk comparisons between document and text objects uploaded into the system.

Classification is key in order to help tell the software how the comparisons should be happening. The system is set up such that there are two classes (resource types), base materials, and support materials. Base and support resources are of similar types structurally and content-type wise. They can be videos, audio, text and images. The system can currently feature all of these within notebooks but the analysis tool is only able to perform similarity analysis on non-video/audio resources. Back to the similarity problem aspect, the system evaluates the similarity of support materials to base materials. Several similarity metrics are

[5] **#ConceptVsStatement:** The key to gauging the usefulness of Amalgam's analysis is understanding the kind of topic being analyzed. It is bound to fail when gauging things with an extremely wide specifications domain like Professionalism but it will shine in domains with a small and standard specification domain. By specification domain like probability and complex systems, I mean the set of keywords, phrases, and words used to describe them. In the latter case, similarity tends to be a good measure hence why Amalgam would be useful in such cases. A good starting point to determine the usefulness is the scores. Jaccard is a strict measure thus high scores will only be seen in very similar resources. It can thus be used to gain a sense of the usefulness while Cos score with its wildness can be used to sniff out the similar sections.

evaluated and made available to the user. Example guides on what the scores might mean and recommended interpretations are provided both in this document and in the demo videos for HCs and LOs (Minerva-HCs-Intro.pdf. (n.d.)).

This base-support problem framework especially works well in the context of Minerva as study guides can be used to gauge the relevance of long research papers by page. Similarly, HC and LO descriptions can be used to gauge which page of assignments and long research papers which may be provided as class readings, a reader should prioritize reading or tag. This makes Amalgam a potential complementary assessment tool for use by professors as well as for students who might forget to tag HCs/LOs they have applied inadvertently. A good example of this is seen in the HC demo video where I failed to see viability of #multiplecauses. This was a complex systems HC and I was checking for it in paper that was more relevant for Empirical Analysis. Amalgam, however, had other ideas and showed a spike in similarity statistics and upon inspection, it was indeed true that #multipelecauses was very relevant for the paper on Acid mine drainage on page 3. Kindly view the demo for this change of heart. There are of course some caveats to this, especially since similarity is not the only way to gauge the effectiveness of applications of HCs and LOs or the relevance of documents but it's a good enough start when trying to pinpoint potential areas one should study. It surely is why we Ctrl-F every time.

Potential use cases

HCS/LO application detection

This was the use case that shaped Amalgam the most. Habits of mind and foundational Concepts(HCs) at Minerva, in particular, come with a description and an example application. These example applications are usually accessible from the outcome index. While Minerva stratifies HCs by cornerstone course or blocks of habits the school tries to instill through those HCs, Amalgam choses a different path. HCs can generally be stratified into two types, conceptual HCs, and descriptive HCs although it is safer to think of this as a spectrum since some HCs can arguably be classified into both categories. Regardless, there is usually a tendency towards one or the other. It should be noted though that this division is purely for functional reasons and when selecting HCs to analyze for, one should follow the expected outcomes index as defined by professors and tutors and use Amalgam to complement these applications.

As said, this binary division is not strict. One way to infer which category a HC or LO would fall into is gauging how human-like a computer has to be in order to tell how a HC or LO is applied. A computer may find hard to infer #context, #professionalism, and #connotation for instance. These tend to be the arts and multimodal communication HCs and LOs (Learning Outcomes). There are however some HCS whose applications will tend to reuse the same buzzwords. These are usually within the hard sciences i.e Empirical sciences and Computational Science. Complex System is surprisingly the best class to apply Amalgam since it has a very standard framework that all complex systems are defined against. This means the words used for all these tend to be similar. Example HCs include #probability, #descriptivestats, #agents among others. One can judge the suitability of Amalgam when deciding whether to test it for an HC application based on an estimate of the size of the minimum set of vocabulary a person will use to describe most applications of the HC or LO. The likes of probability will reuse the same statistical buzzwords like ‘distribution’, ‘dies’, and ‘chance’ while something like #connotation and #denotation can be described for literally all words in the dictionary depending on #context of course.

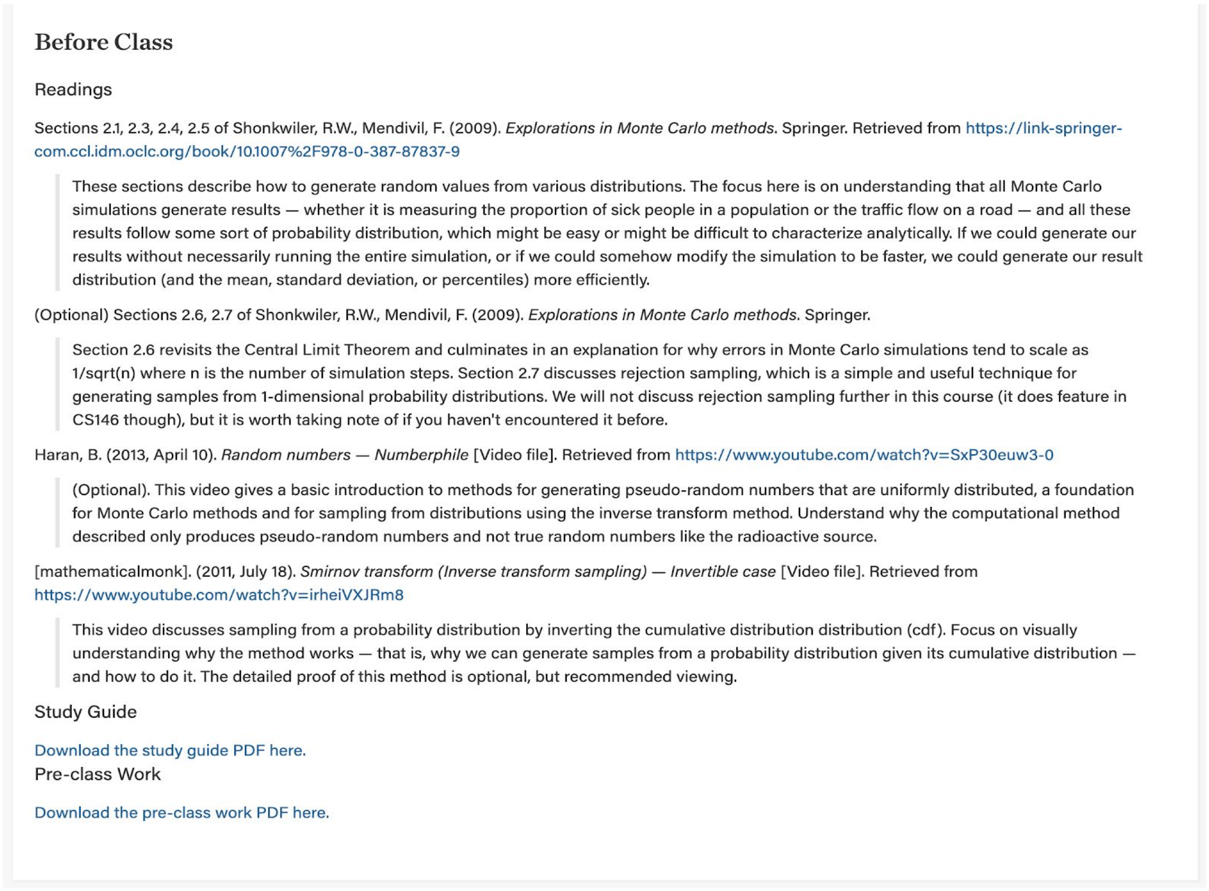
Amalgam thrives when inferring when descriptive HCs and LOs have been applied. It works best for HCs and LOs that are more targeted and structured in their definition and application. With these, the cosine score, normally used as a plagiarism metric, is a good indicator of where to search or tag. If one or two words match in a chunk of 500 words, the cosine score will rise a fair amount, prompting the grader of an assignment to pay attention to the page/text that caused the spike in the score. There is usually a high likelihood that this section will have the HC or LO applied and is a good tagging candidate. It should be noted that in case the HC or LO is tagged as footnotes, it is inevitable that that score will be a bit higher due to the intentional spotlight the student sheds. Thus Amalgam should not be used to inspect for these after tagging is already done in the case of assignments. In the case of tagging as appendices, a professor may use Amalgam to validate the student statement about their own HC and LO application. In case there is a pattern that matches how a student stated they tagged their HCs, then all that will be left will be validating the application by reading their justification and the paragraphs around in-text footnote anchors. Sometimes students may tag HCs or LOs outside the page they applied them. Amalgam can help nail down the exact location they may have applied the targeted outcomes even before the professor checks the validation the student gives in the footnotes and should they match and be argued for correctly, a good score may be awarded. A better use case, however, would be, in case a student has not tagged the HC or applied a HC but did not tag it, Amalgam could hint at areas the professor may grant the score if the HC or LO was applied. More scores, happier students. Students could also maintain a notebook containing HCs as base materials and when done, they could run Amalgam analytics to see if they have any HCs they could have missed and may be able to make a strong case for if the similarity analysis validates it. Amalgam could thus serve as a standard tool that both the professors and students can use to infer the application of HCs and LOs and would even encourage students to make use of concept vocabulary to maximize similarity scores.

Narrowing down readings from study guides and summaries

Within Minerva, study materials are assigned to students along with a short study guide or description. The amount of detail within this depends on the professor or peer tutor preparing the summaries. They range from simple lists of pages to read, outlines of sections that need to be covered during class preparation or short descriptions of the expected outcome accompanying the links to study resources. Class discussions are based on these recommended readings. In light of this, these resources should serve as base resources within Amalgam notebooks. A base resource can be compiled by uploading study guides. Other base resources can be created by pasting the relatively short reading descriptions and expected outcomes into the Amalgam editor. Once these are added to the system, they can then be analyzed individually, or collectively using Amalgam's inbuilt 'All Resources' option, available in the dropdown of the resource selector. On the backend, this combines all the base resources, text generated from the stemmed image classifications, text extracted from the PDFs among others among others. These can then be compared to any resources added to the system as support resources.

For support resources, just like base resources, Minerva usually provides a link to a web page, a pdf or a video (Or a collection of videos thereof (play lists)). Amalgam makes use of an open-source blog editor, quill.js (API. (n.d.), Quill), that is designed to render content on a browser, making the importation of online material a matter of copying and pasting. The distortion of the format is minimized by the editor's inbuilt style processors which have been refined for over 10 years. Quill is a very widely used WYSIWYG editor meaning most changes to it are incremental improvements, feature introductions and bug fixes. This ensures backward compatibility. It is very well maintained and designed. This makes sure that any resource created with Amalgam will retain it's format even as styling, framework and language standards change. The resources attached will always look as they were created. The benefits of this are that a user can simply copy and paste web pages in whole into the editor and save the resource and Amalgam will retrieve the data added without needing to refer to the original source. The analytics process remains the same. We discuss more detail on how the specific resource types are analyzed. Images, for instance, have a neural network on the backend to classify them, videos are currently featured but not analyzed although that endpoint is currently left pending for future development while text resources are fully analyzed and chunked appropriately for the data visualization. Pdfs are chunked by page before processing.

The dedicated view which includes an analytics dashboard is available and functional for support resources..For base resources, the dedicated view simply displays the resources on their own. Support resources are what Amalgam attempts to optimize the study of, thus their dedicated view comes with charts that showcase page by page similarity statistics using various scores. The exact resources to be compared are up to the user. Readings can be compared to each other, descriptions to readings, or as stated, a combination of all base resources to a specific support resource. This is where this application of Amalgam excels as not only does it compact the reading materials into one, unified, stand-alone resource that maintains the format of the source or the format the creator used, it also provides some suggestive similarity analysis between the base material and support materials to complement the purpose of study guides in narrowing down readings. It may even give insight into sections that were not yet recommended that would be worth a look.



Before Class

Readings

Sections 2.1, 2.3, 2.4, 2.5 of Shonkwiler, R.W., Mendivil, F. (2009). *Explorations in Monte Carlo methods*. Springer. Retrieved from https://link-springer-com.ccl.idm.oclc.org/book/10.1007%2F978-0-387-87837-9

> These sections describe how to generate random values from various distributions. The focus here is on understanding that all Monte Carlo simulations generate results — whether it is measuring the proportion of sick people in a population or the traffic flow on a road — and all these results follow some sort of probability distribution, which might be easy or might be difficult to characterize analytically. If we could generate our results without necessarily running the entire simulation, or if we could somehow modify the simulation to be faster, we could generate our result distribution (and the mean, standard deviation, or percentiles) more efficiently.

