Why use computers for textual analysis in the first place?

A lot of the most complicated textual analysis programs work by counting words and classifying words/phrases in texts.

* Read more stuff using a computer than you can by yourself. Computers can also detect the words that appear near each other constantly and see a relation between the words.
* Classify texts (or parts of texts) together (related to the idea of relations between words, but in the sense of how the words come together to form meaning in a way unique to the writer). **Sentiment analysis** is used by big companies to see if customers give good ratings to their products, which should help with this project as well.
* Find and represent networks (related to finding and showing different patterns of ideas that may come within a text).

First, you have to understand the data set that you are working with (whether it be from a database, surveys, etc.), and then you have to prepare the information in a specific way before analyzing the data (this is the preprocessing step).

<https://guides.temple.edu/corpusanalysis#:~:text=Computer%2Dassisted%20textual%20analysis%20just,you%20could%20feasibly%20read%20yourself>.

What about the use of computers for textual analysis of survey data?

A lot of surveys have either numerical responses (based on a scale of 1-10, etc), text response (customer freely explains pros/cons of the product) or both (on a scale of 1-10, with other being an option for explanation).

Numerical-based surveys is easy to analyze, but not much can be understood from these types of surveys due to the limited choices that you have and the lack of explanation for as to why you chose them.

On the other hand, textual-based surveys are harder to analyze, but you have a greater amount of insight to the needs of the customer. Survey text analysis is still rare, and even when it is done, it must be done manually, which can be expensive. The surveys are also limited to the understanding and the biases of the analyzer, which causes a high degree of unpredictability for the analysis of the surveys. Computers don’t suffer from these limitations and are therefore are a good way of implementing textual analysis of survey data.

Two of the most widely used techniques when talking about text analysis are either sentimental analysis (how the author may be feeling while writing the review) and topic detection/categorization (grouping of related words together to show how they relate back to the subject). It would be best to try and use a combination of both sentimental analysis and topic detection in order to look at emotion; sentimental analysis alone would probably not be enough.

* <https://www.surveypractice.org/article/6384-text-mining-in-survey-data>
* <https://www.qualtrics.com/experience-management/research/text-analysis/>
* <https://monkeylearn.com/blog/getting-started-with-text-analytics/>
* <https://monkeylearn.com/text-analysis/>

Discussion of the APIs

**IBM Watson**

|  |  |
| --- | --- |
| Pros | Cons |
| * You can extract relationships, concepts, entities, semantic roles, keywords, etc. (versatile) | * The UI is not very intuitive (settings and requirements aren’t very clear for users |
| * User-friendly | * The pricing can be steep |
| * Quick and high-resolution visuals | * The documentation may seem to be too much for a beginner, but it will become easier with more exposure |
| * Price model is flexible | * More suitable for medium to large sized businesses with lots of data |
| * Quick analysis |  |
| * Community seems very helpful for helping beginners |  |
| * The API seems to give a lot of power to the user through different APIs inside of IBM Watson that deal with different things. |  |

Resources:

<https://www.trustradius.com/products/ibm-watson-studio/reviews?qs=pros-and-cons>

<https://www.pcmag.com/reviews/ibm-watson-analytics>

Documentation for Watson API

<https://cloud.ibm.com/docs/natural-language-understanding>

Documentation for this API seems to be well thought out, and beginner friendly

**Google Cloud Natural Language**

|  |  |
| --- | --- |
| Pros | Cons |
| * Enables you to do tasks like sentiment analysis, entity recognition, topic modeling and text analysis | * Be careful with the pricing |
| * User-friendly | * Documentation isn’t too strong |
| * Suitable for small businesses | * Updates every 2-6 months, but every 12 months a recent version would be deprecated and not supported anymore (could still be used, however) |
| * Good community of developers for help |  |
| * Integration with Google Drive services |  |

Resources:

<https://www.g2.com/products/google-cloud-natural-language-api/reviews>

Documentation for the Google Natural Language API

<https://cloud.google.com/natural-language/docs>

Documentation for this API seems to be well thought out, and beginner friendly

**Microsoft Text Analytics**

|  |  |
| --- | --- |
| Pros | Cons |
| * Supports key phrase extraction, entity linking, sentiment analysis, and language detection | * Could be expensive, depending on how much you use it |
| * Good for processing raw text | * Learning curve |
| * Good documentation | * Reliability can be off sometimes |
| * Good customer support |  |
| * Many advance features provided |  |

Resources:

<https://www.g2.com/products/microsoft-text-analytics-api/reviews#:~:text=%22Great%20api%20to%20extract%20information%20from%20reports%20%22&text=Great%20API%20to%20process%20raw,about%20the%20latest%20product%20releases>.

Documentation

<https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/>

Documentation for this API seems to be well thought out, and beginner friendly

**Open Source Software**

**NLTK**

|  |  |
| --- | --- |
| Pros | Cons |
| * Tokenization, part of speech tagging (classifying words into nouns, verbs, etc), named entity recognition (classifying words into people’s names, locations, organizations, etc), classification, sentiment analysis, access to written texts. | * Slow performance |
| * Well known and has a lot of features | * Learning curve (difficult to learn & use) |
| * Many 3rd party extensions | * No analysis of semantic structure (organization that represents meanaing) |
| * Supports many languages | * The documentation can be confusing to look at compared to private APIs |
| * Free (it is open sourced) |  |

Resources:

<https://medium.com/@tomaszbak/python-nlp-libraries-features-use-cases-pros-and-cons-da36a0cc6adb>

Documentation:

<https://www.nltk.org/api/nltk.html>

Reading the documentation for this is going to be tough.

**Tensorflow**

|  |  |
| --- | --- |
| Pros | Cons |
| * Good computational graphs visualizations | * No support through windows except through installing it as a python package |
| * Since it is created by Google, it has good library management and additional advantages of good performance, new releases and quick updates | * Low level with steep learning curve |
| * Debugging capabilities | * Not really beginner friendly (requires knowledge of advanced calculus and linear algebra and a good understanding of machine learning) |
| * Scalable | * The documentation can be confusing to look at compared to private APIs |
| * Good community support | * Slow computation speed |

Resources:

<https://www.javatpoint.com/advantage-and-disadvantage-of-tensorflow>

Documentation

<https://www.tensorflow.org/api_docs>

Documentation for Tensorflow looks nice, except the seemingly thousands of functions that they give for the API. Trying to look for a specific function out of the thousands is going to be tough.

**SpaCy**

|  |  |
| --- | --- |
| Pros | Cons |
| * Fast | * Less flexible compared to NLTK |
| * Easy to learn and use (documentation looks easy to understand) | * Sentence tokenization is slower compared to NLTK |
| * Uses neural networks for training models | * Doesn’t support too many languages |
| * Tokenization, part of speech tagging (classifying words into nouns, verbs, etc), named entity recognition (classifying words into people’s names, locations, organizations, etc), classification, sentiment analysis, dependency parsing (defines the “main” word in a sentence and how the other words in a sentence modifies that head main word) | * The documentation can be confusing to look at compared to private APIs |
| * Good community support |  |
| * More object oriented compared to other libraries |  |

Resources:

<https://www.softkraft.co/python-nlp-libraries-features-us-cases-pros-and-cons/>

<https://activewizards.com/blog/comparison-of-python-nlp-libraries/>

Documentation

<https://spacy.io/api/doc>

Documentation for spaCy looks really nice, not gonna lie.

