

Text Mining over biological data to extract protein stability and solubility mentions

Jean-Marc R-Mahoro

University of New Brunswick Saint John, Saint John NB E2K 5E2, Canada
jrugombo@unb.ca

Abstract. There are many biotechnology and bio-pharmaceutical industries that find use in proteins. We need to know the stability [4] and solubility of these proteins because their production and engineering critically depends on whether these proteins are stable or soluble. If a protein is poorly stable, misfolded, and/or aggregated, it can have impact on the health of the cell regardless of the function of the protein [1]. To know the stability and solubility of these proteins can be done by reading a biological corpus including such data. Research scientists can use their current methods to find the stability and solubility to physically collect all the data, and also have to read through each paper to find out the stability and solubility of the protein would be time consuming. We'd need more employees specialized to do this job which means more time and funds being used, and this is also not practical on a large-scale proteomics (proteomics is a large-scale study of proteins). There is a great interest in knowing the stability and solubility of protein in large-scale proteomics projects, and to identify mutations likely to increase stability and solubility. Knowing that there is a more efficient way of finding stability and solubility would make the old/current methods used by scientific researches a waste of time and resources. The more efficient method this project proposed is to use text mining to properly extract stability and solubility mentions of proteins from a corpus. There are factors that may affect the process of stability and solubility, by using these factors inside our code we determine the direction of protein stability [4] or protein solubility. The stability or solubility either is positive, negative or has no effect. This will be important to save both laboratory experimental effort [5] and financial resources. To attempt to extract protein stability and solubility we will be text mining over biological corpora which has stability and solubility mentions. This method of text mining could be used for much more than just extracting stability and solubility mentions, the extracting of stability and solubility mentions may be used to further improve stability and solubility prediction through machine learning [1][7][8]. This can be done by using the extracted stability and solubility mentions as the training data to train an algorithm.

Keywords: Solubility · Soluble · Stability · Stable · Protein · GATE · Text Mining · Extraction · Mutation · JAPE Rules · gazetteers · Corpus.

Introduction

Biomedical and biotechnological applications find use in stable and soluble proteins. We do this by text mining for mentions of stability and solubility and the use of this method is becoming increasingly more important for systems biology, genomics and genotype-phenotype studies specifically. Text mining for stability and solubility mentions makes other activities easier including the development of tools to eventually possibly predict stability and solubility based on machine learning [1][7][8], the modelling of cell signalling pathways and protein structure annotation. The usefulness of text mining for stability and solubility mentions can range from simply tagging any mentions of stability and solubility to interpretation of the consequences of stability and solubility on proteins [3]. The demand for stability and solubility text mining software from certain research scientists who approached Dr. Baker has led to this project, in this project I will talk about my journey to extracting stability and solubility mentions from scientific papers and explain why this software can benefit research scientists.

Motivation

Dr. Baker who works with research scientists from all over the world, and approached me with a project which could benefit these research scientists. These research scientists work with proteins but only certain stable and soluble proteins are suitable for development, which means that it would be important to know the protein's stability and solubility and what factors positively or negatively affect these proteins. Factors such as high temperature, pH, protein concentration, certain mutations etc. can affect the process of solubility and stability.

In the abstract I explained the reason why it is important to know the protein stability or solubility, and we want to avoid the protein being poorly stable, misfolded, and/or aggregated [1]. This way we have a higher chance to not negatively impact the health of the cell. This can lead to prevent huge economic losses in the biotechnology and bio-pharmaceutical industries. Therefore, finding out the stability or solubility of protein worked with may allow for many resources to be saved, this is where this project comes in.

Research scientists already have methods to find out the stability and solubility, but as of now finding stable protein adds to the cost and time of finding new drugs. This is because research scientists and drug companies need to experiment on proteins in order to find it's stability or solubility [5]. Some labs have research scientists heat up proteins to see if it does what it is supposed to do, and when it's stable until 28 degrees, it is noted in the scientific paper as stable until 28 degrees. This is just an example, but a realistic example that would be cost effective and not time efficient for it to be done on a large-scale. Therefore, research scientists and drug companies care about the ability to text mine scientific papers in order to extract stability and solubility mentions. This project is for the research scientists and the motivation is to save them on both time and money.

Problem Statement

Low protein solubility is often reported to be behind many human diseases. Understanding the effects of mutations on protein solubility can therefore help us understand the mechanisms connected with the development of human diseases. These effects of mutations on protein solubility would be easier found by text mining over a large group of scientific papers, this will allow us to be able to go over a large corpus (corpus meaning a collection of written texts) and easily find the effects of mutations on protein solubility in different scientific papers. Therefore, the reason why this project is of interest is because we introduce an efficient method to accurately extract stability and solubility mentions. The method we will be using is to text mine over scientific papers looking for keywords on stability and solubility. This will be important to save both laboratory experimental effort [5] and financial resources.