(Optional) Sections 2.6, 2.7 of Shonkwiler, R.W., Mendivil, F. (2009). *Explorations in Monte Carlo methods*. Springer.

> Section 2.6 revisits the Central Limit Theorem and culminates in an explanation for why errors in Monte Carlo simulations tend to scale as 1/sqrt(n) where n is the number of simulation steps. Section 2.7 discusses rejection sampling, which is a simple and useful technique for generating samples from 1-dimensional probability distributions. We will not discuss rejection sampling further in this course (it does feature in CS146 though), but it is worth taking note of if you haven't encountered it before.

Haran, B. (2013, April 10). *Random numbers — Numberphile* [Video file]. Retrieved from https://www.youtube.com/watch?v=SxP30euw3-0

> (Optional). This video gives a basic introduction to methods for generating pseudo-random numbers that are uniformly distributed, a foundation for Monte Carlo methods and for sampling from distributions using the inverse transform method. Understand why the computational method described only produces pseudo-random numbers and not true random numbers like the radioactive source.

[mathematicalmonk]. (2011, July 18). *Smirnov transform (Inverse transform sampling) — Invertible case* [Video file]. Retrieved from https://www.youtube.com/watch?v=irheiVXJRm8

> This video discusses sampling from a probability distribution by inverting the cumulative distribution distribution (cdf). Focus on visually understanding why the method works — that is, why we can generate samples from a probability distribution given its cumulative distribution — and how to do it. The detailed proof of this method is optional, but recommended viewing.

Study Guide

Download the study guide PDF here.

Pre-class Work

Download the pre-class work PDF here.

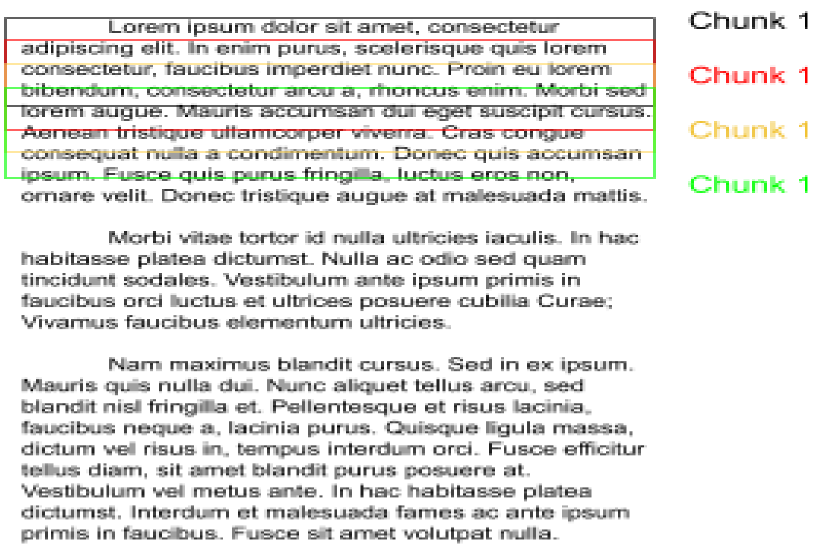
An example of the list of readings, a study guide, pre-class work and reading descriptions. Note how some of the readings are optional and do not go into detail on the sections to read. Even for the main reading, the description is short and the guide is simply a highlight of entire sections or chapters which can be up to 5 pages each meaning the whole resource can span upto 60 pages long incontent. This is for just one session. Amalgam refines the reading even further by analyzing them page by page. Mind you this is one of the very well done study guides. Some classes will not have this much detail in their study guides as the amount of guidance provided is up to the professor and their teaching assistants and not all of them provide similarly detailed support resources.[6]

[6] **#multipleagents:** The design of Amalgam and the way the scores are laid out are inspired by the projected requirements of the target **#audience.** The system is meant to appeal to all these agents and the design has been inspired by the kind of work that different professors have presented in the guides and requirements ranging from technical notebooks to basic study guides.

Use case deduction, generalization and other examples

As has been highlighted in the above use cases, Amalgam performs analytics on problems which can be structured following the base-support framework. This allows us to think of them as similarity learning problems. Objects need to be classified by the notebook creator as either base resources or support resources. The dedicated view will show the analytics for the support content as these are the ones that tend to be longer and more detailed as well as the focus of the user. Amalgam helps guide the skimming process for these. Given this setup, we will now discuss some general use cases for Amalgam and justify them. These can be thought of as potential use cases. They are all within the academic realm.

Amalgam can be scaled to operate as a plagiarism checker by academic practitioners. It would be particularly useful when students are coworking or working on similar assignments. A professor may choose to upload all gradable assignments as both base and support resources then compare them using the cosine score and Jaccard score to fish out potential plagiarism cases. The cosine score, in particular, is a ‘cry wolf’ score and will sway towards the higher values when there is any bit of similarity. While it is not always the case that high scores are plagiarism cases, they could be viewed as potential cases being flagged. A final review step by the professor, teaching assistants or some other proper authority will be necessary in order to determine if there is indeed a case to argue. This could also be done by comparing the base study resource to uploaded assignments in case students copy solutions from a teacher’s guide word for word. This is an instance where Jaccard’s score may thrive. Students tend to cheat using marking schemes available online. They copy solutions especially for mathematical problems and having a system that can flag these instances out would be beneficial. A modification that can help with this is changing the chunking length for resources to the length of the base resource rather than a fixed length of 500. Allowing for overlap chunking so that the system looks at all possible contiguous blocks of 500 words would be necessary to ensure this check is thorough. This, however, is extremely computationally intensive and could be best done using a simple regular expression matching based search. Regardless, it is something Amalgam is capable of performing



Lorem ipsum dolor sit amet, consectetur adipiscing elit. In enim purus, scelerisque quis lorem consectetur, faucibus imperdiet nunc. Proin eu lorem bibendum, consectetur arcu a, rhoncus enim. Morbi sed lorem augue. Mauris accumsan dui eget suscipit cursus. Aenean tristique ullamcorper viverra. Cras congue consequat nulla a condimentum. Donec quis accumsan ipsum. Fusce quis purus fringilla, luctus eros non, ornare velit. Donec tristique augue at malesuada mattis.

Morbi vitae tortor id nulla ultricies iaculis. In hac habitasse platea dictumst. Nulla ac odio sed quam tincidunt sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Vivamus faucibus elementum ultricies.

Nam maximus blandit cursus. Sed in ex ipsum. Mauris quis nulla dui. Nunc aliquet tellus arcu, sed blandit nisl fringilla et. Pellentesque et risus lacinia, faucibus neque a, lacinia purus. Quisque ligula massa, dictum vel risus in, tempus interdum orci. Fusce efficitur tellus diam, sit amet blandit purus posuere at. Vestibulum vel metus ante. In hac habitasse platea dictumst. Interdum et malesuada fames ac ante ipsum primis in faucibus. Fusce sit amet volutpat nulla.

An example of how overlapping chunks work with linewise splits. Word-by word splits would also do something similar with a chunk starting from the next word of the one Its predecessor started from. The number of chunks would be equal to the number of words thus the system would perform the analytics process n times where n is the number of words in a text.

Another potential application is the use of Amalgam as a supplementary grading utility for assignments. We have seen how Amalgam is able to point out potential HC/LO application points especially when the said HCs and LOs reuse similar words and jargon. For professors, especially when dealing with the application of concepts from another school, this can come in handy. Imagine a situation where a student applies fermi estimates or probabilistic analysis or even deduction in an art assignment, Amalgam could help bridge the gap between schools by pointing out this application thus an art professor, who may not have been focussed on grading these LOs and HCs, may be prompted to inquire and potentially grade the student on the applications. This expands the scope of concept application across schools and does not limit their use to the class they are taught. Having this system in place unifies schools and makes the Habits of Mind and Foundation Concepts, not just academic tidbits needed for a good grade in the class they are taught, but tools one can apply everywhere. [7]

Similarity Learning

Introduction

Similarity learning is a form of supervised machine learning. The goal of it is to compare the similarity of two objects (Similarity Learning, Ch3.pdf. (n.d.)). There are several types of similarity learning but the most popular one, and the one we focus on, is distance metric learning. Distance metric learning entails fitting a distance function to multiple objects that are being compared to each other. In this regard, the distance is the difference between the objects in comparison. Similar objects will have little to no distance separating them while different objects will have huge distances separating them. Distance in our case will be mathematically defined be it through trigonometric relationships or the differences counts.

Our paper focuses on two main distance metrics, which we interpret as similarity metrics, Jaccard similarity, and Cosine similarity though more may be added. We take these similarity scores as indicators of relevance. They are also the easiest indicators to compute since they make use of similar values such as the

[7] **#levelsofanalysis:** We have analyzed Amalgam's **#utility** for **#multipleagents** in a school level. We have discussed how it might help professors, how it might help students, and even how it might unify the two by standardizing the format of assignment writing. In doing so , we discussed how the system can even **#nudge** students into using technical vocabulary as it might make the system spike, making professors note their applications more. This agent level analysis which encompsses what they stand to gain as well as encouraging them in some directions justifies the apploication of these HCs.

size of the union/intersection of our compared objects. In the case of Amalgam, we compare base and support resources, which form the objects of our comparison functions. We standardize the text to eliminate any variation due to grammar rules through a process called stemming. We also remove stop words since these may distort our similarity measurements. After standardization, we then perform a frequency count of the words for optimum processing. The union and intersection operations necessary for the two distance metrics are very computationally intensive thus saving these values for reuse is going to save us precious computation time.[8] Similarity is then evaluated between key vectors as generated from the texts and then Amalgam uses the lower of the frequencies for common words to determine the size of both intersections and unions. This speeds up the comparison operation as we do not need to recompute these statistics, rather, we just plug them into the jaccard and cosine formulas and generate our values. The comparisons are conducted for each page of the support resources in case they are pdfs. Should they be created within Amalgam, i.e using the blog editor,the input text is split into sections of 500 words each, 500 being the assumed length of a single page of text.

Cosine similarity

We have stated that the most paramount measures of similarity in Amalgam are the Cosine and Jaccard similarity scores. The Cosine score, in particular, is the most frequently used. The intuition behind it is explained and illustrated below:Assume we have 2 texts,

S1: Julie loves me more than Linda loves me, **S2:** Jane likes me more than Julie loves me

[8] **#CS110-dynprog:** Saving of generated set properties such as the union and intersection for reuse.

To estimate how similar these texts are, purely in terms of word counts (and ignoring word order), we find the union of the two sets of words from the two texts, then have a frequency count vector for each of the text. The union set is as follows:

[me, Julie, loves, Linda, than, more, likes, Jane]

The frequency of words from each of the texts can be vectorized to generate the following n-dimensional vectors where n is the length of the union.

words	S1-Vector	S2-Vector
me	2	2
Jane	0	1
Julie	1	1
Linda	1	0
Likes	0	1
Loves	2	1
more	1	1
than	1	1

The Cosine score ignores the order of the words. It also focuses on the difference between the vectors whose dimensions are set by the individual words from the union, The vector magnitude is equivalent to the number of words (sklearn.metrics.pairwise.cosine_similarity). The cosine score is essentially an estimate of the inner product space which is the angle between the two vectors.