Out of all the possible open source libraries, I would probably go for spaCy, because of its ease of use and beginner-friendliness. There is also a ton of tutorials online that could help us with this, as well as the documentation. However, we would then need to create our own model that would tell the program whether the text we were looking at has positive or negative sentiment, which would take too much time and would require more knowledge on our end of textual analysis and how machine learning works. Because of this, we decided on going for IBM Watson.

**9/22/20**

Currently, we are all learning Python, at least until the end of the month. In this case, while we are all learning Python, I decided to go ahead and get started on understanding how to get our input from ClassieEvals. One valid way of doing this is to download all of the HTML files for the classes we were looking at on ClassieEvals (in our case, just the whole ESE department). However, this isn’t feasible, as it would make our application bulkier and just be more tedious in the long run. It would also be a problem later when Fall and Winter classes are up on ClassieEvals and we decided to download all of the courses from ClassieEvals to use for our program. Because of this, we are most likely going to use libraries (BeautifulSoup and Requests) in Python to webscrape. Webscraping is basically the process of taking information from the web (HTML file) and using that information for the program. We could also put all the information that we have into a csv file that we can create within Python, or just store it in a data structure. As of right now, I have to look at the structure of the HTML files that I am going to be working with and see how I would be able to play around with that file to get the information that I need.

**10/11/20**

For the past few weeks, I was trying to work on a way to access ClassieEvals through the program in Python. I discovered that there was indeed a way to do this, using the requests library in Python. Requests is a library that basically allows us to access the web through using get functions (you just need the website link of the website that you are trying to take the html file from) and also post functions (this is used for giving data to the website you are trying to access information from, as in the case of ClassieEvals). We used requests to access information on the web and use it as a file in our program, and we also used BeautifulSoup in order to parse the file that we had requested from ClassieEvals. BeautifulSoup is another library that is used for parsing HTML files, amongst other things, and in this case, I was able to use both requests and BeautifulSoup together in order to access information from the Internet and take away comments that we need from that file. As of right now, we are still adding to the program in order to take the information from the graphs and use that. We also need to account for erroneous inputs, but overall, we have most of the basic structure of the code down. We are most likely going to be moving on to learning how to use IBM Watson in order process the information next, but in order to do this, we need to learn more about text analytics keywords to understand what we need to do next.

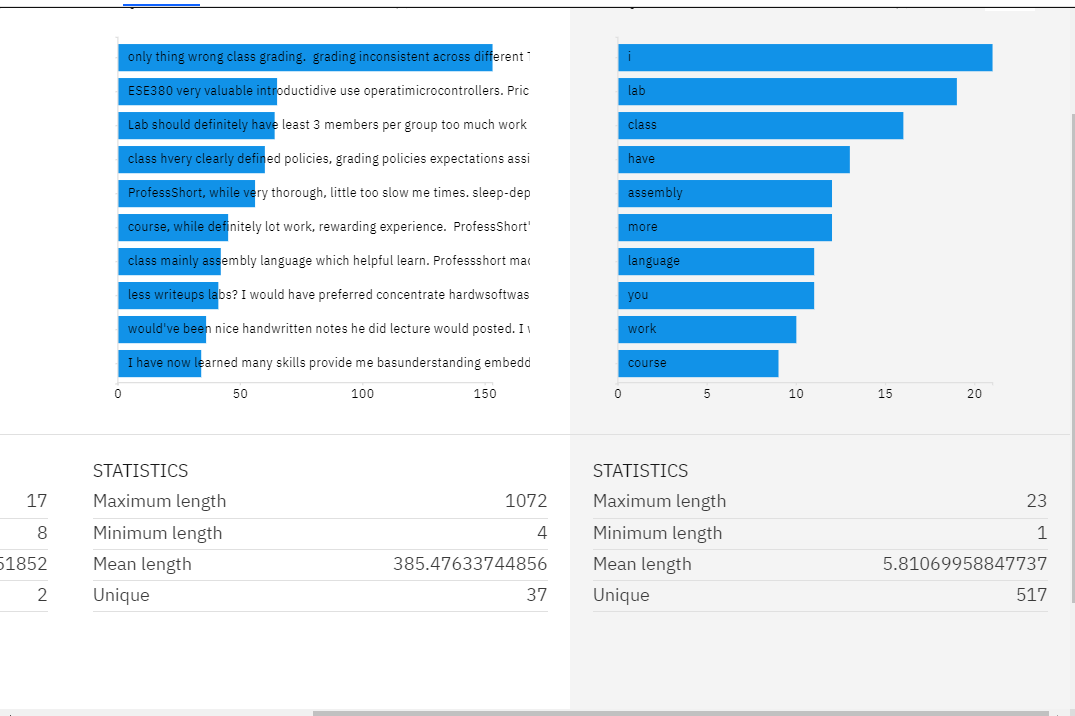
**10/23/20**

Within the past week, we have been looking at the IBM Watson API documentation for the different APIs that IBM Watson has to offer, and we decided that the Natural Language Understanding API has a lot to offer in terms of analyzing text (aka the comments from surveys) that we will be pulling from ClassieEvals surveys. In order to understand what functions we will need to use from the API and how to use them, however, we will need to do a bit of research into how machine learning falls into the realm of textual analysis. To do this, here are some links that will be helpful:

1. <https://monkeylearn.com/textual-analysis/>
2. <https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/>

**11/6/20**

So we were looking a bit into how we could start our project. To get a sense of the data that we were dealing with, we were playing around a bit with IBM Watson’s Data Refinery. So IBM Watson has this thing called Cloud Pak, where they have a bunch of services on the cloud that we could access through authentication keys. Data Refinery is a part of Watson Studio, which is one of those services on the cloud that is used for a variety of Data Science tasks, and it basically helps us try and clean up data before using it for analysis. In our case, we don’t really need any preprocessing, as we can just use the data that we take from Classie Evals, but data scientists would usually use Data Refinery in order to play around with the data that they have by counting the frequency of words, etc.



You could also view the words as a word cloud as well, which is a common task in data analysis.

Here is what happens before we try to remove stop words, and just create a word cloud:



The way word clouds works is that the size of a word corresponds to how frequently it appears in the text. In this case, ‘the’, which is a common stop word, is shown to appear most often in the analysis.

However, if you try to remove stop words from the analysis:



We see that the word cloud most directly represents what we are looking for within the class.

This was a great way of looking into what we are looking for in terms of a class (ESE 380 in this case). Next, we will look at the Natural Language Understanding API, which would allow us to simulate human understanding in machines.

**11/7/20**

Here is a little discussion on the different features that the NLU API have:

**Entities**

***What does it do? What would it be important for?***

The Entities feature in the NLU library identifies people, cities, organizations, companies, etc within a body of text (a being). The IBM Watson API has predefined models that you can look at through this link on the documentation page for the API.

<https://cloud.ibm.com/docs/natural-language-understanding?topic=natural-language-understanding-entity-type-systems>

In this link, IBM watson has a list of the types of entities that are defined (like person, vehicle, etc), and then sub types of entities which are a bit more specific (like blogger, spacecraft, etc). This feature is important if we want to find out the different entities within a single piece of text and find out what role they play within the sentence.

***Can we use it for the project?***

Yeah, we can use this feature to break down the positive and negative comments that we input in in order to find out different things like professors, teaching assistants, different course material topics, class names, etc.

***Limitations***

The only thing is, the default entities model that IBM Watson gives us doesn’t have a professor or a teaching assistant entity option, so we would most likely try and create a custom model in order to deal with this, if we don’t find another way to go around this. We would also need to try to think about what entities we should be adding to our list that we need.