Some of the known challenges are that the accuracy to extract stability and solubility mentions of the existing protein engineering tools is often not where it needs to be due to limited stability and solubility experimental data available for training and testing. Published data containing mentions of solubility and or stability upon mutation were not easy to find, but there are currently certain databases collecting data to solve this. One of the data bases is called SoluProtMutDB [6], which is a comprehensive, manually curated database of the protein solubility data. They have a claim to 120 experiments, and right on their site you can see the mutations and whether their solubility has a positive or negative effect. SoluProtMutDB [6] also has other tools, one of them called SoluProt from the Loschmidt Laboratories [7]. They are focused on predicting changes to protein stability and solubility due to mutations since they have both bioinformatics and experimental expertise in those domains [7]. Making predictions based on machine learning [1] requires a large amount of data to train, since they currently only have 1000-2000 mutations [7], they do not have enough but the SoluProtMutDB tool is supposed to work towards solving this issue [6].

The unresolved issues/open issue is exporting annotations from GATE [2][4] into Java [9]. GATE stands for General Architecture for Text Engineering [2], it is a software used for text mining and can read over documents and annotate those documents. Once I am able to export the annotations from GATE into Java [4], the program will be able to print out the desired results. Another unresolved issue is that I am currently focused on extracting the stability mentions in the code, but the goal is to also extract solubility mentions. These two should be hand in hand as they would use identical algorithms. So, once I am able to extract stability mentions, I will probably be able to extract solubility mentions soon after.

Research scientists and drug companies will benefit from the outcome. I previously explained the reason, I gave an example of an experiment with heating up a protein until it is not stable anymore at 28 degrees and this is just not scale able. This experiment [5] made it clear that text mining to extract stability (and solubility) mentions would be much more cost efficient and time efficient then

to experiment on each protein, which is why these research scientist and drug companies would benefit from the outcome.

Goals

My ultimate goal is to introduce a system that can analyse scientific documents for relevant information and accurately extract mentions of stability and solubility.

To break this down, I had a few smaller goals. Since GATE [2][4] was a completely new software, the first goal was to complete a tutorial and play with it to figure out how to test and run JAPE [14] rules (Designed by Shawn Alexander Kroetsch) inside the software. These JAPE rules are crucial because they contain the rules on how words will be tagged during the text mining; an example of how these rules work is if you have self designed stability JAPE rules, and GATE finds the word stable inside a corpus, your rules may annotate this word along with words that may have an impact on the the word stable (depending on how you designed your JAPE rules). Words like more, less or not would have an impact, so the words "more stable" may be annotated. Another goal for the project was to be able to add these JAPE rules in the Java [9] code which was done in the Pipeline.java class. I also had to do this whilst customizing a large existing script that was already able to output a significant amount of information. The final smaller goal is what would lead to a successful project. I had to extract, save and write what is being tagged by the JAPE rules onto a CSV file.

Reproducing and assessing results from previous work was a major help on my project. Even if I had to customize a large existing script, that also meant that the large script already had an output on certain data. I could use the existing code that gives the output at the time to guide me while programming my additions. This would allow me to get closer to my goal.

Dogma

There is a requirement of having a good experience in Java [9] programming. Good experience would help in being able to understand what is going on in the 35+ Java classes where some of these classes contain over 3000 lines of code making it difficult for a beginner to know where to start. Knowledge on CSV files may help since the results are written to CSV files. It would also help if you're able to grasp some understanding of the GATE [4] Java-doc API [11], this will help you understand the objects and methods already used in the program. This is something I ignored and I paid for it. Reading the Java-doc API when you don't think you understand anything related to GATE in the java code will put you in a better position. This may just be advice for any programming done.

You also need to understand how the GATE [4] software works. Knowing how GATE works may help you test some of your results directly in GATE, if you are text mining, and you're not sure if your Java [9] file did a good job (there

may have been different variations that your program missed because you did not get the necessary features) then you can go directly into GATE to verify.

Common knowledge about protein stability and solubility. Protein is soluble if it prefers to interact with water, rather than with other protein molecules. Protein is stable if it is properly folded.