The two vectors are labeled S1 and S2:

S1: [2, 0, 1, 1, 0, 2, 1, 1]

S2: [2, 1, 1, 0, 1, 1, 1, 1]

The cosine of the angle between them is about 0.822. To visualize this logic, let us do the operation again. Assume we have these simple 2-word example,

S1: “I am”

S2: “I will”,

The union set is :

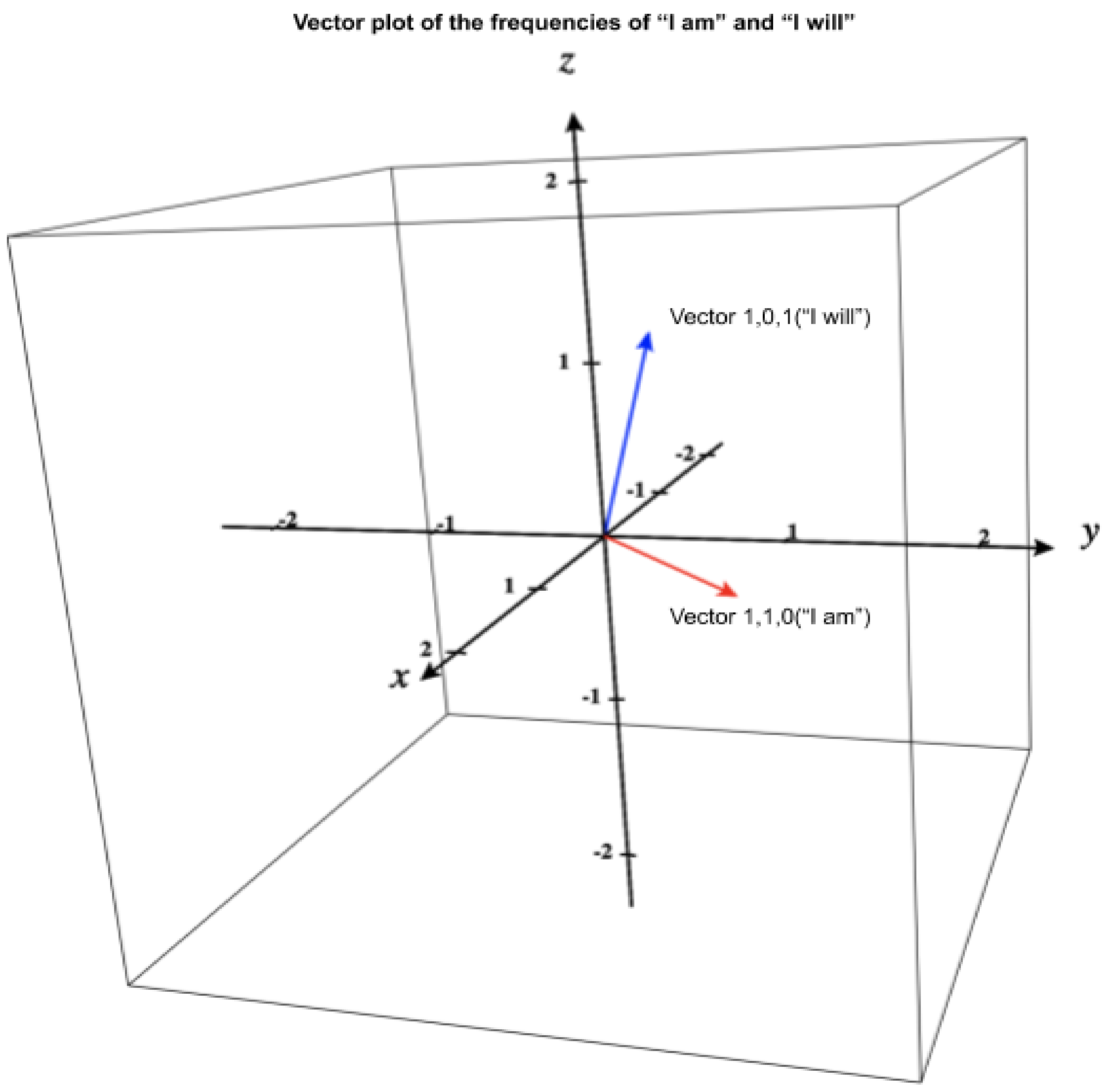
[I, am, will].

The vectors from these are

S1: [1,1, 0]

S2: [1,0,1]

Visually speaking, this is a plot of the vectors in Euclidean space.



9

Observe the large angle between the red and blue vectors. The cosine does not care about the frequency of the words, rather, it only needs there to be an intersection for the overlap to exist and create a huge space between the vectors. The angle between these two is 60 degrees thus a cosine similarity score of

[9] **#dataviz:** I think I will let the paper speak for itself on these, all complex topics are broken down to corresponding visualizations that illustrate what they are all about. See for instance the best of the cases, the 3D visualization for the cosine score about, and many more!

0.5. It would stay the same for “I am, I am” and even “Am I am I”. Repetition does not affect the angle between the vectors, only the magnitude. This is why the cosine score is a good plagiarism checker. It stays high should there be any similarity but as we see here. It does, however, ignore the order of the words leaving it prone to loss of context defined meaning be it through the use of tenses, prepositions, emphasis through repetition etc. This is why human inspection is necessary.

Jaccard’s similarity

Jaccard’s similarity is a much simpler score. Simply put, it is the proportion of text that is in the intersection of the two texts being compared to the total amount of text(Size of the union) (Jaccard Similarity—An overview | ScienceDirect Topics. (n.d.)). Using the same example we had:

S1: Julie loves me more than Linda loves me

and

S2: Jane likes me more than Julie loves me

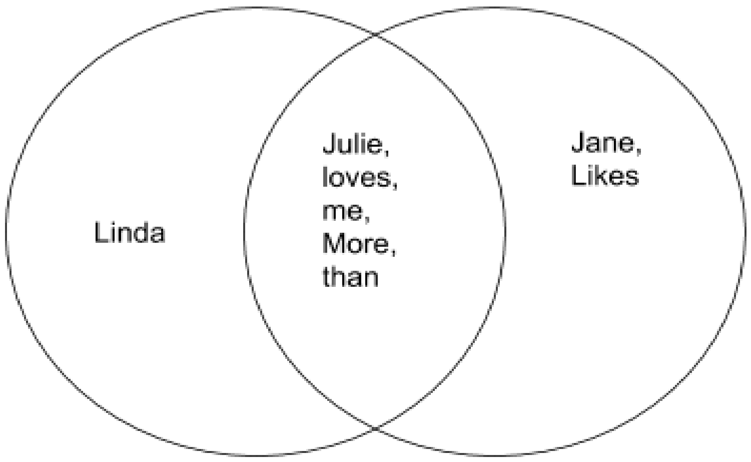
We do the same thing and find the union of the two sets to obtain the following:

[me, Julie, loves, Linda, than, more, likes, Jane]

From this, we then obtain the intersection of the two texts. We can do this simply using the same frequency table. An element can be considered to be in the intersection of the minimum of the frequency of the elements from both is at least 1. For the 8-dimensional vector, we simply need to count the elements from each dimension that are not 0 in either of the sentence vectors. The Jaccard score will be the length of this intersection divided by the length of the union of the two above. For our examples above, the bolded entries are the intersection, and our union is of length 1:

words	S1-Vector	S2-Vector
me	2	2
Jane	0	1
Julie	1	1
Linda	1	0
Likes	0	1
Loves	2	1
more	1	1
than	1	1

The Jaccard score for this is thus about 0.625. This is fairly close to the Cosine score. Jaccard analysis tends to be strict as it takes into account the length of the union too. This makes it poor as a plagiarism checker since, in the grand scheme of the document, a single sentence or two being copied will be insignificant. The union set is too big relative to the intersection. The score becomes even less significant in the case of books and research articles. The score will, however, excel when comparing resources of similar length. We optimize it’s implementation using the frequency tables and the counter scheme. Visually speaking, the Jaccard score can be represented for our example above using a Venn Diagram as follows:



S1:*Julie loves me more than Linda loves me*
Set length: 6

S2:*Jane likes me more than Julie loves me*
Set length: 7

Union = 8
Intersection = 5
Jaccard = ⅝

Jaccard Score vs Cosine score

Jaccard and cosine similarity are two of the most popular similarity measures. We have illustrated why the cosine score is wilder with slight similarities causing huge jumps in the score and pointed out why this makes the cosine score ideal for plagiarism analysis since, in such instances, caution is more beneficial than surety. The cosine score is a 'cry wolf' score.[10] With cosine similarity, the number of common attributes is divided by the total number of possible stemmed words whereas in Jaccard Similarity, the number of common stemmed words is divided by the number of stemmed words that exist in at least one of the two texts being compared (Zahrotun, L. (2016)). In most instances, the set of all words is smaller than the total number of words, making the cosine score more often larger as it has a smaller denominator. We also highlighted why the Jaccard similarity metric is strict. A single similar word on it's own does not affect the overall score significantly. This strictness stems from the fact that it looks at the intersection compared to the total collection of words in the documents. A single word in such an instance is not significant relative to the length of the whole document. We discuss more on the interpretation of Jaccard in light of the text document later but analogically speaking, Jaccard is the kind of score that only shouts when the wolf has already 'bitten off a hand', 'biting off a hand' here meaning lazy copying and pasting.

In Natural Language Processing applications and web development, Jaccard's score is commonly used to test for mirroring (321.pdf. (n.d.)). It is commonly used by web crawlers when scanning for mirrors of illegal entertainment websites and piracy websites which usually. These usually have multiple mirrors of themselves as they get taken down often by law enforcement. The reason Jaccard works so well when identifying mirror websites is that it will only show high scores when there is an almost 1:1 similarity

[10] **#analogies:** I use analogies alongside corresponding illustrations to explain the scores often. Cosine score and the cry wolf is one example. **#biasmitigation:**Amalgam is inherently a bias mitigation and coercion utility. Students might avoid some readings because they are too long, too technical among others. The system very objectively presents all page statistics for the user in order to have them make data driven decisions which might **#nudge** them objectively to study even things they may be uncomfortable with.

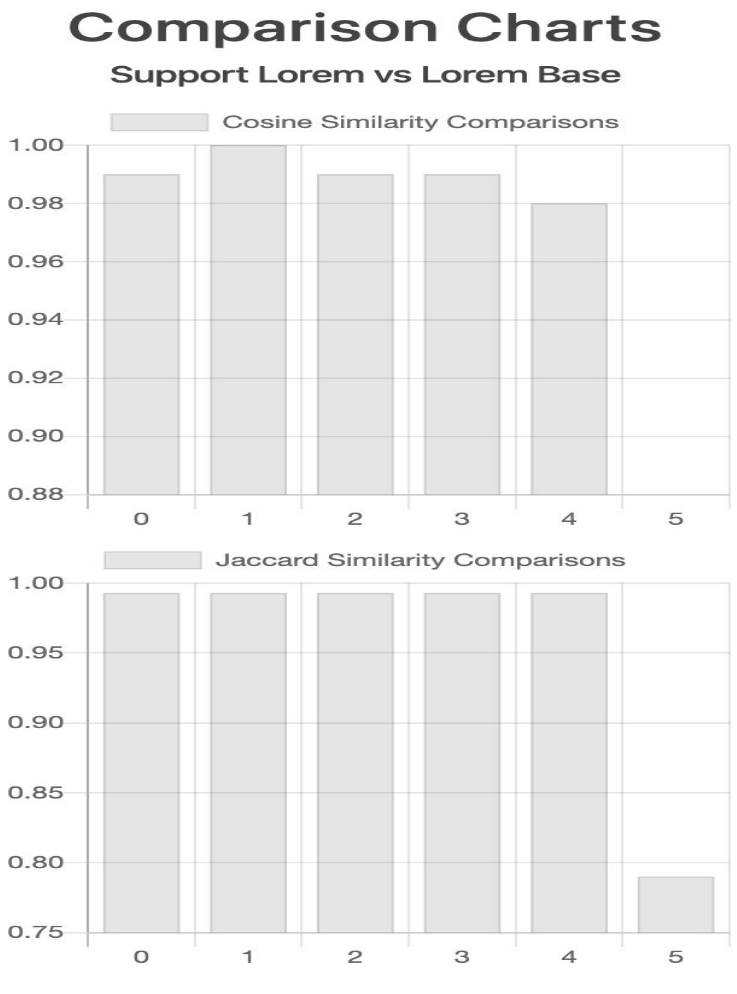
between the objects being compared. The thing with websites is that they are all built on tags which are common among almost all of them and thus, there will always be a lot of similarity in the markup text that composes any two websites even if they are not structural replicas of each other A strict score is thus necessary to detect mirrors as structural similarity is a necessity for a website to qualify as a mirror. Movie and music titles stay the same in both the legal and illegal websites. The score used must thus ensure the website being scanned resembles the reference website structurally for it to be flagged and Jaccard thrives in such instances.