**Keywords**

***What does it do? What would it be important for?***

The Keywords feature returns important keywords inside of the input that is given. We can also try and ask for sentiment information and analyze emotion in the keywords that were returned. In response, we will get a relevance score which rates the relevance from a 0 to 1 (higher values mean higher relevance, maybe referring to how relevant the keyword is to the entire text input). We will also get a score for the emotion and sentiment analysis, if those options have been enabled by setting sentiment = True and emotion = True in the KeywordsOptions setting. This feature is important since we can determine the keywords within the text and use the keywords with the highest relevance in order to initialize the analysis.

***Can we use it for the project?***

It would definitely be useful for the project. We could try and look at different keywords and use the sentiment analysis to see how people feel about certain keywords (sentiment analysis gives values from -1(negative) to 1(positive)). We could then try and see the emotions that people feel within certain keywords that would help us to try and figure if the sentiment score is indeed accurate.

***Limitations***

As of right now, I don’t see any limitations with the Keywords feature. The thing is, I’m not sure how IBM Watson figures out what a keyword could be, and there could be times when a keyword that is chosen isn’t really relevant to the overall analysis of the text. We’re going to need to find a way to filter useless keywords within the code.

**MetaData**

***What does it do? What would it be important for?***

The MetaData feature returns information from the document (URL/HTML) like author name, title, etc. It would most likely be important if you needed information about the article or the page itself. The important thing to note is that this feature is only used with URL and HTML files.

***Can we use it for the project?***

Since we already have the data from ClassieEvals in the form of positive and negative comment lists, this feature is not applicable to us. In other words, this feature is **useless** to us.

***Limitations***

This can only be used with HTML and URL files

**Relations**

***What does it do? What would it be important for?***

This library is used to tell whether two entities are related to each other. An example that the documentation of the NLU API gives is that an Awardedto relationship would be seen with the Albert Einstein entity and the NobelPeacePrize entity. This can be important if we wanted to see how different entities within the body of text we want to analyze are related to each other.

***Can we use it for the project?***

We can use it to see how different entities are related to each other (teacher teaches student), and then look at the adjective that describes how they teach, etc. But honestly, there isn’t too much we can do with this feature at first glance. We would need to play around with this feature some more before making any concrete conclusions about using it for the project.

***Limitations***

There aren’t that many options for customization that we can do for the relations feature. Only thing we can really do for customization is to provide an ID for the model that we would like to substitute for the default model.

**Categories**

***What does it do? What would it be important for?***

The Categories feature returns and categorizes different fields such as Art & Entertainment, Education, Food/Drinks, etc. This is important as it contains a wide range of different things already. Provided is a link to the list of the different categorizations that it can separate into.

<https://cloud.ibm.com/docs/natural-language-understanding?topic=natural-language-understanding-categories-hierarchy>

***Can we use it for the project?***

This feature will allow our program to understand where keywords should be placed. It will organize our text under different topics.

***Limitations***

States that maximum categories that it can return are 3 categories. Not sure if that can change but the categories.limit function defaults to 3.

**Concepts  
*What does it do? What would it be important for?***

This feature of NLU allows concepts to be generated/informed based on the context of the text. For example, if a paper is about deep learning, then this feature will return “Artificial Intelligence”,basically generating an assumption.

***Can we use it for the project?***

We can use this feature to generalize what the topic of the comments are whether is specifically about a TA, equipment, teaching style,etc

***Limitations***

As this may not be a limitation, I am just unsure how IBM Watson reads the input text and generalizes everything into a single keyword.

**Emotions**

***What does it do? What would it be important for?***

This feature allows detection from the emotional spectrum within a certain amount of text. It can detect entity emotion and emotions of certain keywords

***Can we use it for the project?***

Yes we can as this can help play a part within the sentiment analysis that will understand how the user feels towards a certain topic.

***Limitations***

The function that provides emotional analysis targets string words separated by commas. Not sure what keywords are categorized under what emotion and how to generalize those. There doesn't seem to be a list such as categories or entities.

**Semantic Roles**

***What does it do? What would it be important for?***

This feature parses specific sentences into subject, action, and object form.

***Can we use it for the project?***

Yes we can use this because it helps identify certain subjects, actions and objects. Semantic roles can sort and differentiate different keywords in certain sentences.

***Limitations***

The default limit for maximum number of semantic roles returned is 50. To account for all the different types and lengths of the reviews, we would probably either have a default high number inputted, or we have to have something that self checks the input to match the limit #.

**Sentiment**

***What does it do? What would it be important for?***

This feature analyzes the general sentiment of your text. Can also be configured to analyze specific target phrases.

***Can we use it for the project?***

This feature allows for you to look for specific target phrases within text. Can create a custom model ID to override standard se sentiment model. Very important if we have any specific cases to be detected by the analyzer in the reviews. This is good for engineering terminology and abbreviations.

***Limitations***

One limitation of his feature is the difficulty of creating a personal model ID. May be frustrating to account for possible mistakes compared to just using the default ID.

**Syntax**

***What does it do? What would it be important for?***

Tokenization of text speech.

***Can we use it for the project?***

Yes, syntax can be used to identify key adjectives, verbs, nouns, etc.

***Limitations***

A limitation that this provides is oversaturation of obtained data from accumulation of repetitive and redundant tokens.

**11/20/20**

Ideas that we can do for the program:

1. Generate sentiment about the course based on the general comments that we input into IBM Watson (basically add up the course sentiment scores and take an average of those scores in order to generate a sentiment score for the course, similar to what the group from last year did).
2. Generate a word cloud or bar graph based on the keywords that are extracted from the keywords feature in the NLU API library. We would need to add up the scores of the most common words, and then maybe take the next words that are most relevant to the class
3. Consider different categories of the class the teacher may be interested in (labs, projects, lectures, etc.). We would most likely use the entities feature and create a model to allow us to find different entities we are interested in. We could also use the categories feature to help us with dividing the class up into different taxonomies.

**12/28/20**

What we need to do:

1. Complete the sentiment of the course, and the bar-graph based on the keywords
2. Add in a custom model for Entities that would allow us to accomplish the actual machine learning in NLP with number 3. As of right now, we are looking into the documentation in order to investigate creating a custom model within IBM Watson.

**1/11/21**

As stated previously, we decided that this would be our worst-case design for the software:

* generate sentiment about the course based on the comments that we input into IBM Watson.
  + In this case, we would basically add up the comment sentiment scores and take an average of those scores in order to generate a sentiment score for the entire course.
* generate two word-clouds and two bar-graphs based on the keywords that are extracted from the keywords feature in the NLU API library.
  + In this case we would add up duplicate keywords together and output bar-graphs that would show the frequency and sentiment of the most common keywords for the class.