Core Methodology

You can follow the applicable diagram below called **Fig. 1** with this explanation. I first got comfortable with the JAPE [14] rules and gazetteers (gazetteers are where we put the actual words that we want to be annotated according to the JAPE rules, kind of like a dictionary) [15] through the GATE software itself. Then I add these JAPE rules and gazetteers into the Pipeline.java class which basically makes it that the classes connected to it now has access to the annotations with the correct imports. The DocumentInformation.java class then takes this opportunity to get the annotations and save them into classes that are contained in the mutation.jar file. The Corpus.java file then retrieves these saved annotations and prints them to a csv file which all happens when the pipeline runs.

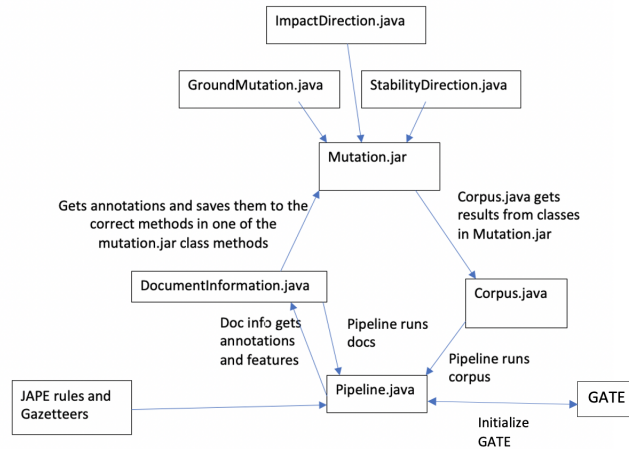


Fig. 1. A visual of a summary diagram of the technicality I worked with

Following the technical steps I took, I first got comfortable with how to use GATE [2][4] and ran the JAPE [14] rules to test some of the results that I'll be looking for. The basic things to know about getting comfortable with GATE is that when you open the software, there are ready made applications that come with GATE such as ANNIE. An application consists of any number of PRs (PR stands for Processing resources which are tools that process and annotate text), run sequentially over a corpus of documents. Once you activate ANNIE, you're able to add multiple JAPE [14] rules through the transducer, the transducer can be accessed due to ANNIE. Once you've added the JAPE rules you wanted to add the next step is to load a corpus into the program. Once the corpus is loaded, you want to run the JAPE rules on the corpus. When running the JAPE rules on the corpus is done, you're able to click on any of the documents in the corpus and when you click on Annotation Sets and Annotations List you will be able to select what rules you want to be tagged. In **Fig. 2** you see an image of what the GATE software looks like after going through the process that I just explained. I ran the stability direction JAPE rules on a stability corpus (stability corpus is just a collection of written texts on stability). In **Fig. 2** you can see some of the tagged words that appeared from selecting the STabilityDirection Annotation Sets.