The cosine score, on the other hand, fails spectacularly as a mirror content detector. This is because it jumps about widely should any similarity be detected. A single similar word or tag changes the difference angle and subsequently, the trigonometric measure substantially. This is not ideal, especially when scanning large documents. A lot of words tend to be reused. We try to mitigate this by chunking our data into sections of 500 or analyzing pdf documents page by page but this is not foolproof. Cosine similarity, however, thrives in cases where slight similarities are expected or plagiarism is suspected as it will point out any bit of similarity between documents, resources, text, and objects. This wild nature of the score thus makes it suitable when comparing short texts, making it the default score in Amalgam. A lot of Amalgam users will seek to fish out any similarity between short base and support resources. It is recommended that the cosine score be used in collaboration with other scores to ensure that its predictions are validated. Atop that, we cannot stress further that the scores should be thought of as hints and human validation is absolutely necessary. They will cast a spotlight on areas worth looking for discrepancies in or worth focussing on as but they do not assure that whatever they pointed out is relevant.

Amalgam Jaccard and Cosine similarity usage and interpretation

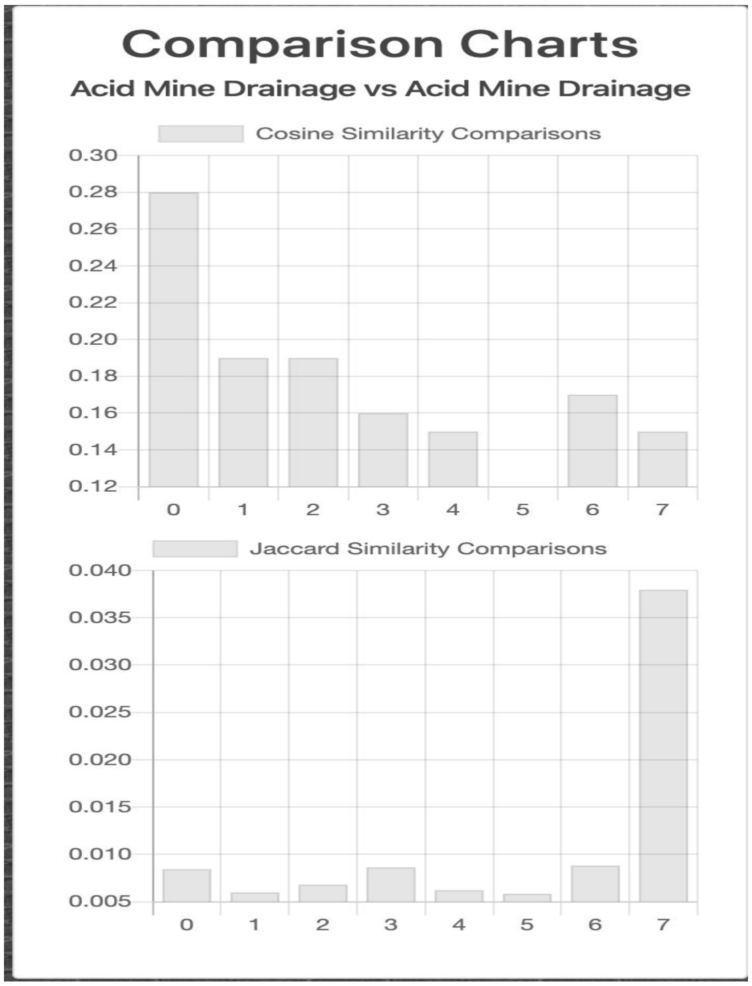
Amalgam evaluates multiple similarity statistics i.e Jaccard score, cosine score, euclidean distance (Euclidean vs. Cosine Distance. (n.d.)).As stated, Jaccard and Cosine are the two main ones. We discuss in this section how to interpret these using some charts generated by Amalgam.

Take this screenshot for example that shows the Jaccard score and cosine score from some random Lorem Ipsum text:



There clearly is a pattern given the two scores agree all too well. The base, as well as the support resources, are all random Lorem Ipsum generated text thus there is a lot of similarity between the two texts. One can be very sure that such a document would most likely be a mirror of sorts of the base given all the scores are well above 0.80.

But what if we introduced another file that is completely unrelated to the topic of discussion? Say we upload a paper on Acid Mine Drainage, we should expect little similarity between this and the random Lorem Ipsum text. The analysis Amalgam gives us is as follows.

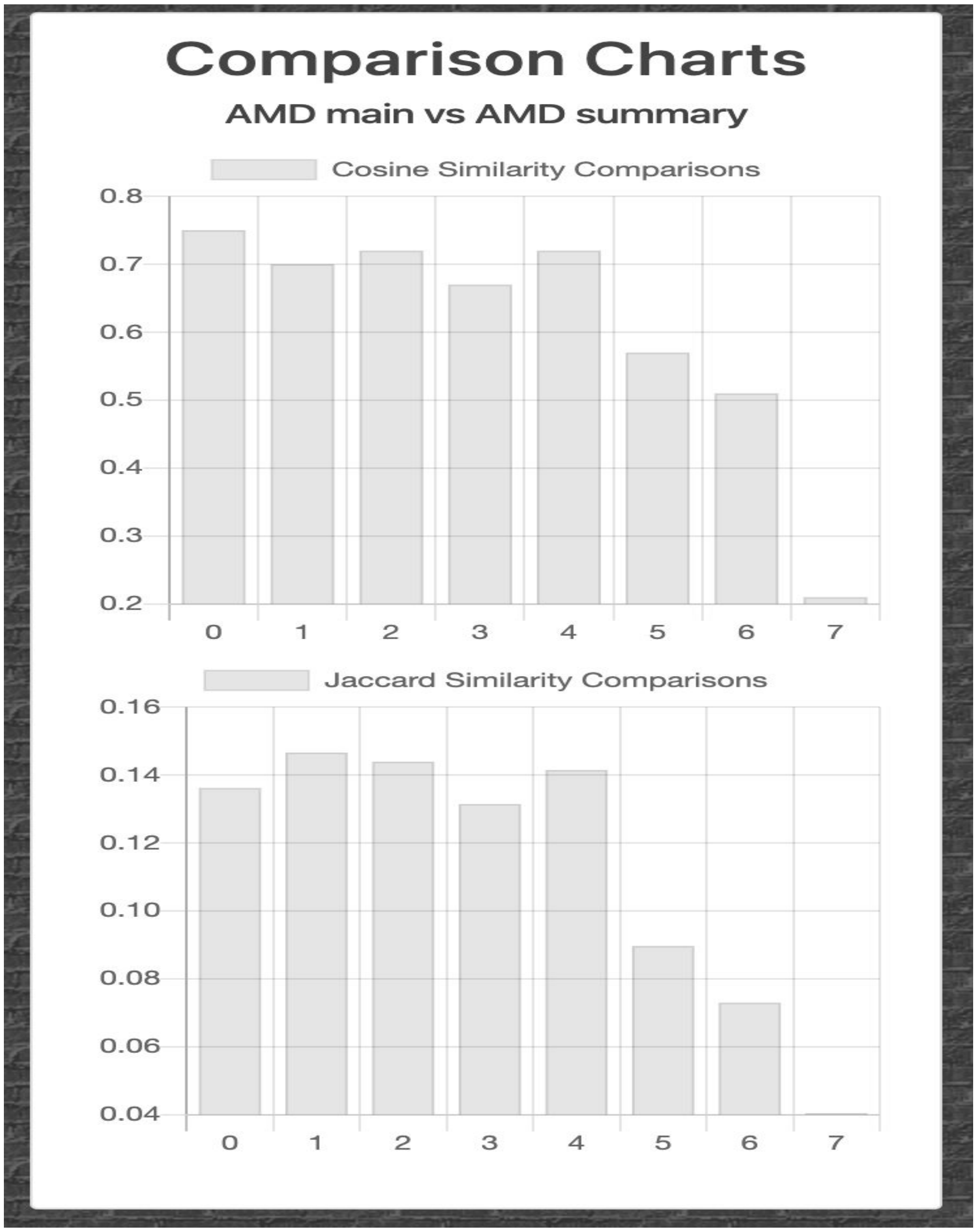


Notice how low the scores are in both. This alone indicates to us that there is likely no significant similarity between the two. Atop that, there is a lack of coherence between the Jaccard's score and the Cosine score[11]. The variation is a result of the fact that the cosine score indicates to us the total similarity in light of the total number of possible attributes whereas Jaccard is a score indicating the number of common attributes is divided by the number of attributes that exist in at least one of the two objects. For the Jaccard score, assuming no similarity and normal document structure where the end page is often left empty or with hugely spaced references, this will most often showcase the highest score since it will have a smaller union size than the rest of the pages. In this regard, the denominator is smallest given the small union size thus a higher score. Regardless, we should expect low scores altogether and the lack thereof of a common pattern between both scores. It should be noted, however, that the cosine score remains fairly wild and considerably high for some pages since there exists a bit of similarity. This is why it should always be cross-examined against other scores to determine if a similarity pattern exists as we do for the examples above.[12]

[11] **#multiplecauses:** There are various patterns we can expect from documents and we point this out. We even highlight examples with different scores and describe why these are as they are like the last page having scores as high as some similar content pages despite being empty. It is necessary to be aware that scores can have varying causes for their values and how to interpret them..

[12] **#evidencebased:** The effectiveness of Amalgam is not just stated, we have shown both in the demos and in the charts within this document that the system performs its analytics in its current, first version form.

Now, what if we compared documents of fairly similar topics? Say a summary document on acid mine drainage and a longer research document on the same topic.



While the scores are not as high as the Lorem Ipsum generated text, it is clear that there is a pattern in the text. Page 2 and 4 (to be interpreted as 3 and 5 given 0 indexing)), in particular, perform well from the in both scores. This is the expected kind of output we will get when the notebook is about a particular topic and we have a sensible summary. The exact examples that generate these charts are used in the demo videos and their links are provided. It is wise to toggle through the thresholds Amalgam offers the user and observe if there exists a pattern and if there exists one, what are the scores given by our metrics. If the scores seem to agree, then the user may follow the pattern of similarity as an indicator of relevance to optimize their study of the longer support resources or in case they are tagging/marking HCs and LOs, check for them in the spikes.

Implementation Analysis

Jaccard Score

Assuming we have a support resource of length S, i.e it has S-words, the Jaccard step of the program will be performed on chunks of 500 of any resources being compared. Say we are comparing the S-length support resource to a base resource of length B. Afterwards, what Amalgam does is compare the whole length of B to chunks of 500 words in the S length support resource. It is assumed the user will always have quick summaries and short descriptions for bases and longer support resources thus no page wise split for bases has been done. With this in mind, in the next section, we analyze the performance of the Jaccard implementation for a single chunk (page) of support resources with a base of length B.

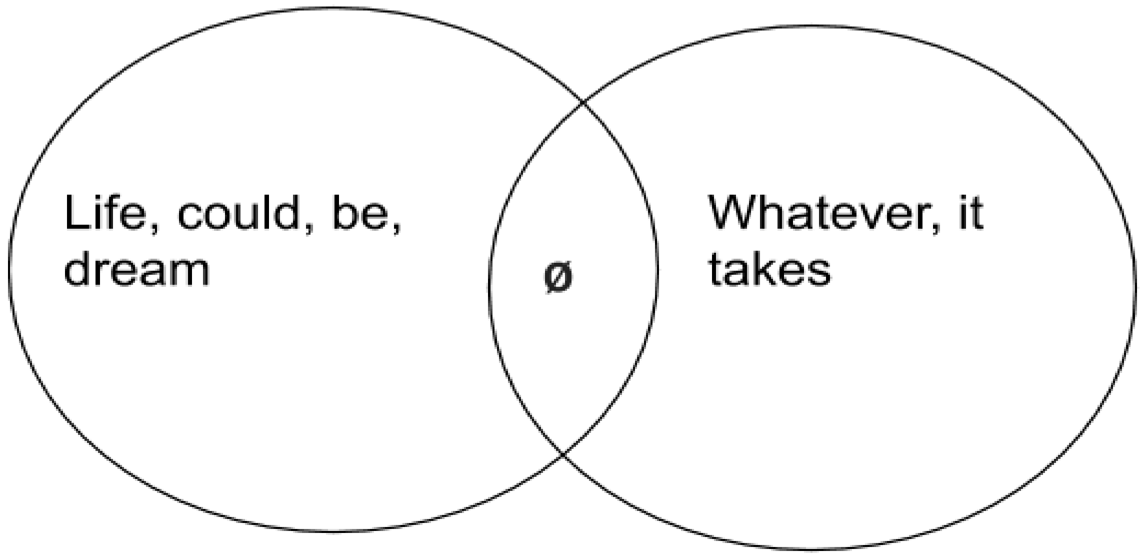
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

The Jaccard Score as is implemented first generates a frequency count for the words in both text objects. The worst-case scenario in this regard would be a situation where all words in both the base and support are unique. This means we end up with a frequency object of length B for the base and length 500 max for the support. Our worst-case memory complexity as such is O(B+S). With this frequency table generated, we then find the length of the intersection of the two text objects which entails finding the number of words with a frequency count of at least 1 for both. In the case of our unique words object, this would be 0 but we would have to run through the whole frequency count object at least once for a runtime of O(B+S). The Jaccard computation is a simple mathematical operation with these values generated. The complexity for a single page calculation is thus:

O(B+S) extra memory and O(B+S) runtime. [13]

[13] **#CS110-complexity:** Analyzed the performance of Amalgam from a complexity analysis perspective.
#CS110-novelapplication: Used machine learning and various algorithms and data structures to tackle a unique problem.