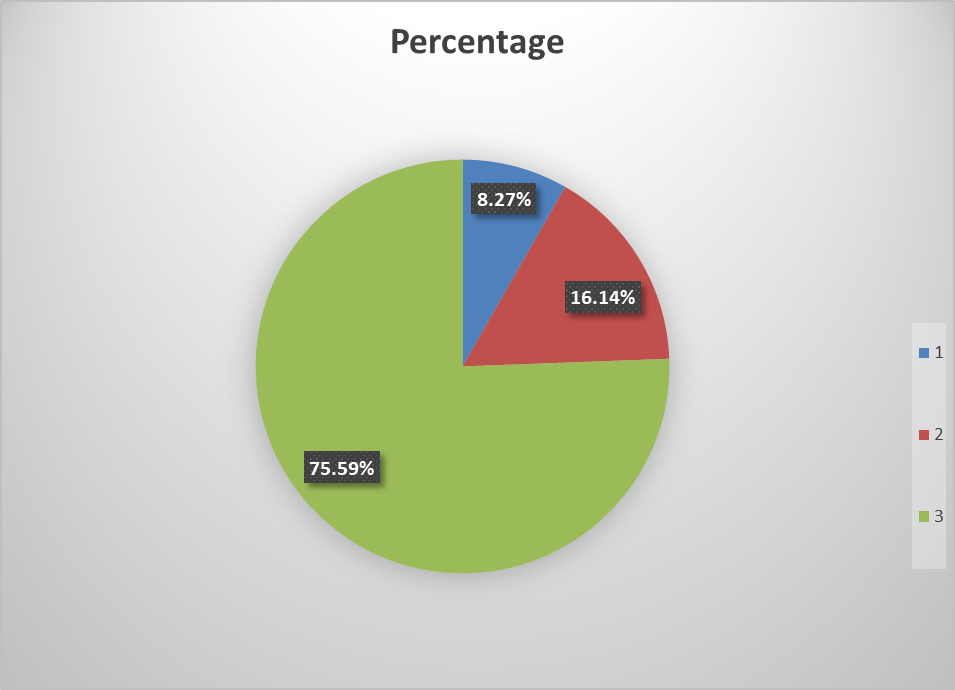
This is what our best-case design for the software would look like:

* Consider different categories of the class that the teacher may be interested in (for example, the labs, projects in the course, lectures, etc.)
  + To do this, we would use the Entities feature and create a model that would allow us to look at different entities that we are interested in (for example, course material, lectures, projects, homework, etc. and view the sentiment/emotion with those entities). This would take more time but is useful in the long run.
  + We could also use the categories feature to help us with dividing the class up into different taxonomies. However, the categories feature doesn’t allow customization (aka using our own categories model). In this case. We’d probably not use the categories feature.
  + Implement a GUI in order to help professors with navigating through the program. This is our least important goal and will only be investigated if we have enough time to implement it.

At this point in the project**, we were able to complete step 1 in the worst-case design**. For step 2, we decided that having both the word-clouds and the bar-graphs outputted would be redundant information, as they both give the same information, just in different ways (they both give the relative frequency of the keyword). In any case, we decided that we would output one of the two instead of outputting both, and in this case, we decided on outputting the bar-graphs instead of the word-clouds. Although the word-clouds look pretty, it can be pretty confusing to look at because of how all the words are jumbled up in a word-cloud (the number of keywords are obviously dependent on the type of class that we are looking at, since not a lot of people do Classie-Evals surveys for the elective classes). We decided to output interactive graphs to allow the user to play around with the graphs and see any words that may not have appeared properly on the graph due to the vast number of keywords in the graph. Because of this, **we were also able to complete step 2 in the worst-case design.**

One thing that was brought up by our advisor was the fact that within our bank of keywords (for valuable comments and needs improvement comments), there happened to be both singular and plural versions of the same words in the banks. Our team members also understood that it would also be a good idea to account for this by using the syntax library and getting the lemma of the keyword and adding together any keyword that has the same lemma. However, we decided that this would be a task that we would be looking at throughout the semester, since we wanted to get onto starting the best-case design of the project, since this part is going to take the most time for us develop.

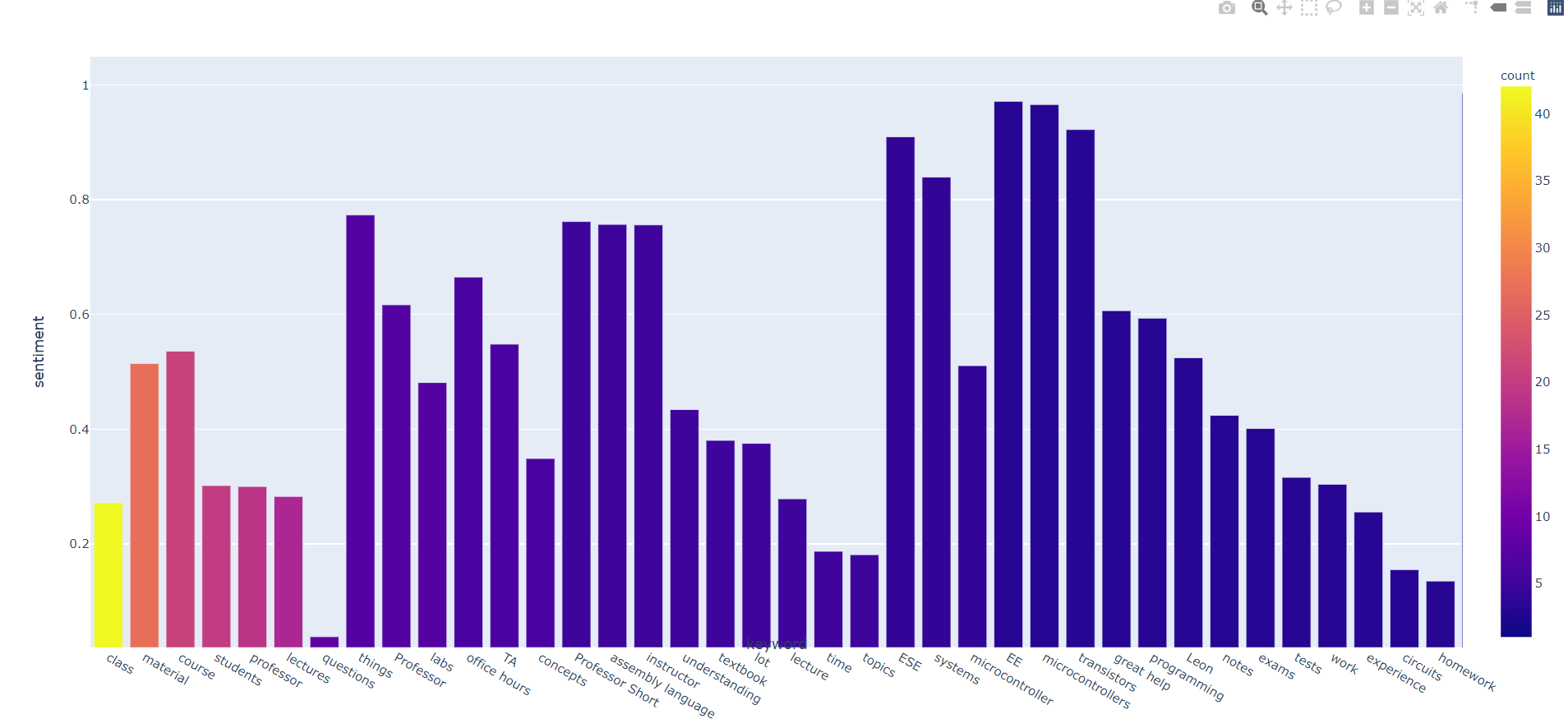
Also, another thing that our advisor had brought up was that she was worried that our sentiment model may not be accurate enough. This was based on us looking at various keywords, viewing the score that the model gave them, and then looking at the comments where the keywords came from and seeing if the score matched what we thought the sentiment value should have been. To address this issue, we decided to analyze the sentiment scores for the class that we were looking at (ESE 380) and determine the percentages based on what was correct, what was wrong and what we weren’t sure about:



The green corresponded with the keyword sentiment scores we agreed with, the red corresponded to the keyword sentiment scores we weren’t sure about, and the blue corresponded with the sentiment scores we didn’t agree with. Looking at the graph, we can see that we have around a 75-76% agreement rate, and the very best that we can say for using the default model is around 91%. In reality, this percentage rate is probably going to be low-mid 80s. There probably is some room for improvement with the default model that we are using, but for now, we will make do with what we have and try to see if we can train the Entities model to more accurately determine the sentiment of Entities we need it to determine.