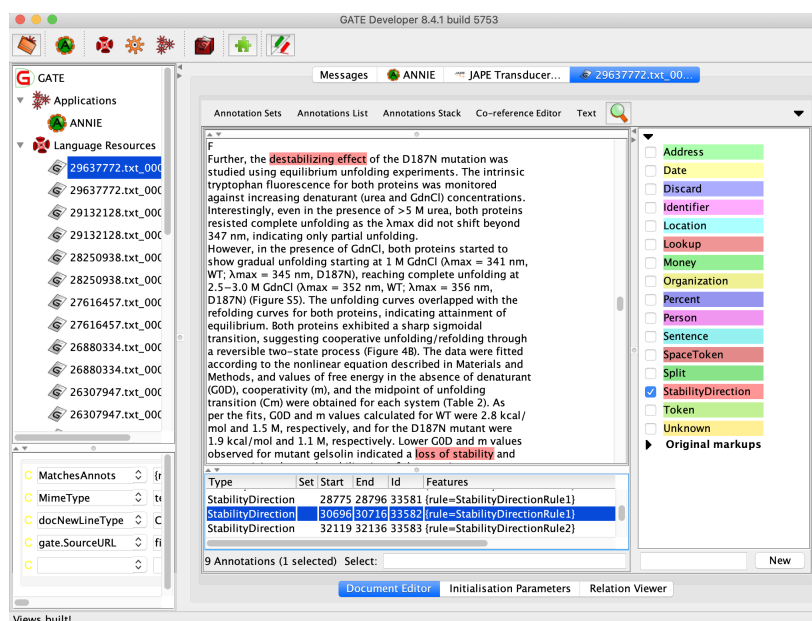


Fig. 2. An image of the GATE software in action

Next, I added the JAPE [14] rules and gazetteers [15] in the Pipeline.java class. **Fig. 3** Shows a yellow line in the middle because I combined two images. The first image above the yellow line show me adding the stability value and stability direction JAPE rules into the pipeline from the correct folder location. This is the part where you must be familiar with the GATE Java-doc API [11]. We use a FeatureMap object which represents the content of an annotation. We then put the JAPE rules into the FeatureMap and add this to the Annie Controller which runs all the analysers in turn over each of the documents in the corpus. The second picture below the yellow line follows similar steps, but the biggest difference is that instead of it putting the JAPE rules, it puts the gazetteer and there are some more put methods to control the formatting of the gazetteer.

```

271     if(runSv){
272         // Add stability value.
273         FeatureMap iT3Params = Factory.newFeatureMap();
274         iT3Params.put("grammarURL", this.getClass().getClassLoader().getResource( name: "gate/japerules/stabilityValues.jape"));
275         ProcessingResource iT3Pr = (ProcessingResource) Factory.createResource( resourceName: "gate.creole.ANNIETransducer", iT3Params);
276         annieController.add(iT3Pr);
277         log.info("stability value added");
278     }
279
280     if(runSd){
281         // add stability direction.
282         FeatureMap iT4Params = Factory.newFeatureMap();
283         iT4Params.put("grammarURL", this.getClass().getClassLoader().getResource( name: "gate/japerules/stabilityDirection.jape"));
284         ProcessingResource iT4Pr = (ProcessingResource) Factory.createResource( resourceName: "gate.creole.ANNIETransducer", iT4Params);
285         annieController.add(iT4Pr);
286         log.info("stability direction added");
287     }
288
289     // runSdExtractor
290     if(runSdExtractor){
291         // Add Gazetteer: Stability Direction.
292         log.info("Stability Direction gazetteer adding...");
293
294         // File oDir = new File(log.getClass().getResource("gate/gazetteers/organisms/").toURI());
295
296         // File oDir = new File(this.getClass().getClassLoader().getResource("/gate/gazetteers/organisms/").toURI());
297         FeatureMap ohParams = Factory.newFeatureMap();
298         ohParams.put("caseSensitive", false);
299         ohParams.put("encoding", "UTF-8");
300         ohParams.put("gazetteerFeatureSeparator", null);
301         //oParams.put("listsURL", new URL(oDir.toURI().toURL().toString() + "lists.def"));
302
303
304         ohParams.put("listsURL", new URL(log.getClass().getResource( name: "/gate/gazetteers/stability/lists.def").toURI().toString()));
305         ProcessingResource ohPr = (ProcessingResource) Factory.createResource( resourceName: "gate.creole.gazetteer.DefaultGazetteer", ohParams);
306         annieController.add(ohPr);
307         log.info("Stability Direction gazetteer added");
308     }
309 }

```

Fig. 3. An image of adding the JAPE rules and Gazetteer into the Pipeline.java class

The final step is where I had trouble. We wanted to be able to extract, save and write what's being tagged onto a CSV file. Just like **Fig. 3**, I also split two pictures by a yellow line in **Fig. 4**. In the picture above the yellow line you can see that the for loop writes all the content we get to the CSV file. This is where I added the "getStabilityDirection()" method, this method is in a class called StabilityDirection.class in the Mutation.jar file. I will go into more detail explaining these files when I describe my process to get to my result. The picture below the line is where it got a bit more complicated, I basically attempted to extract the annotation features and save them so that they could be written to a CSV file. This attempt went wrong and I will explain in the result section why I think this went wrong.

```

174 for (Entry<String, DocumentInformation> e : docInfo.entrySet()) {
175     DocumentInformation docInfo = e.getValue();
176     csvWriter.writeNext(new String[]{" "});
177     //System.err.println(docInfo);
178     if(docInfo.getAllGroundedMutations()!=null && docInfo.getAllGroundedMutations().size()>0){
179         for(GroundedMutation gm : docInfo.getAllGroundedMutations()){
180             GroundedPointMutation gpm = gm.getCompoundMutation().next();
181             int offset = gpm.getMentionedPosition()-gpm.getCorrectPosition();
182             String[] csvLine = {
183                 docInfo.getName(),
184                 gm.getProteinAppliedTo().getSwissProtId(),
185                 Character.toString(gm.getCompoundMutation().next().getWildtypeResidue()),
186                 Integer.toString(gm.getCompoundMutation().next().getMentionedPosition()),
187                 Character.toString(gm.getCompoundMutation().next().getMutantResidue()),
188                 Integer.toString(offset),
189                 gm.getMutationImpacts().next().getAffectedProteinProperty().toString().split(" ")[1].replaceFirst("regex:", "00:");
190                 gm.getMutationImpacts().next().getDirectionAsString(),
191                 gm.getStabilityDirection().getStabilityDirection(),
192                 "Stability Value",
193             };
194             csvWriter.writeNext(csvLine);
195         }
196     }else{
197         log.info("No results of Mutation Impact Extraction.");
198     }
199 }
200
201 AnnotationSet stabilityTempSet = doc.getAnnotations().get("StabilityDirection"); //a
202 for (Annotation stability : stabilityTempSet) { //a
203     String stabilityString = (String)stability.getFeatures().get("StabilityDirectionRule1");
204     if(stabilityString==null)continue;
205     String[] parsed = parseNNFormat(stabilityString);
206     if(!stabilityString.contains(stabilityString)){
207         GroundedPointMutation gm = new GroundedPointMutation();
208         gm.setWildtypeResidue(parsed[0].charAt(0));
209         gm.setMentionedPosition(Integer.parseInt(parsed[1]));
210         gm.setMutantResidue(parsed[2].charAt(0));
211         gmResult.add(gm);
212
213         FeatureMap stabilityDirectionAnnFeatures = stability.getFeatures();
214
215         int stabilityDirectionAnnId = (Integer) stabilityDirectionAnnFeatures.get("StabilityDirectionRule1");
216
217         Annotation stabilityDirectionAnn = doc.getAnnotations().get(stabilityDirectionAnnId);
218         stabilityDirectionAnnFeatures = stabilityDirectionAnn.getFeatures();
219
220         String sDirection = (String)stabilityDirectionAnnFeatures.get("StabilityDirectionRule1"); //a
221
222         StabilityDirection sDirection2 = new StabilityDirection(sDirection); //a
223
224         GroundedMutation groundedMutation = new GroundedMutation();
225         groundedMutation.setStabilityDirection(sDirection2); //a
226     }
227 }
228 }