Worst case analysis example



Union set	S1 vector	S2 vector
{	<	<
Life,	1,	0,
Could,	1,	0,
Be,	1,	0,
Dream,	1,	0,
Whatever,	0,	1,
It,	0,	1,
takes	0	1
}	>	>

Intersection: **ø,** length = 0
Union = len(B)+len(S) = 7
Jaccard = 0/7 = 0

14

It should be noted, the computations that are generated are reused especially since the same S1 vector that we generate for the Jaccard computation is the same that will be parsed into our Cosine Score calculator. This makes the system more efficient as we avoid any unnecessary set operations and recomputations. It should, however, also be mentioned that whole document traversals are done at one point especially when doing the stemming of words and stop word removal hence the optimization is more or less a micro optimization and it does not affect the overall complexity. At the moment, these steps are conducted separately and they all have linear complexities since the stop word removal, for instance, is a lookup in a set which is a constant time operation. As such, to look up and stem all S or B words in a support or Base operation, it would take no more time than S*t where t is the time needed for one word lookup. This is still less than the O(B+S) time complexity for the union and intersection processing and is still linear. As stated, given the fact that the minimum possible amount of work we can do still needs a scan of the whole document, most optimizations performed are micro-optimizations.

Cosine Score

The cosine score as we have repeatedly stated is our cry wolf score. It too requires the compilation of a union set just like the Jaccard Score so let us define the problem in a similar light. Given a support resource of length S, the cosine score computation the program performs is on chunks of 500 of any support resources being compared. When comparing the S-length support resource to a base resource of length B, similar to the Jaccard step, Amalgam compares the whole length of base resource B to chunks of 500 words in the S length support resource.

[14] **#optimization:** We discuss how we sacrifice some memory to speed up processing and analyze the performance of various functions in this regard.

We then compute the union set which in the worst-case scenario will be the union of two text resources where all words are unique in each. This will make use of O(B+S) extra memory and will run in O(B+S) time. The Union computation too will be an O(B+S) runtime process as it will need to run through the whole frequency table to generate our B+S dimensional vectors. These vectors are the ones for which we will compute the cosine score. The cosine score will be computed as follows:

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

This is essentially the dot product of our B+S dimensional vectors multiplied by the magnitude of their lengths. In our case, this will be:

$$Score\ complexity = O\left(\frac{(B+S).(B+S)}{(B+S)^2}\right)$$

For any two vectors a and b, the dot product is computed as follows:

a.b = a1 * b1 + a2 * b2 + ... + an * bn[15]

[15] **#estimation:** All the complexity measures are done based on estimates of the overall performance and generalizations on memory usage. The estimates are informed by computational expertise and generally agreed complexities for mini operations like addition.

Assuming multiplication and addition are constant time operations, then the dot product is a linear-time operation O(n) and we only need to keep track of product sum as we compute thus it needs only O(1) extra memory. The final runtime thus has a worst case complexity of O(B+S) operations for both the numerator dot product and the denominator product of the magnitude of the vectors. The end result is that both the cosine score and Jaccard score run in O(B+S) time, linearly defined with respect to the total length of the base and support resources with similar space complexity of O(B+S, for the n dimensional frequency vectors).

Supported Resources

User entry: Raw text

Amalgam makes use of a quill editor for entry of all resources other than pdfs. The editor provides simple styling controls for including justification of text, bolding, italicization among others. Given that the editor was designed to render entries as viewed during creation within, it comes as no surprise that the data is exported as HTML. This makes it really easy to work with and save as once the data is saved, it is retrieved from the editor element. For raw text, these are usually header, paragraph and link tags. We make use of Beautiful Soup (Beautiful Soup Documentation), a tool designed for content extraction from markup languages and used mainly for web scraping. Amalgam repurposes it and uses it locally for entry scrapping.

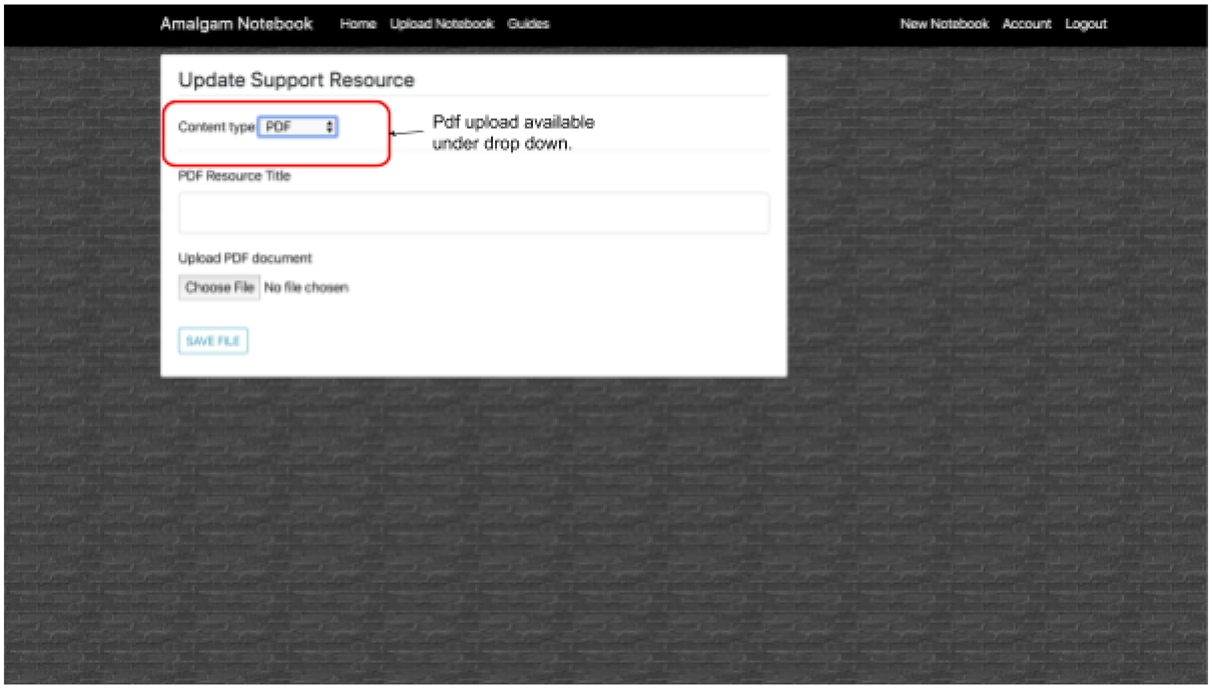
Once beautiful Soup extracts the text, we proceed to chunk the text into sections of 500 words as we have discussed in the past sections. The chunk size is constrained by the length of entry as we expect some of the resources added such as HC descriptions won't be a whole page long. We thus set the chunk size to the minimum of the length of the text and 500. This way, we can easily work with resources that are shorter than 500 pages long the same way we do with longer ones. After chunking, the generated object is parsed through a similar page by page analysis as pdf documents, only this time, the pages are chunks of 500 words. The preprocessing steps are the same, stop words are removed, then stemming is done before the similarity analysis is conducted on the final output.

The beauty of saving raw text is that the user is in full control of how the text looks like. The editor automatically generates the necessary tags needed to display entered text and the style scripts needed to display it as they desired. One thing to note is that this text can have images embedded into it as well as videos. This makes it easy to retain the flow of information that the user may be trying to communicate using various media although the different resource types are processed differently as discussed in their dedicated sections. Currently, the ability to export compiled text as PDF is being explored though it is not a feature that will be present in the submitted capstone version.

User entry: PDF files

PDFs are the most popular standard of sharing documents. This is because they retain the format of creation which makes them secure and hard to manipulate. It is for this reason that most reading materials, be it books, research papers, articles amongst others are presented as PDF documents. The key to their structure, just like other markup files, is the use of tags. We will not get into detail with how these work but simply put, tags are used to define the architecture of a document the same way they are used to define the structure of web pages (PDF File Format: Basic Structure. (2018, May 6)). In this regard, we can scrap PDF documents of their content using specialized software like Beautiful Soup.

Before discussing how Amalgam handles PDFs, let us first discuss how they are saved. Unlike images, videos and raw text which we save in the database, we save pdf documents to memory once uploaded. This is why they are the only ones that are uploaded separately in Amalgam. The reason for this is to make sure that not only we retain the original structure of the document but to avoid redundant computational work by extracting content from the document only to feature it again in our viewer. We sacrifice the editability that we acquire in converting the pdf to html. Should the user desire the ability to edit their PDF, they can always copy the content and paste it into the editor.



Once uploaded, we make use of a python pdf utility called pdfminer (Shinyama, Y. (n.d.). pdfminer) which is fed the URL of the saved resource. It then loads this up to memory and extracts the text, saving it into an object whose keys are the pages from where the text was extracted. The text in this object is later preprocessed. The preprocessing step is undergone by all text as previously stated entails the removal of stop words whose role in most sentences is the providence of context. It is hard for machines to infer such conceptual subtleties thus filtering for such words all together rarely affects the performance of text analysis. The filtration of these stop words is done using the Natural Language Toolkit (NLTK) package (Natural Language Toolkit—NLTK 3.4.5 documentation. (n.d.).) which contains a set of stopwords in which we can search when scanning the documents. After the filtration of stop words, Amalgam then stems from the words left. Stemming entails the removal of tense vowels, removal of quantifiers among others to produce standard words. These are not necessarily correct English words in the context of English but for purposes of Machine Learning, they work better for analysis since they tend to convey a similar meaning. An example, say we want to preprocess the sentence:

“I have never been to the top of the world.”

We first remove the stop words: “I, have, to, the, of”

We are left with:

“never, been, top, world”

After stemming and tokenization, we have:

“never, be, top, world”

Note how ‘been’ becomes ‘be’. This is because it is a variant of ‘be’ i.e “be, being, been...”

The ‘be’ is the common part of all, thus it is the stemmed version of all such words. The stemmed words are then parsed to the analytics tool which conducts the similarity analysis on them, saving the statistics generated data into a python dictionary that will be processed appropriately for visualization.

User entry: images

Amalgam analytics at its core can simply be described as a text comparator. There needs to be some form of text representation available for the system to compare to all forms of resources. With images, we have to find a way to determine what they contain in order to appropriately classify them. In comes image classification to save the day!