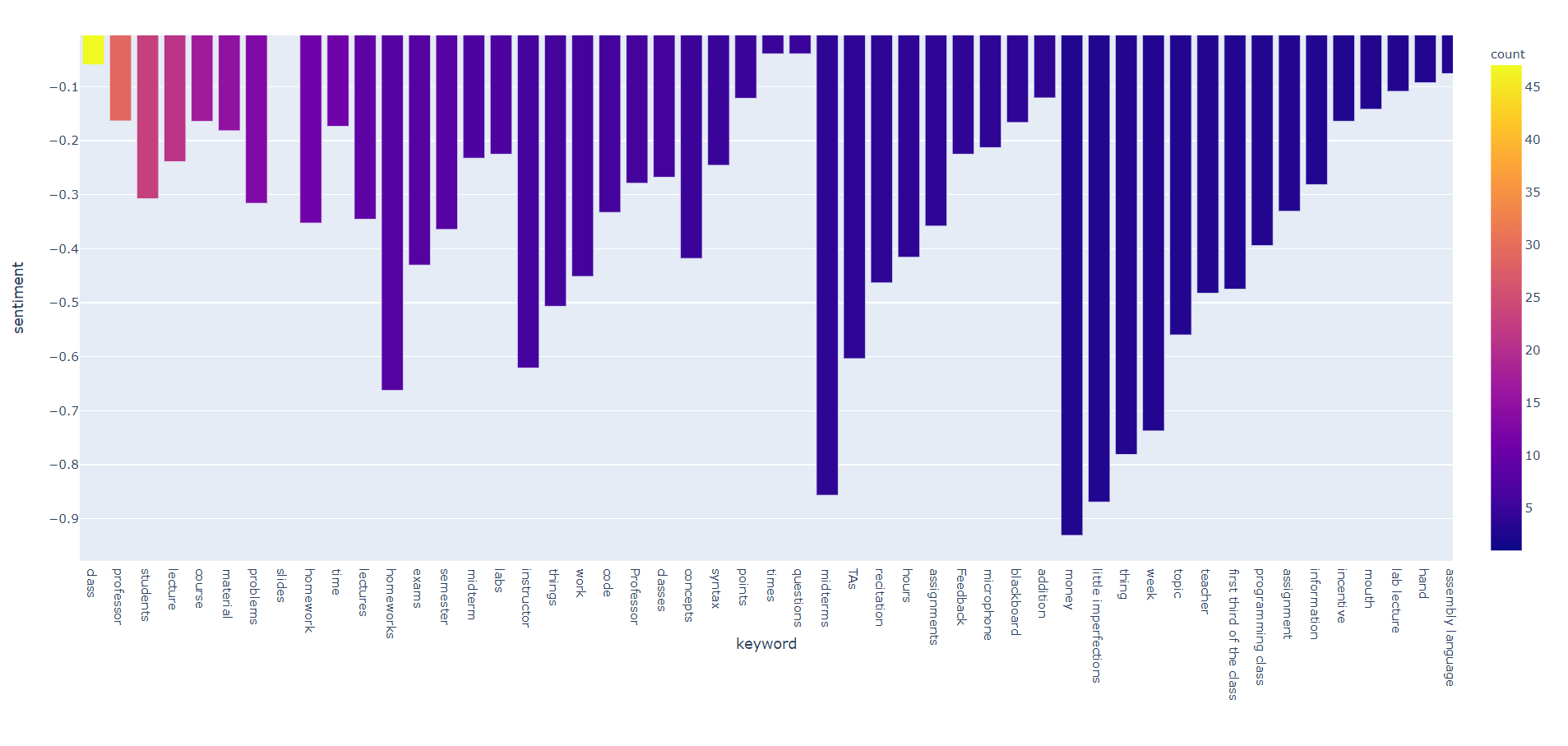
To determine what is important for the type system when creating the entities model, we need to see how we can divide a class into. To do this, we decided to use all the files we currently have in our database as test data. We then added up all the comments into a single file, and then used our current code to analyze what people talk about when they refer to a class in general.

This is what people had to say when referring to what was valuable about a class:



According to this instance, we see that a lot of people talk about either the class, material, course (which is related to the class in our case), students, the professor, and other things like pacing, labs, etc.

This is what we see for what people consider when looking into what needs to be improved:



It seems that this also holds true for the keywords in comments talking about what needs to be improved within a course. We have things like class/course, which seems to be what people talk about the most. Then we see other things too, like the professor, students, lecture, material, homework.

**2/5/2021**

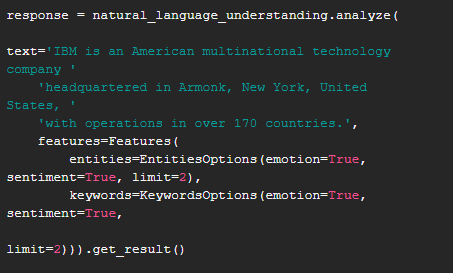
After looking at the graphs for inspiration on how we could divide up a class into individual parts that teachers can use for improving a course, we decided on this setup:

The above picture shows the main entity type system that we had decided upon for a class. We had other entity types for things like people, but as of right now, we were mainly focusing upon the above system as our type system. This is when we started to annotate the documents to try and start training the model to detect the entity types we are interested in.

The **first layer**, the course layer, is the **entity type** we are interested in. We then have the **second layer** (layer starting with lecture) that shows the **entity subtypes** we want to look at. We then have the **third layer** (layer starting with material) that are **the roles** that the phrase plays in the sentence (it is used to clarify the context of the phrase in a sentence).

However, we realized that within the NLU API, when we use the entities feature with a custom model, the model returns a disambiguation list (basically the subtypes entity that we are interested in shown above in the second layer).

Here is an example of this disambiguation list result, given within the NLU API documentation, below, with the default model:





However, in our case, since we are using a custom model, we would not have a dbpedia resource. We would just have the subtypes, and this is obtained from our entities subtypes.

With the method that we have right now, the best that we could do in terms of categorizing a class is with lecture, recitation, contact, exams, projects, homework, and lab categories, because the only categories that we can access are the entity types and subtypes (aka the first and second layers, in the picture above). This is nice, as we can classify the class down to these elements. However, we agreed that this would not be enough since each aspect of these elements can still be divided down further. It would be nice to keep the system we had We then decided to change up the entity type system slightly to address the drawbacks that we had faced.

With this type system, we basically used course as an entity and the other entity subtypes in our previous type system as their own separate entities. We were then able to divide each entity into having their own subtypes, depending on which ones we saw fit to add to each of the entity types. Similar to the previous type system we had come up with, we now also had other separate entities like a people entity to distinguish between the professor, students and TAs, etc.

We were all satisfied with the way our entity type system came out, so now it is a matter of using this type system to annotate documents, which is what we are currently doing right now. We are currently a third of the way through with annotation, and we have to make sure that everyone is on the same page in terms of the annotation style, since we are using only one account to do the annotations (we could use multiple accounts, but we are trying to save as much money as we can so we could use it for later).

**2/27/21**

To ease into the machine learning aspect of the project, we have decided to start training the model as we were annotating. We knew that we would need to create some sort of standard (or guidelines) that would allow us to annotate the documents without any conflicts occurring between us due to variables like differences in mastery of the English language, differences in understanding of how words interact with each other to form meaning in a sentence, etc. To account for all of these variables, we would need to create a set of guidelines that would allow us to decide how to annotate entities, and in what instances we should annotate those keywords in as well. We haven’t started making the annotation guidelines because we want to see if the differences in the understanding of the English language between us is large. This measurement will dictate the amount of effort that would need to go into creating the guidelines for annotation.