```

Fig. 4. An image of working on the final step of my goal

Resources

My personal hardware used is a 2016 MacBook Pro. This is not one of the best devices to use today but it delivers the solutions without problems. The hardware specifications for my specific device are: the Mac OS Catalina Version 10.15.7 operating system, a 2.3 GHz Dual-Core Intel Core i5 processor, 8 GB of RAM and an Intel Iris Plus Graphics 640 1536 MB graphics card.

A third party software used is called GATE [2]. GATE is a software that allows for you to text mine over a corpus using specific JAPE [14] rules [2] attached, which determines the words to look for and annotate. Another third party software would be Java [9]. Java is a popular programming language which I will be using alongside GATE in order to text mine and extract stability and solubility mentions. The IDE's that I used are IntelliJ [12] and Visual Studio Code [13]. IDE stands for Integrated Development Environment, and is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools and a debugger. I used the Visual Studio Code IDE to easily search for specific code because I am more comfortable with this IDE and I also used it to create a few classes, this is my default IDE. I used the IntelliJ IDE to run the program as the README required me to follow steps that I could only find in the IntelliJ IDE. Another third party is CSV. The Java programming was used to write results to CSV files.

For knowledge resources I searched articles online and links sent by Dr. Baker. I also had calls with Dr. Baker where he would help me past obstacles. Another person who would help me with obstacles is Mohammed Sadnan Al Manir. Lastly I also had help from Shawn Alexander Kroetsch who provided me with the preexisting script, every corpus I have access to and the JAPE [14] rules.

Describe your Results

Unfortunately, I was not able to get the desired results for the Stability Direction. As you can see in **Fig. 4**, there is a highlighted Stability Direction column. The output in this column comes from a call to a method that I used in the Corpus.java class, I showed a picture of this call in the Core Methodology section. The line writes `gm.getStabilityDirection().getStabilityDirection();`, to explain this line in detail, `gm` is a variable for the `GroundedMutation` object. `GroundedMutation` is an object of the `GroundedMutation.java` class, in this class we have multiple methods that call to other methods from classes that have the annotations saved. The first "`getStabilityDirection()`" method gives me access to the `StabilityDirection.java` class where I have a method saved with the same method name (not a great programming idea in hind sight) which I then directly call. This second "`getStabilityDirection()`" method has the Mutation Impact annotations saved and will print less stable, more stable or no effect depending on the annotation extracted. You may notice from that sentence what is wrong with my project, I extracted Mutation Impact annotations instead of Stability Direction annotations meaning that my results are based on the Mutation Impact annotations. The reason I did this was to test the classes I created and see if they are able to print anything before I directly print the desired results, the issue I came across after testing this was that based on my annotation code I was not able to print anything directly based on the Stability Direction and I was not able to fix my code of extracting and saving annotations in time. I did get some positive results where my JAPE [14] rules and gazetteers [15] successfully ran, and although the annotations results that I printed were for Mutation Impact direction, I was able to extract save and print results to the CSV file based on annotations which is a positive result considering that is the main goal.