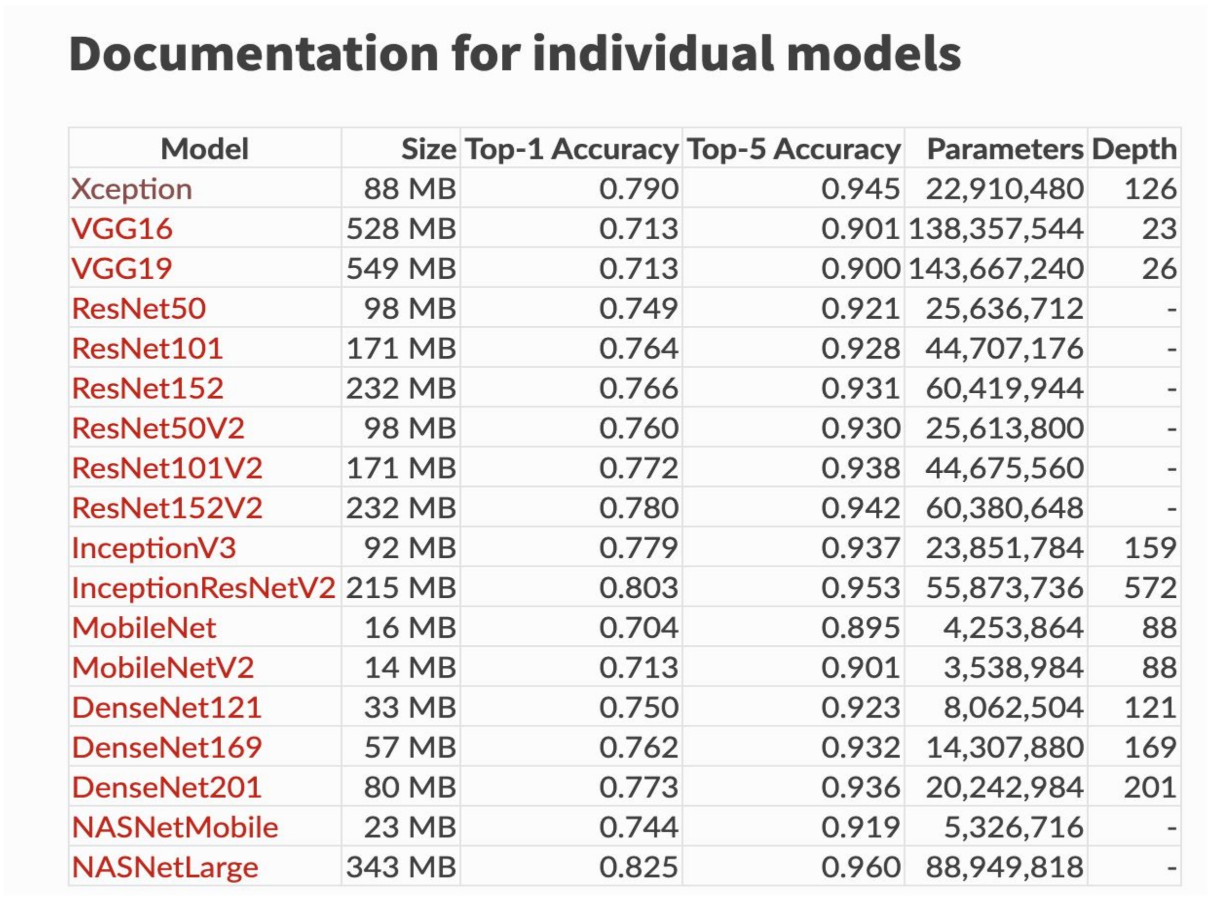
Amalgam makes use of a combination of neural networks, one which is a variant of the Xeption V8 model on Keras (Applications—Keras Documentation. (n.d.-a))[16] while the other is a cloud option using Imagga APIs. At the moment, Amalgam makes use of simple image classification. In the future, feature extraction such as text extraction might be supported but given limited resources and the fact that most of the content being analyzed are not images, focusing on perfect image processing was not a priority. Regardless, we discuss in detail how the classification of images and the output of said classifications play a role in the analytics.

For the local neural network, an implementation of the necessary architecture to load up the pre-trained weights for a high-performance image classification convolutional neural network was done. The python model replicated the structure of Xeptions hence why it could load and make use of its

[16] **#CS156-deeplearning, #CS156-feedforward-nets:** Choose an appropriate neural network model, implement the necessary architecture needed to load the weights for it, wrote functions to modify input to match Its expected inputs and interpret the outcomes correctly for use within the system. I also highlight some of the strengths of the model and weaknesses eg for classification vs feature extraction such as text reading.

weights(Applications—Keras Documentation, Xeption. (n.d.-b)). This model was chosen due to its relatively small size and high performance. It has 22,910,48 parameters and on ImageNet and has a top-1 validation accuracy of 0.790. It also has a top-5 validation accuracy of 0.945. The ease of selection of models also made the Keras applications easy to use thus in the future, the user might be granted the ability to make use of multiple neural networks or even select the model they desire to use. Images were standardized to the 299*299 pixel size Xeption requires for its input then classified. A threshold is also set to ensure that only classifications that the network is at least 30% confident are accurate are taken into account by the analyzer.
[17]

[17] **#CS156-classification, #CS156-neuralnetworks:** Detailed the problem set up as a supervised learning problem. I acknowledged the use of Image net for training the neural networks although in order to have my own supervised learning problem, I also did the similarity analysis, a form of similarity learning which is a supervised learning problem. Followed the necessary preprocessing steps and justified them and then did the necessary evaluations to showcase document similarity.

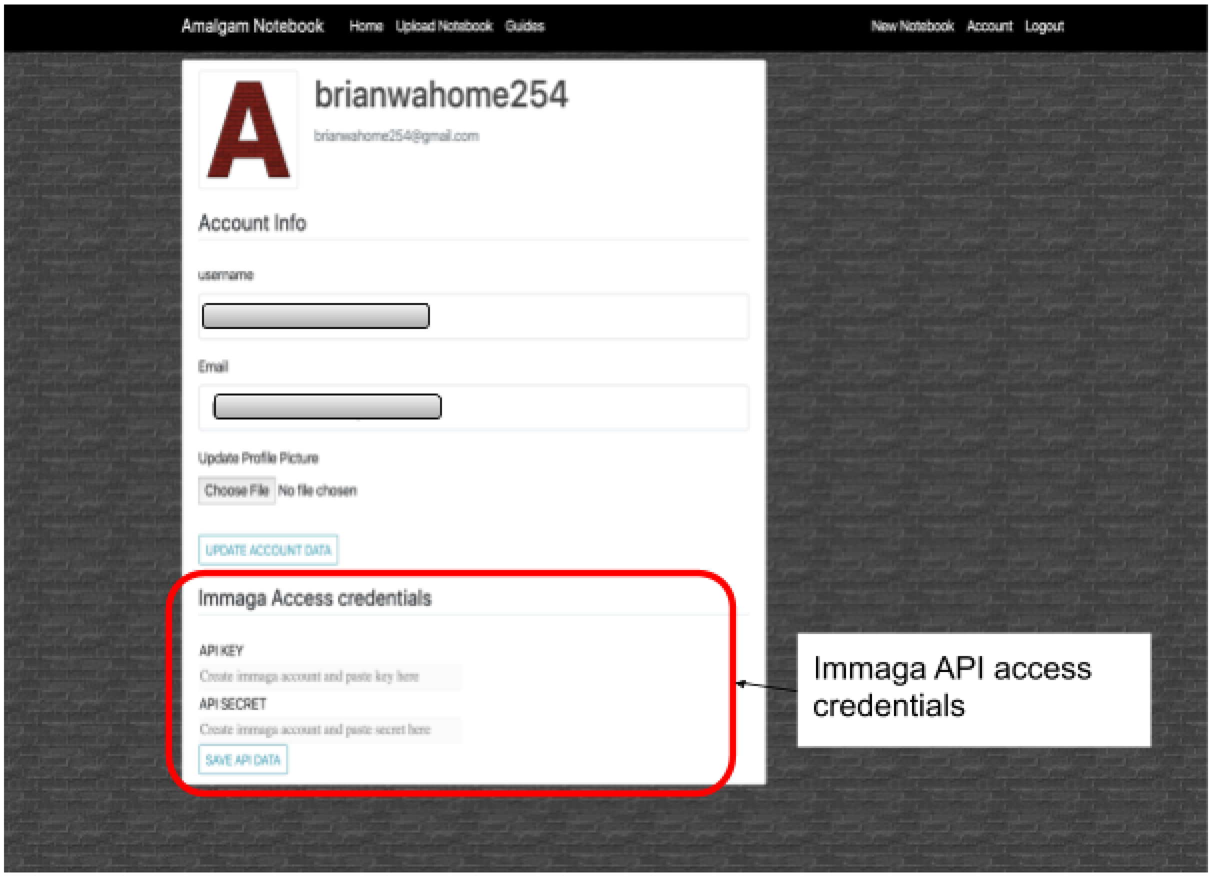


Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

Retrieved from: Applications—Keras Documentation, Xeption. (n.d.-b)

For the cloud option, Amalgam makes use of a robust and developer-friendly API provided by the Imagga project (Image Recognition API, Computer Vision AI – Imagga. (n.d.-a)). While the service is available for commercial purposes, they offer a fairly robust and ‘’casual developer’ friendly free option to classify 2000 images a month. As such, Amalgam provided input fields for the user to add their own API keys and secrets as generated from Immaga to be used to access the API services. The instructions on how to add these to the system as well as how to create the keys are provided within the guides as well as the Input prompts for the keys. This commercial neural network is more robust, provides more accurate classifications, and provides endpoints that may be used for text extraction let alone simple classifications. These will be explored more in the future but at the moment, a simple classification endpoint is called for images included as part of Amalgam resources. Similar to the local neural network, once the classes are sent back to the system by Imagga, they are filtered by a set threshold and the qualifying classes are appended as part of the base/support text for preprocessing and subsequent similarity analysis.



For image uploads, to standardize the format of uploads, Amalgam encourages users to either embed these directly into the editor as this also allows the user to style them by uploading the images to the system. Pasting may not always work due to variation in image types. Uploading ensures that all image encoding is done by Amalgam. On the back end, the editor exports images as encoded base64 strings which are very easy to send as part of query strings to the Imagga APIs. Prior to encoding, they are standardized and first processed by the local neural network as PNGs. The encoding also allows for the saving of the images as part of the notebook and resources sqlalchemy models in lieu of saving images as separate files[18]. Strings are as a rule of thumb easy to work with. The native conversion of images to base64and vice versa facilitated by the editor makes image handling very simple as we can easily save the base64 code as strings. The editor comes with an inbuilt base64 processor thus when images are parsed into it for display during the retrieval of notebooks, no special decoding process is required other than parsing the string into the editor as "...| safe". Flask then decodes it into HTML. This abstracts the complexities of saving multiple versions of the images uploaded for display, processing and saving and allows the developer to focus on classifying the image and the user on styling the image display as they desire. This same editor is used by LinkedIn and a variant of the same has been used by Facebook and even google for its docs project (The Story Behind LinkedIn's New Publishing Platform. (n.d.).). It is powerful and well-backed thus we can rest confident that it will be maintained well into the future by good developers to accommodate legacy projects of the future such as Amalgam.

User entry: Videos

Videos are in their simplest form a series of time-spaced image frames designed to give the illusion of motion. In this regard, being able to analyze images automatically means we should be able to analyze videos. A caveat to this though is the fact that we rarely are sure of the video section we should be analyzing and running algorithms to infer the meaning behind videos if an extremely daunting computational task. As

[18] **#CS162-sql:** Normalized and implemented appropriate queries for the databases in order to save our current data.

such, Amalgam opted not to analyze any videos embedded. It does, however, offer all the necessary tools needed to embed and play videos from any URI within the editor and content views.

We expect that most videos will be played from youtube. While it is possible to have iframes that we can feature the links through, these are not ideal as they do not provide a means through which we can embed said videos directly into our text. A lot of times videos come with descriptive text, captions, titles among other text information that may give us a hint about what the video entails. Amalgam thus provides video embedding options,which were facilitated by special Quill video modules (Building A Custom Module. (n.d.)). The functions provided by the module are seamlessly integrated within the editor. We rely upon the user to simply upload raw videos if the content is self-created to Youtube or any other free video hosting service and then we make use of the video embedding feature that we activated as an option within our instance of quill editor to feature these.

Note: At the moment, exploration is being done to automate the extraction of a sample of 10 frames equally spaced across any video and then performing classification on this text in a similar way to how images are analyzed. The option to also retrieve youtube subtitles from the URL is also being explored in order to directly analyze the audio content. Assuming Amalgam is used as a study tool as intended, then most videos embedded should be educational tutorials and presentations thus the subtitle will almost always be sufficient. This is why the Amalgam development is not dedicating too many resources to building frame extraction and analytics tools for videos yet rather, it is focussed on making videos easy to access and play within Amalgam.

Database Structure

For all content other than uploaded documents, we convert them into some form of representative string which we then save into the several flask sqlalchemy models. We have three main models, the notebook, base resources and support resources. Let us break these down:

Structure: Notebook model

The notebook model is the root model for Amalgam.It unifies the base and support resources. A notebook at its minimum contains just a title. Once created, a user can then proceed to add base and support resources that are back referenced to the notebook parent. A ‘delete orphan’ option is also activated to ensure that should a notebook be deleted, all the children’s resources are deleted in a cascading effect. This ensures we do not have to manually delete these when the delete notebook process is invoked and also allows us to deal with base resources separate from the notebook. The notebook id is used as a secondary key unifying resources that have any particular notebook as a parent. This effectively eliminates that as a possible transitive functional dependency since the modification of the notebook state is not dependent on any of the other resources. Similarly, modifying the base and support resources is not dependent on anything else other than a notebook existing.