Our advisor told us that it would probably be a good idea to focus on one specific class first and do annotations for multiple years of the class to increase the reliability of the model for that specific class. After that, we would then branch off to different classes, until we have achieved sufficient data to allow the model to produce accurate results. To test this, we decided to use all of our completed annotations for classes we have finished annotation (in this case, ESE 380, 382, 372), and the results were lackluster. We were getting results that showed a F1 score of around 38%, a precision score of around 33% and a recall score of 40%.

A **precision score**, defined in the NLU API documentation for creation of a custom entity model, is defined by the number of correctly labeled annotations divided by the total number of annotations added by the machine learning model. In other words, this score basically lets us know about how good the model is at classifying a keyword (entity we are interested in) as the entity types that we created in our type system and specified in the annotations. It will calculate this score based on using our annotations (ground truth) as a reference point, and then check to see if it labels that entity the same way the human annotators label it. **A low precision score lets us know that the machine learning model needs improvement, since the model is generating incorrect annotations** (using the human annotator’s annotations as a reference).

There are many different reasons for this, some of which are given in the documentation as:

* Domain
* Type System Complexity
* Appropriateness of Training Documents
* Human Annotator skills (This will play a big factor for us)
* Clarity of the annotation guidelines

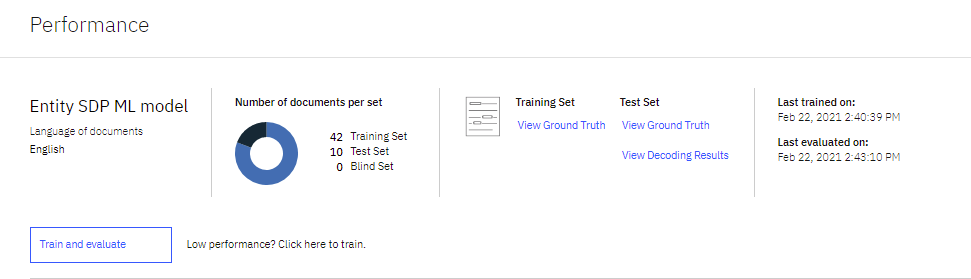
A recall score, which is also defined in the NLU API documentation for the creation of a custom entity model, is defined as a measurement which specifies how many mentions that should have been annotated by a given label were actually annotated with that label. To calculate that, you would take the number of correctly labeled annotations divided by the total number of annotations that should have been made. This score gives us an idea of how much the model misses marking annotations with a given label. **A low recall score allows us to identify places where the model failed to create annotations it should have made (aka it misses a lot of annotations for a given label)**.

The reasons for low recall are the same as for the reason given for low precision scores, which are:

* Domain
* Type System Complexity
* Appropriateness of Training Documents
* Human Annotator skills (This will play a big factor for us)

As of right now, we have started to annotate for a specific class that we have been looking at (ESE 380). We are currently looking at ESE 380 and 382, and we are currently these documents (this will take a large amount of time, as there are tons of comments that we have to go through and we need to make sure that the annotation guidelines are clear for everyone).

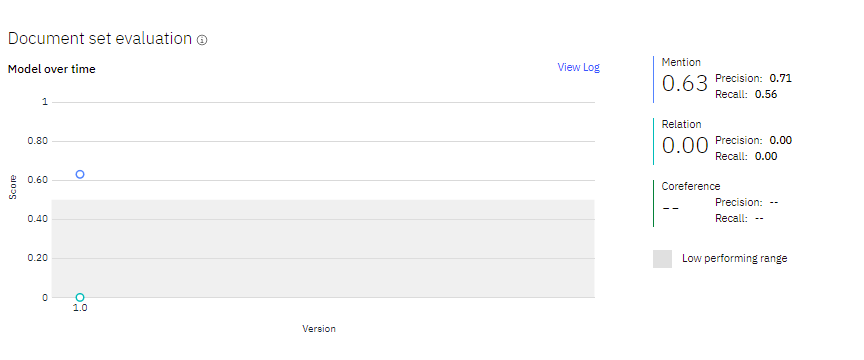
We started to do a little bit of training of the model by the current documents that we have done (in this case4, ESE 380 from Fall 2018 and ESE 382 from Spring 2019). This is the data that we have gotten.



As you can see, this is what we have for the user interface for the training and evaluating of the model. We have separated the documents/comments into 2 groups:

* The first group is the training set (these are the ground truth documents that we are using to train the model). In this case, we are teaching the model about correct annotations. IN this case, we have 42 comments/documents going into this section.
* The second group is the test set (these are the documents that we are using to test and evaluate the model that we have trained). As of right now, we have 10 comments/documents going into this section.

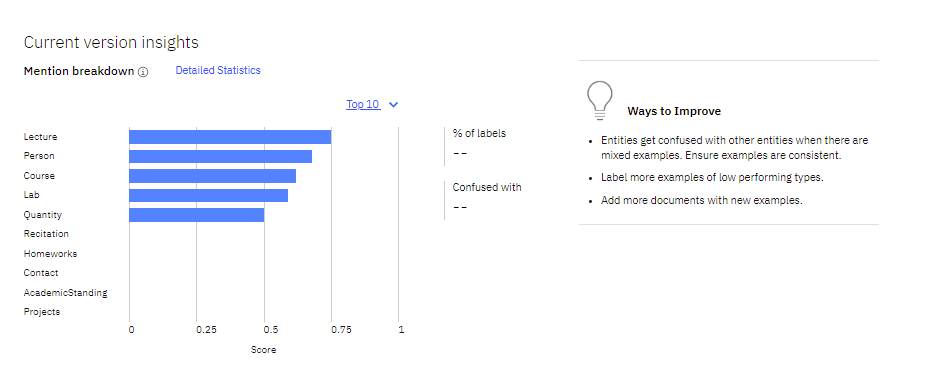
There is also a third group, which is called the blind set. This group is a separate set of annotated documents that is set aside and used to test the system periodically after several iterations of testing and improving have occurred. This is used to prevent the test documents from influencing the training model indirectly as you keep on using the testing data for a while.



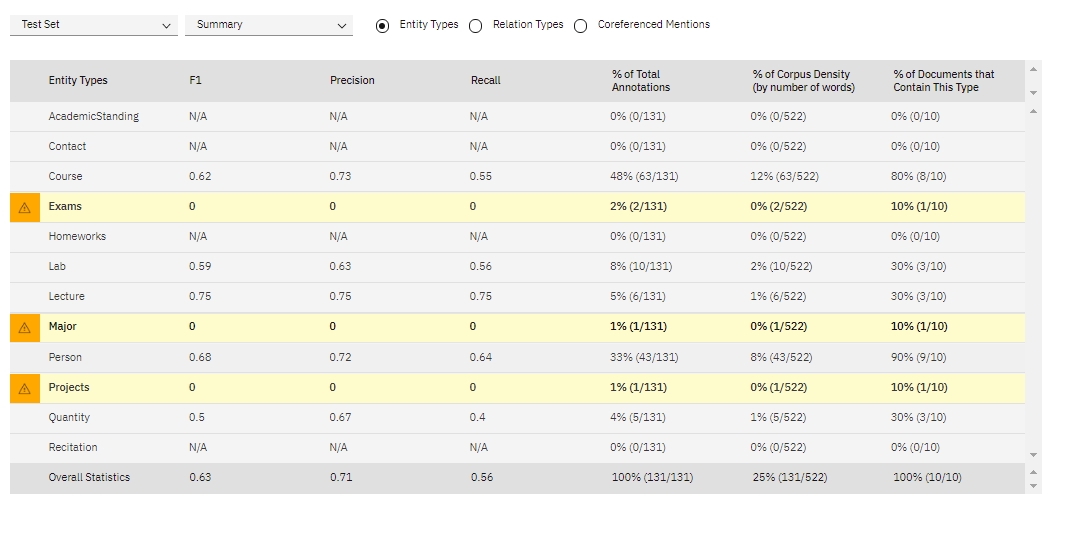
This picture basically gives us an idea of how well the model is doing. According to IBM Cloud, the model is a low performing model if it scores below 0.5. Our model seems to be well above that, but we would still want to try and improve the performance of the current model that we have currently. We have a low recall score, so to improve the model, we would most likely need to find some way of improving the recall score. We would also try to improve the precision score as well, but the most important task to do right now is to improve the recall score. To do this, we are going to check over the annotations, and then try to annotate more documents and use those documents for training and evaluating.