I learned that I should approach bigger projects more targeted in the future. I also learned that my main goal was probably not achieved in time because of my lack of understanding of the Java-doc GATE API [11]. The part where I did not succeed is visible in my core methodology where I attempted to extract annotations and save them. I attempted this in the `DocumentInformation.java` class, I choose this class because this is the class where all other annotations were extracted. My theory on why my result is empty when I try to run my program based on these annotations is that it is not able to find the annotations.

document_id	uniprot_id	wt_residue	position	m_residue	coordinate_offset	property_id	direction	text	stability_direction	stability_value
10100638.xml.00008	P51698	E	244	Q	0	GO:0004806	[+]	Less Stable	Stability Value	
10100638.xml.00008	P51698	E	244	Q	0	GO:0004806	[+]	Less Stable	Stability Value	
10100638.xml.00008	P51698	D	108	N	0	GO:0004806	[0]	No Effect	Stability Value	
10100638.xml.00008	P51698	D	108	N	0	GO:0004806	[0]	No Effect	Stability Value	
10508409.xml.00009	P22643	T	197	A	0	P11111_0	[+]	Less Stable	Stability Value	
10508409.xml.00009	P22643	T	197	A	0	CatalyticRateConstant	[+]	Less Stable	Stability Value	
10508409.xml.00009	P22643	F	294	A	0	P11111_0	[+]	Less Stable	Stability Value	
10508409.xml.00009	P22643	F	294	A	0	MichaelisConstant	[0]	Less Stable	Stability Value	
10508409.xml.00009	P22643	F	294	A	0	CatalyticRateConstant	[+]	Less Stable	Stability Value	
10508409.xml.00009	P22643	T	197	A	0	MichaelisConstant	[0]	Less Stable	Stability Value	
10545279.xml.0000A	P22643	W	109	L	50	GO:0018786	[+]	Less Stable	Stability Value	
10545279.xml.0000A	P22643	W	109	L	50	GO:0018786	[+]	Less Stable	Stability Value	
10545279.xml.0000A	P22643	F	151	W	50	GO:0018786	[0]	No Effect	Stability Value	
10545279.xml.0000A	P22643	W	109	L	50	GO:0018786	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	D	260	N	107	MichaelisConstant	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	A	149	T	107	EnzymeEfficiencyConstant	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	A	149	S	107	GO:0004817	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	A	149	T	107	MichaelisConstant	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	A	149	T	107	CatalyticRateConstant	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	D	260	N	107	CatalyticRateConstant	[+]	Less Stable	Stability Value	
10579523.xml.0000B	A71PJD	A	149	T	107	CatalyticRateConstant	[+]	Less Stable	Stability Value	
1063662.xml.0000C	O35543	W	175	Q	-15	MichaelisConstant	[+]	Less Stable	Stability Value	
1063662.xml.0000C	O35543	W	125	Q	-15	MichaelisConstant	[+]	Less Stable	Stability Value	

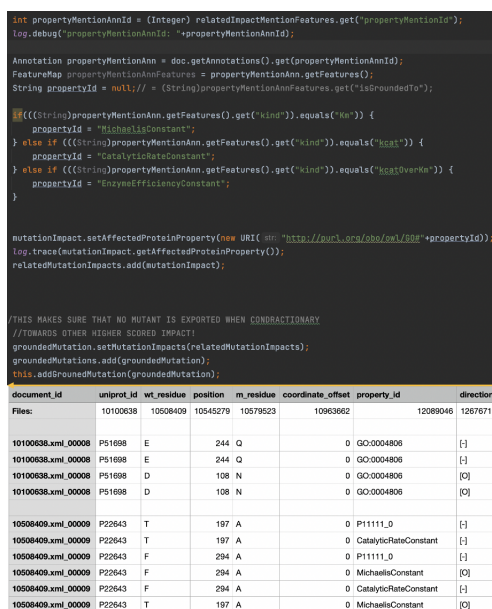
Fig. 5. CSV file results

Samples and path to Result

I created this section to explain my process to get to my results in detail. First there's need for understanding of what existed before I even touched the code, I will explain how I came to understanding of this code and then I will explain what code I added and why I did.