Structure: Resource model

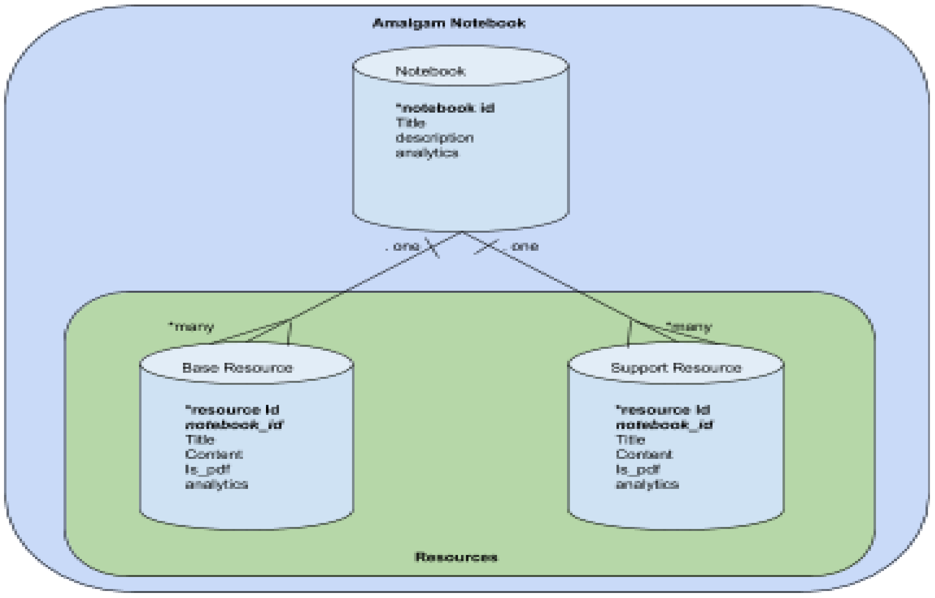
There are two kinds of resources which we have extensively discussed, base and support resources. These can be pdfs or full-on text. The notebook model contains several columns including a mandatory title made on creation, a boolean variable indicating whether it is a pdf also selected during creation, the content can either be user entries in case the data is not pdf or an auto generated URL of the uploaded resource if it is. There are multiple auto-generated values found in both types of models. We discuss these shared attributes in the next section. In summary, all forms of text content, html tags as generated by the editor, the styling sheets, embedded video and image URLs among others are converted to strings which are then saved in the content section of the resource module. Images, in particular, are converted from their graphs pixel values form to byte64 strings and embedded as data parameters for the <img/> tags and then converted to string along with other content. This byte64 data is what the image analytics tools we discussed analyze. We use beautiful soup to extract this and send it to the online classifier for analysis and is converted to jpg temporarily for local analysis by the local neural network. The editor has inbuilt mechanisms to facilitate the display of the content entered. In case the content is pdf, we save the file and then the content section of the resource model is populated with the URI of the pdf which we feature on display. Using the pdfviewer package.

Shared/ Common attributes.

Both the resources and notebooks keep track of analytics. The bases keep track of the analytics data for all content relative to each other. This redundancy keeps the data accessible and safe and allows for the modification of base and support resource analytics data simultaneously and only upon committing do we modify the notebook analytics data. Analytics data is generated when resources are modified. By saving the information, we prevent recomputation since this usually entails the conversion of entire resources and reindexing of the analytics objects. In the future, we will seek to keep track of which sections are modified to prevent the recomputation of unmodified data analytics but for now, the system simply saves this data and then the front end modules process it for display.

Normal forms

The current structure of the notebooks database is such that there is no transitive dependency between resources and notebooks. There is no transitive functional dependency between any of the values we save. The one potential source for transitive dependency is with analytics since notebook analytics depend on resource analytics which depends on the resource data. To prevent the notebook analytics → resource data transitive functional dependency, we simply saved the notebook analytics separately with the notebook module. We already have our data in first normal form since we kept a related notebook and resource data separate, have no repeated grouping as each column carries Its own compact content and we have a primary id key for each. We satisfy second normal form conditions since we are already in first normal form and all non-key attributes are fully functional dependent on the primary key i.e the id. The resources, despite being attributes of the notebook are separately indexed with their own keys. We thus eliminated transitive functional dependencies implying our model structure is currently in third normal form.



Structure: PDF Files

Amalgam presents a dropdown that allows the user to select the resource type they want to add or upload. All resource types other than pdfs are handled within the main editor but pdfs are special, they are handled by an upload system.These are saved in memory. We have already discussed how, post upload, the text is extracted from them, chunked, preprocessed and analyzed the same way we do normal text. pdfs being static documents present a considerable challenge transforming into manipulatable formats like HTML (CodexWorld. (2019, June 4)). We will lose all styling which could alter the flow of information as intended by the authors. Atop that, a lot of research work comes with watermarks and copyright protection mechanisms. A lot of pdf documents are pdfs because the original authors intended they stay static. For that reason, Amalgam simply saves the documents and provides the mechanisms to feature them. Once uploaded, the document is saved in subdirectories in the static folder and given a hash code name.[19] A URL for the saved document is then saved as the content for the resource and the is_pdf flag is set to true. The document can then be retrieved using the appropriate URL. The documents are accessible to other notebooks. The URLs are not session protected thus we do not advise using Amalgam to analyze sensitive documents unless using the local version of the system.

The reason for keeping the URL unprotected is to ensure that when notebooks are shared, there is no need to re-upload pdf URLs. This ensures that the user when recompiling their notebooks does not need to re-upload the individual pdfs after uploading the notebook JSON files and reduces the complexity of

[19] **#CS110-hashing:** Hashed content names for security, to avoid collision and for easy indexing. We also do this to generate unique URLs.

downloading notebooks. This reduces the download of notebooks to a matter of sharing the JSON file that defines how resources are mapped.

PDF files remain unmodified within Amalgam, they are display-only. This allows their use in multiple notebooks simultaneously. Should a user desire to modify the file, a download option is provided to facilitate this and provide them with a local copy that they can do their write operations on. Once uploaded, all they will be able to do is read it, analyze and display it using the Amalgam editor frames. In terms of analytics, there is no benefit to saving pdfs as uploaded since we end up having to extract the text anyways so their analytics complexity is roughly the same as the raw text complexity. In both instances, we scrap the text out of the tags that give the information its structure. We sacrifice text extraction speed to ensure we maintain document structure as this is key in maintaining the flow of text data which usually carries a lot of context information necessary to articulate the gist of said documents. We are already ignoring the context in our analytics thus stripping the user of this too would be an injustice and would render Amalgam almost useless from a day to day non-technical user perspective.

Extra Functions

Downloading notebooks

It makes little sense if once uploaded, notebooks cannot be shared with other users. While there is a myriad of file formats that documents can be exported as, Amalgam draws inspiration from Jupyter notebooks and Google collab to facilitate the download of notebooks as JSON files. JSON is lightweight, very widely supported and is the most commonly used resource sharing format. This means that Amalgam notebooks should be easy to share and process online given the wide support for JSON files. Atop that, the format is supported across multiple languages natively. With python, for instance, we build up the resource being exported as a python dictionary before using Python's built-in JSON library to dump it which generates a JSON string. This can easily either be evaluated by the python compiler or even loaded using the JSON package itself. The same file is used to parse analytics information to the charts which show up in the dedicated view for Support resources using JavaScript. JSON stands for JavaScript Object notation so there is no extra processing of this information needed other than just parsing the JSON string into Javascript then proceeding to use it as one would a normal JS object. This ease of transfer continues with practically all modern languages, there always is a data structure that resembles the JSON format be it hash tables, dictionaries, objects, etc.

At the moment, as soon as a notebook is created, an editable JSON object is created. This object is saved under the JSON generated folder with the notebook id as its name. The id was chosen given it is unique to each notebook. And it makes it easy to identify the accompanying notebook being processed. A valid notebook requires only a title although upon download, the notebook takes the following format:

```
{
        id: 1,
        title: "title",
        date_posted: "2020-02-24 05:47:19.327410",
        description: "optional desc",
        analytics: "{}",
        user_id: 1,
        base_resources:
        {
                0:
                {
                        id: int,
                        title: "title",
                        date_posted: "2020-02-24 05:48:38.081533",
                        notebook_id: 3,
                        relevance: 0,
                        content: "html content or data",
                        analytics: "{'Support Resource 4': {'Jaccard Similarity':float 'Word Mover': int, 'Text
                        Distance': float, 'Cosine Similarity': float}}",
                        is_pdf: false,
                        pdf_url: "NONE"
                }
        },
        support_resources:
        {
                4:
                {
                        id: 4,
                        title: "AMD main",
                        date_posted: "2020-02-24 05:49:14.291664",
                        notebook_id: id number,
                        relevance: integer,
                        content: "content URL or text/HTML string",
                        analytics: "{...}',
                        is_pdf: true/false,
                        pdf_url: "NONE"
                }
        }

}
```

This is the general format when downloading and auto-generating. The minimum format though assuming no base or support resources would be:

```
{
            id: 1,
            title: "title",
            date_posted: "2020-02-24 05:47:19.327410",
            description: "optional desc",
            analytics: "{}",
            user_id: 1,
            base_resources: {},
            support_resources:{}
}
```

The upload processor will load the information in this as appropriate. A full example of this is shown in the demo as well as a generated file example from the download button.

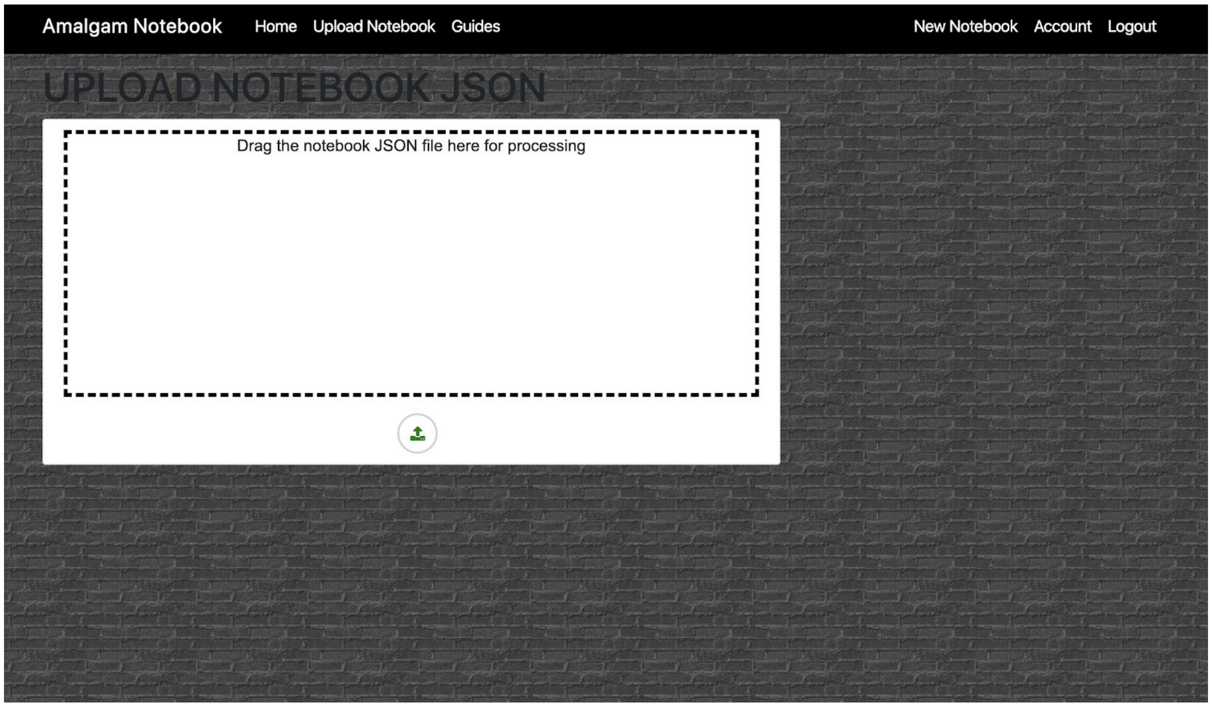
Uploading notebooks

The upload functionality is a simple drag and drop upload field/ button with constraints to take only JSON files. As additional download formats are added, these constraints will, of course, be expanded. On the backend, the entries to the field are handled by a route that first calls a validator. As stated initially, a valid JSON is one that has a title. Any optional and variable features such as support resources are not going to prevent the upload of a document. The notebook description can be null, but there must be a title.