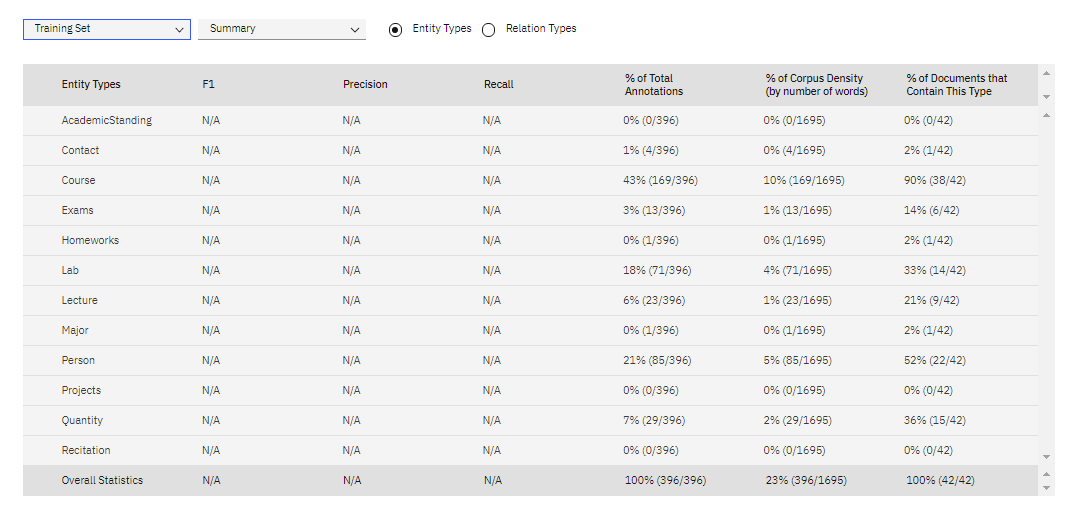
This picture gives us an idea of the entity types that the model does bad on. For any score below 0.5, we know that we would need to improve the score for that entity type, like changing the way we annotate that type to make it more consistent. If you hover over each of the bar graphs, you can also get an idea of which types get confused with the entities that you are looking at on the bar graph.



This picture is also like the picture above, except it shows the entity types for which the model does well on. According to this picture, anything above a score of 0.5 is working well according to the model.



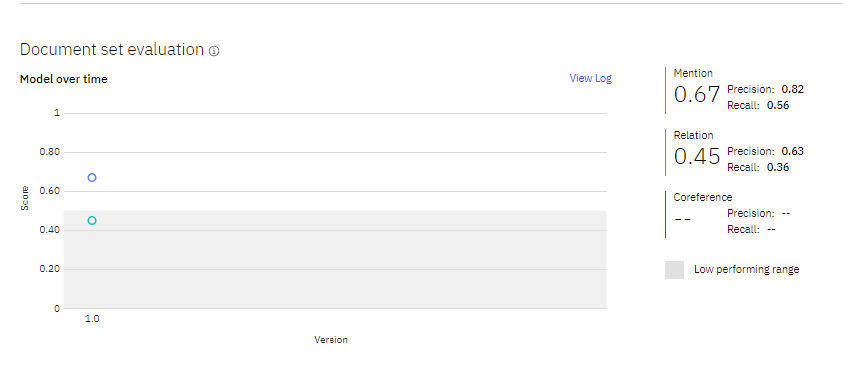
This picture is a matrix that IBM Watson provides that gives similar information to what the previous pictures above give us. However, this matrix gives us all of that information in much greater detail, and also provides an exact number of annotations that were given for each type and some other statistics.



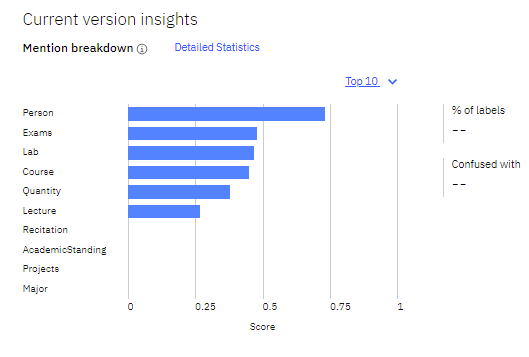
This is also another matrix that IBM Watson provides, but the matrix gives us an idea of how other types get confused for the types that we are looking at.

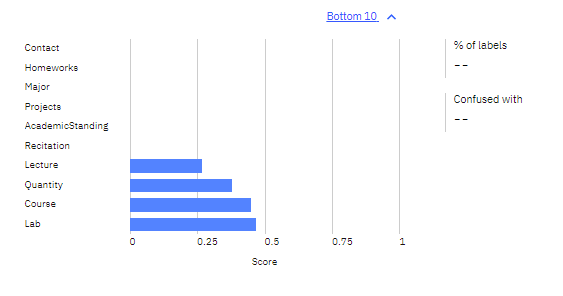
**4/16/2021**

So recently we managed to get the score of the entity model up to around 0.67.

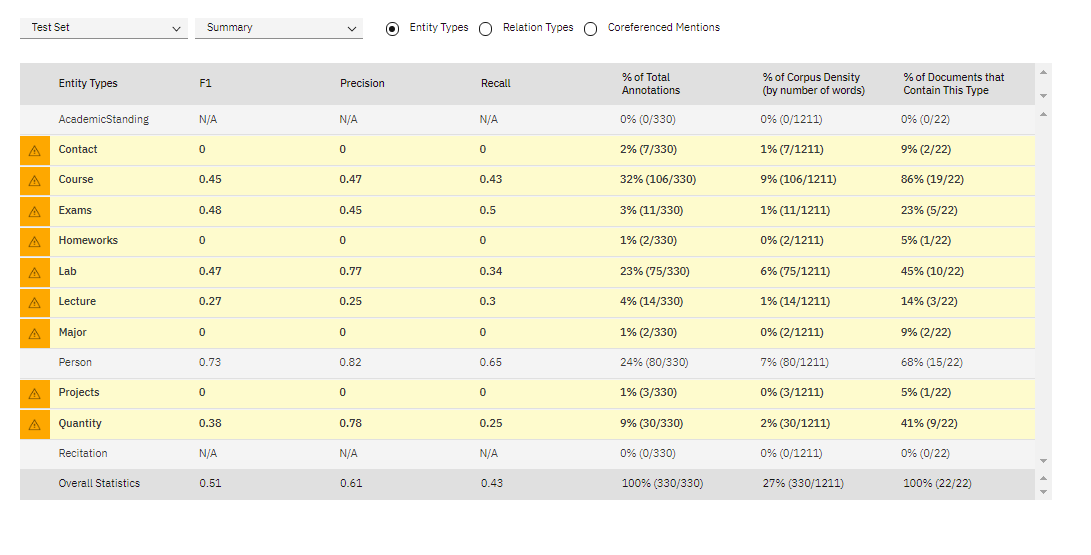


According to IBM Watson, we see that our model seems to work best when looking for people entities, but performs poorly for other entities.