The **Fig. 1** diagram in the Core Methodology explains the main files that I worked with. These files are where I could comprehend what was going on based on what was programmed and what the output was. Starting with the Corpus.java file **Fig. 4** above the yellow line, it stands out because it had the CSV writer, meaning that all the output is coming directly from that file. There are two output files created, the MG (Mutation Grounded) [10] and the MIE (Mutation Impact Extraction), for the sake of my results we will only be looking at the MIE output. Now that we have something to work with, we look at what is being written to the file. We come to find out that those method calls are in a Java [9] class called GroundedMutation.java which is in the Mutation.jar folder as a .class file. This folder contained multiple classes with methods that would be called in the Corpus.java file to get the CSV file to display the results, now we need to find out where these methods are getting the annotations from. I figured out correlations (see **Fig. 6**) between the output and code in the DocumentInformation.java file, the output below the yellow line has a property_id column that shares some results with what is put in the DocumentInformation.java file. This alone is not clear evidence, but there were many more instances in the

DocumentInformation.java file that led me to suspect it to be the file where annotations were extracted. This file also uses multiple Annotation and FeatureMap objects which I will explain when I get to my own code that I added. The current most important clue I was looking for was where the GroundedMutation.java class got the annotations from, and **Fig. 6** just above the line clearly shows the grounded mutation variable adding the annotations. Now we know where to extract annotations (DocumentInformation.java), we know where to save them (in one of the Mutation.jar classes), and we know to extract them inside the Corpus.java class. Only one thing is missing, we need to add the JAPE [14] rules and gazetteer [15] into the program so we are able to access the annotations. The Pipeline.java had JAPE rules and gazetteers, so it would only make sense to add mine there. This is the existing Script that I had to work with and now my own work comes in.



```

int propertyMentionAnnId = (Integer) relatedImpactMentionFeatures.get("propertyMentionId");
log.debug("propertyMentionAnnId: " + propertyMentionAnnId);

Annotation propertyMentionAnn = doc.getAnnotations().get(propertyMentionAnnId);
FeatureMap propertyMentionAnnFeatures = propertyMentionAnn.getFeatures();
String propertyId = null; // = (String)propertyMentionAnnFeatures.get("isGroundedTo");

if (((String)propertyMentionAnn.getFeatures().get("kind")).equals("ke")) {
    propertyId = "MichaelisConstant";
} else if (((String)propertyMentionAnn.getFeatures().get("kind")).equals("scat")) {
    propertyId = "CatalyticRateConstant";
} else if (((String)propertyMentionAnn.getFeatures().get("kind")).equals("kcatOverKm")) {
    propertyId = "EnzymeEfficiencyConstant";
}

mutationImpact.setAffectedProteinProperty(new URI("http://pubchem.org/chem/508" + propertyId));
log.trace(mutationImpact.getAffectedProteinProperty());
relatedMutationImpacts.add(mutationImpact);

//THIS MAKES SURE THAT NO MUTANT IS EXPORTED WHEN CONTRADICTIONARY
//TOWARDS OTHER HIGHER SCORED IMPACT!
groundedMutation.setMutationImpacts(relatedMutationImpacts);
groundedMutations.add(groundedMutation);
this.addGroundedMutation(groundedMutation);

```

document_id	uniprot_id	wt_residue	position	m_residue	coordinate_offset	property_id	direction
Files:	10100638	10508409	10545279	10579523	10963662		12089046 12676719
10100638.xml_00008	P51698	E	244	Q	0	GO:0004806	[-]
10100638.xml_00008	P51698	E	244	Q	0	GO:0004806	[-]
10100638.xml_00008	P51698	D	108	N	0	GO:0004806	[C]
10100638.xml_00008	P51698	D	108	N	0	GO:0004806	[C]
10508409.xml_00009	P22643	T	197	A	0	P11111_0	[-]
10508409.xml_00009	P22643	T	197	A	0	CatalyticRateConstant	[-]
10508409.xml_00009	P22643	F	294	A	0	P11111_0	[-]
10508409.xml_00009	P22643	F	294	A	0	MichaelisConstant	[C]
10508409.xml_00009	P22643	F	294	A	0	CatalyticRateConstant	[-]
10508409.xml_00009	P22643	T	197	A	0	MichaelisConstant	[C]

Fig. 6. Existing Script

I started with adding the JAPE [14] rules and the gazetteers [15] into the Pipeline.java class. As you can see from **Fig. 3**, this was successfully done. I explained the code in detail in the Core Methodology section. The next step was to add classes into the Mutation.jar folder so that when I was to extract annotations, I had a place to save them. It's important to note that the Mutation.jar file only had .class files so I had to figure out how to add files and edit the existing ones. Luckily Dr. Baker had previously sent me a Mutation folder with .java files that were identical to the .class files, so for my solution I used

this folder to add and edit classes. I'd compile all the files inside the Mutation folder that Dr. Baker previously shared with me, and then use the command line to empty the Mutation.jar folder and add the .class files that I edited back into the Mutation.jar folder. I'd have a Mutation.jar copy on the side just in case my changes negatively impacted the code. I started with a class named StabilityDirection.java, Below is some pseudo code of how the class looked.

```
Public class StabilityDirection
  constructor
  setStabilityDirection[
    if negative: lessStable = true
    if positive: moreStable = true
    else no effect = true ]
  getStabilityDirection[
    if moreStable: return "More Stable"
    if lessStable: return "Less Stable"
    else: return "No Effect" ]
```