Once the JSON is uploaded, the system initiates a notebook and assigns it a new id. In this regard, when a notebook is uploaded, not all information is carried over to avoid collisions in the database. The primary key for notebooks is the id thus Amalgam does not assign this, it automatically generates one in light of the database structure at any given time. It populates the description with that which the user provides or an empty string if none exists. Should base resources exist, it loops through these adding them to the database session, then it does the same with the support resources before committing the added information to the database. The initial database commit of the notebook is to ensure we first have a notebook id generated so that the added base and support resources have child relationships defined with their parent notebook. This way, should one delete the notebook, then by hierarchy, the resources are deleted too.

The upload system is implemented to support files of up to 50MB. The reason for this constraint is that most notebooks JSON files are pure strings. These are rarely big files thus imposing a size constraint prevents system abuse by hackers. An upload API is in development to facilitate mass addition and sharing of notebooks and facilitate the development of tools. Currently, we are even exploring the possibility of exporting resources on their own although given that this can be achieved by a simple copy and paste, it is not a priority.

It is also worth pointing out that the upload functionality was implemented as a drag and drop feature to facilitate easy uploading. Amalgam drew inspiration from the preclass notebooks that some Minerva professors provided as a study guide for technical classes and these usually are created on chrome which allows for uploads of files through either drag and drop or button upload, a philosophy Amalgam follows. Experts suggest that drag and drop features are so intuitive and easy to use that people often overlook them since they do not require the memorization of file locations and paths rather, access to the file (Noh, G. (2018, January 19)). This prevents the redundancy that locating a file presents when using non-drag and drop file input handlers. Should one have the working directory open, they can easily drag and drop the file into the upload field although the option to open the file handler is also presented in case these files are not open.



Code modularization

The current intent is to make Amalgam open source immediately after Capstone submission. For this reason, the program is split by functionality. The machine learning algorithms are made separate as well as utility functions while routes are kept in one file. Further modularization is being explored to separate related routes and have them in separate files to ensure that in the future, these can be maintained separately. In doing so, developers can work on multiple sections without worrying about collision. A developer can thus focus on specific functionalities which will correspond to specific files. They can then be instructed on the exact files they need to modify to create new features, fix bugs etc while limiting their changes to said documents. It also aids with the detection of bugs through unit testing to ensure various modules in the code work well together as planned.

One way that modularization has been done extensively is with the page template organization. There has been minimal recreation of the wheel. Code sections being imported regularly where needed. The layout has been reused and flask block tags used to post content to the sections under these tags. The templates were fairly easy to modularize since they served separate functions and host different components. In fact, a lot of the user interface components are dynamically generated and populated with data visualizations before futuring the custom content within their parent templates. This approach allows for the maintenance and modification of pages separately without affecting the unrelated pages.

Modularization can also be observed with the forms (Flask, WTForms, and Quill). Amalgam makes use of multiple forms to collect data. Instances of these are initialized for use within routes. There is a lot of separation of concerns done depending on the fields a form is required to retrieve from users. As such, the following ten forms are defined in the form file:

Form	Purpose
NewSupportResourceForm	Support Resource initialization form with no content.
NewBaseResourceForm	Base Resource initialization form with no content.
SupportResourceForm	Full support resource form when updating including content.
BaseResourceForm	Full base resource form when updating including content.
NotebookForm	New notebook form
UpdateAccountForm	Account modification form with validators

LoginForm	Account login form
RegistrationForm	Account registration form
AddPDFForm	Upload PDF file under resources form. Reused for both resource types.
AddJSONForm	Upload JSON file form.

[20]

[20] **#CS162-abstraction, #CS162-separationofconcerns:** We modularize the code extensively to separate the different functions for easy maintenance as well as reuse a lot of modules to minimize the amount of repeated work. For this, we had to define a lot of the complex functionalities carried and performed by the forms, editor amongst others, all of which we simply imported into the system after having one main implementation that specified our configurations for the packages that offer the form and editor services.

Demos

HCs and LOs

We have already discussed Amalgam's potential use as a plagiarism detection tool, specifically with the cosine score. Within Minerva, we can leverage this to use Amalgam to track applied HCs in the case of students and for professors, to check for HCs that may have been missed during the grading process. For all parties, Amalgam can be used as a tool to track which HCs should be applied and interestingly, as we will see in the demo, it can sometimes reveal information that the reader may have not had an intuition of. This is because sometimes one may inadvertently apply HC in the process of applying some other HCs and LOs. Amalgam may serve as a reminder for the user to check these out.

View the demo here:

AMALGAM HCS DEMO

Study guides and skimming

Similar to the HCs and LOs application, one can use Amalgam to preemptively skim a document and get an idea of the pages to focus on when reading the document based on a study guide. This is due to the fact that a lot of study guides as prepared by professors are usually short summaries containing keywords from recommended reading sections. Amalgam does an even finer selection and presents the user with a document layout accompanying the resources[21]. As such, it comes in handy in case there are multiple

[21] **#CS162-communication, #CS162-agile**: We discuss why we modularize the code, give them appropriate names and have extensive comments within the repository as well as a video guide on how to use the system, how to install it amongst others.Its has also been the case that project development has been gradual based on feedback from the students, professor among others to get to the level we are at today hence the agility.

readings to get through. Currently, we are exploring auto-scroll technologies to embed atop the heatmaps to ensure the document automatically scrolls to the pages that are marked when a user clicks on the heatmap instead of having them input the page themselves although this is not to be expected in the final submission. The analytics visualization though is already implemented and all data on the relevance distribution visualized within the dedicated view page.

View video demo for Study guide application here:

AMALGAM STUDY GUIDE DEMO

Amalgam installation guide

This information is pasted as posted in the GitHub readme file. The link to the repository is included below

Github Link: https://github.com/GitWahome/Project-Amalgam

Project-Amalgam

A resource Standardization Utility with a Machine Learning backend for Relevance analysis and automated Skimming.The system aims to provide alternative formatting to study material such that distractions are minimized. In extension, a lot of internal analysis has been implemented to dictate how resources are organized by metrics such as Cosine Score and Jaccard Analysis. Ease of use is a key feature thus a lot of the resource compilation is intended to be drag and drop, Copy and Pasting and Direct Uploads.

To run the demo, clone the repo. You may use the command:

```
> git clone https://github.com/GitWahome/Project-Amalgam/edit/master
```

Afterward, navigate to the project directory. You may wish to create a virtual environment within which to work but that is all up to you. Whether in a virtual env or your global environment, you will need to install several packages to make the system work. A requirements.txt file is generated from which you can install all the packages using the command:

```
> pip3 install -r requirements.txt
```

After that, you should be able to launch the program by running the run.py file

```
> python3 run.py
```

The project is still a work in progress so keep an eye on new functions, redesigns and capabilities. The creation of db and model creation is automated under the run.py file hence why it is sufficient to just run it that file and the whole thing will come to life. The database is named site.db. Do not delete this file otherwise all your notebooks will be lost.

After that, launch the system in your browser at the url:

```
> http://127.0.0.1:5000/
```

Play around with the features, and to see the demo live, you may follow the youtube link below:

AMALGAM FEATURES GUIDE

References

321.pdf. (n.d.). Retrieved February 17, 2020, from http://www.cs.yale.edu/homes/radev/nlpclass/slides2017/321.pdf

API. (n.d.). Quill. Retrieved February 22, 2020, from https://quilljs.com/docs/api/

Applications—Keras Documentation. (n.d.-a). Retrieved February 22, 2020, from https://keras.io/applications/

Applications—Keras Documentation. (n.d.-b). Retrieved February 22, 2020, from https://keras.io/applications/#xception

Beautiful Soup Documentation—Beautiful Soup 4.4.0 documentation. (n.d.). Retrieved February 28, 2020, from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Bernecker, S. (Ed.). (2008). Diachronic Content Similarity. In The Metaphysics of Memory (pp. 155–167). Springer Netherlands. https://doi.org/10.1007/978-1-4020-8220-7_9

Building A Custom Module. (n.d.). Quill. Retrieved February 28, 2020, from https://quilljs.com/blog/building-a-custom-module/

Similarity Learning, Ch3.pdf. (n.d.). Retrieved February 21, 2020, from http://infolab.stanford.edu/~ullman/mmds/ch3.pdf

CodexWorld. (2019, June 4). Convert HTML to PDF using JavaScript. CodexWorld. https://www.codexworld.com/convert-html-to-pdf-using-javascript-jspdf/

Definition of AMALGAM. (n.d.). Retrieved February 27, 2020, from https://www.merriam-webster.com/dictionary/Amalgam

Euclidean vs. Cosine Distance. (n.d.). Retrieved February 21, 2020, from https://cmry.github.io/notes/euclidean-v-cosine

Flask, WTForms, and Quill—Daniel J. Beadle. (n.d.). Retrieved February 22, 2020, from https://danielbeadle.net/post/2019-01-24-wtforms-quilljs/

Google Colaboratory. (n.d.). Retrieved February 27, 2020, from https://colab.research.google.com/notebooks/welcome.ipynb

How many tabs do people use? (Now with real data!). (n.d.). Retrieved February 12, 2020, from https://dubroy.com/blog/how-many-tabs-do-people-use-now-with-real-data/

HowtoReadPaper.pdf. (n.d.). Retrieved February 12, 2020, from https://web.stanford.edu/class/ee384m/Handouts/HowtoReadPaper.pdf

Image Recognition API, Computer Vision AI – Imagga. (n.d.-a). Retrieved February 22, 2020, from https://imagga.com/

Jaccard Similarity—An overview | ScienceDirect Topics. (n.d.). Retrieved February 21, 2020, from https://www-sciencedirect-com.ccl.idm.oclc.org/topics/computer-science/jaccard-similarity

Minerva-HCs-Intro.pdf. (n.d.). Retrieved February 27, 2020, from https://www.minerva.kgi.edu/public/media/enrollment-center/Minerva-HCs-Intro.pdf

Mirković, J., & Altmann, G. T. M. (2019). Unfolding meaning in context: The dynamics of conceptual similarity. Cognition, 183, 19–43. https://doi.org/10.1016/j.cognition.2018.10.018

Natural Language Toolkit—NLTK 3.4.5 documentation. (n.d.). Retrieved February 28, 2020, from https://www.nltk.org/

Noh, G. (2018, January 19). Drag and Drop for Design Systems. Medium. https://uxdesign.cc/drag-and-drop-for-design-systems-8d40502eb26d

PDF File Format: Basic Structure. (2018, May 6). Infosec Resources. https://resources.infosecinstitute.com/pdf-file-format-basic-structure/

Project Jupyter | Home. (n.d.). Retrieved February 27, 2020, from https://jupyter.org/

Shinyama, Y. (n.d.). pdfminer: PDF parser and analyzer (Version 20191125) [Computer software]. Retrieved February 28, 2020, from http://github.com/euske/pdfminer

sklearn.metrics.pairwise.cosine_similarity—Scikit-learn 0.22.1 documentation. (n.d.). Retrieved February 21, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

Supervised Learning—An overview | ScienceDirect Topics. (n.d.). Retrieved February 27, 2020, from https://www-sciencedirect-com.ccl.idm.oclc.org/topics/computer-science/supervised-learning

text—Can someone give an example of cosine similarity, in a very simple, graphical way? (n.d.). Stack Overflow. Retrieved February 21, 2020, from https://stackoverflow.com/questions/1746501/can-someone-give-an-example-of-cosine-similarity-in-a-very-simple-graphical-wa

The Story Behind LinkedIn’s New Publishing Platform. (n.d.). Retrieved February 22, 2020, from https://engineering.linkedin.com/blog/2016/09/the-story-behind-linkedins-new-publishing-platform

Zahrotun, L. (2016). Comparison Jaccard similarity, Cosine Similarity and Combined Both of the Data Clustering With Shared Nearest Neighbor Method. Computer Engineering and Applications Journal, 5(1), 11–18. https://doi.org/10.18495/comengapp.v5i1.160