To be fair, we still need to finish checking over Ervin’s annotations, and also need to check up on Richard’s annotations as well.



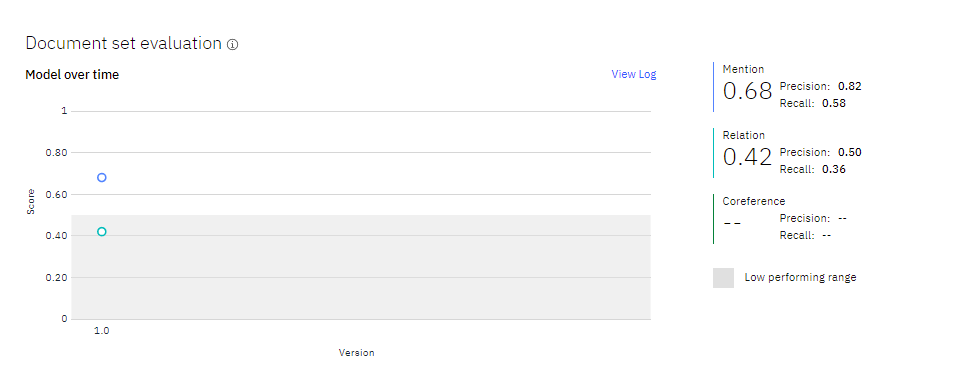
Here, we see where the model is performing poorly. We see that our lecture entity has the worst F1 score, which is to be expected, since we don’t really have as many words that pertain to lectures compared to course. However, for something like the course entity we have here, we see that even though almost 1/3 of our annotations pertain to the course entity, our entity score is still pretty low. We would need to go into the annotations and look to see how we annotated all of those entities, when we get the chance. The important thing, though, is to finish checking over Richard and Ervin’s annotations.

We aren’t too interested in the quantity, the major and the academic standing entities, since those don’t pertain directly to the course. We need to be careful of how we annotate our documents.

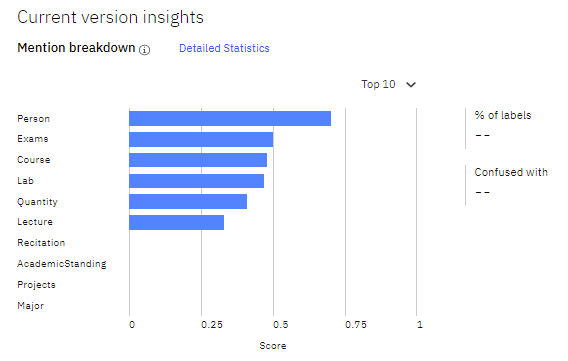


From what we see here, our course entity is confused with other types, and so are some of the other entities too.

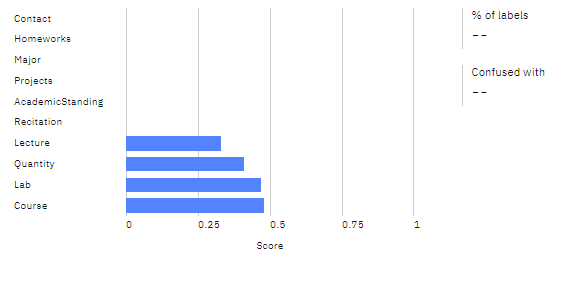
Here is what happened after we finished checking over Ervin’s annotations:



We see that even though the precision score of the model stayed the same, we see a slight bump up in the recall score.

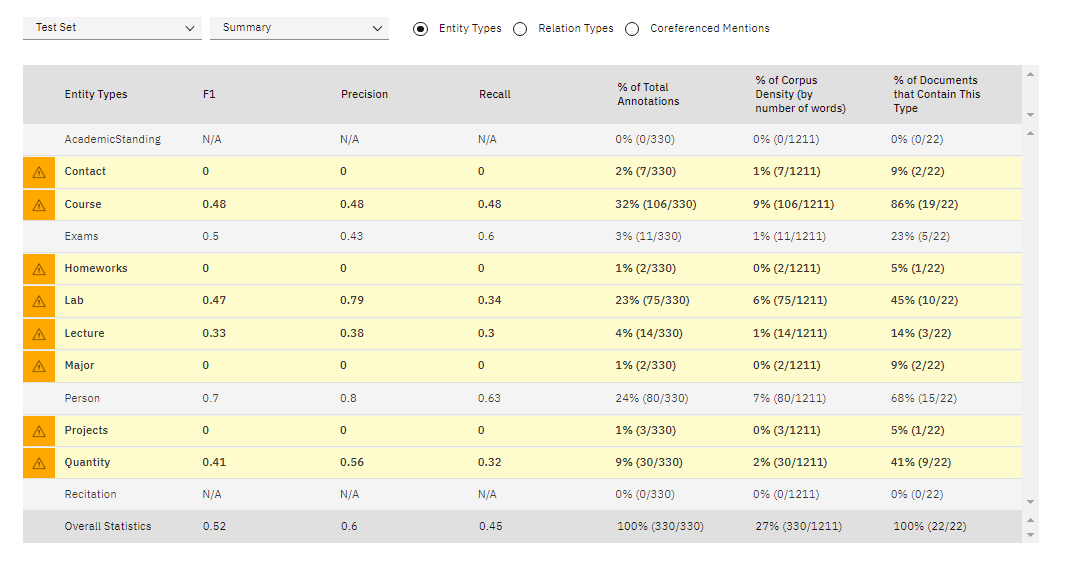


Here is the graph showing where the model does well in the entities we have.

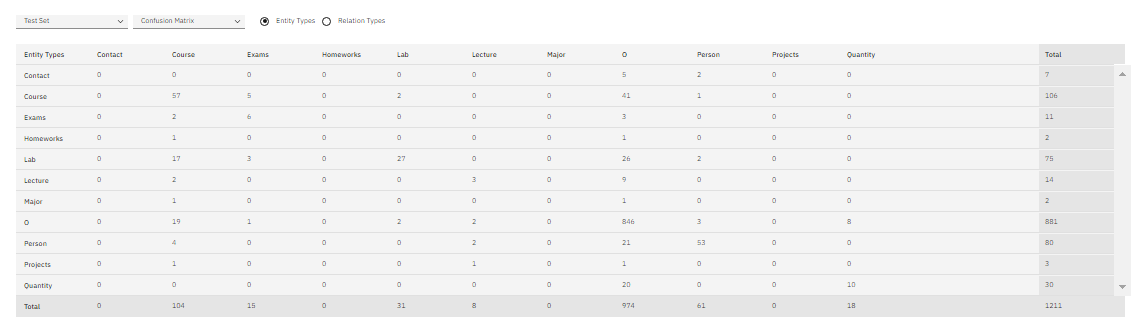


And here is where the model performs least to our expectations. We would want mostly everything to be above a 0.5 at the very least.

Here is an in-depth graph of the above information shown in the summary matrix:



And here is the confusion matrix letting us know where the model was confused:



There’s a good chance we would need to go over Richard’s annotations as well to see what is going on with the model performance.