The StabilityDirection class has more methods but these are the main methods that I was working with. This forced me to add an object of the Stability Direction class inside the Grounded Mutation class like all the other classes in the Mutation.jar folder had done. This will make it so that when I make a Grounded Mutation object variable, then I am able to call the Stability Direction. An example of this is in **Fig. 4** in the picture above the line, `gm.getStabilityDirection.getStabilityDirection()`. `gm` is a variable of the Grounded Mutation object, and the first `getStabilityDirection()` is a method inside the Grounded Mutation class which then gives access to the StabilityDirection.java class. I also created a StabDirection.java class which simply contained the more stable, less stable and no effect strings that you see in StabilityDirection column in the result on **Fig. 5**. It's important to know that there were other parts in the code that needed minor editing like initialization, imports etc. for the code to run properly without errors. The reason I don't include pictures or much explanations of these minor edits is because besides impacting the code's ability to run, they are not of importance and do not require much thought. I rather explain the parts in the code that required more complex thinking. Now that we are equipped with a place to save the extracted annotations (the StabilityDirection.java class), it's time to go over how I tried to extract the annotations. For this part you can follow along below the yellow line on **Fig. 4**. I will be using definitions directly from the Java-doc GATE API [11]. The first line is where I create an AnnotationSet object which I give the variable name `stabilityTempSet`. Annotation set is a set of annotations on a document. Annotation sets are attached to documents - they cannot be constructed directly, but are obtained via the `getAnnotations` methods of Document. Which is why I did a `getAnnotations` to get the StabilityDirection JAPE [14] rule. The next line is a for each loop to get every annotation from the annotation set. An Annotation is an arc in an AnnotationSet. It is immutable, to avoid the situation where each annotation has to point to its parent graph in order to tell it to update its

indices when it changes. on the next line I would use a `getFeature()` with the specific rule I want to receive and translate it to a `String`. `getFeatures` just gets the feature set which (if you look at **Fig. 2**) is just the what rule the annotation is based on. We can skip some lines that require basic Java [9] knowledge over to line 188. On this line I used the `FeatureMap` which I already described when I explained **Fig. 3**. The next few lines (190, 192 and 193) create an ID and store it in an `Annotation` object where we then use the `getFeatures` method on to store it in the `FeatureMap` object variable called `StabilityDirectionAnnFeatures`. I then extract the first feature for the first rule and store it into a `String` that I then save in the `StabilityDirection` class that I created. I then set the `StabilityDirection` `setStabilityDirection` method to the annotation I attempted to extract but when I run the program based on these annotations, I end up with an empty output. I have tried many solutions with this code, therefore it may look confusing but I haven't had any desired output yet. In this specific attempt I suspect that it went wrong when I did not add the `GroundedMutation` to the `GroundedMutations` set like it was done in **Fig. 6** just above the yellow line.

Updated GANTT Chart

Fig. 7 shows my schedule of the semester through a gantt chart. The chart is an approximation as accurate as possible, for example if I just missed 1 hour and worked for 50 minutes instead, I'd write down 1 hour. The same is true for the opposite, if I studied for 1 hour and 10 minutes, I'd write down 1 hour on the gantt chart. The top left of the gantt chart has a legend explaining the colors in the gantt chart. I was 8 hours shy of spending 200 hours on this course and 76 of those hours came from July alone, it is too bad that the results do not show for it.

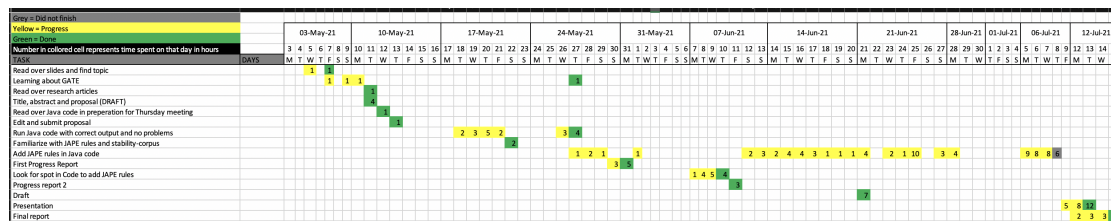


Fig. 7. GANTT chart July 16, 2021.

Evaluation

I evaluate my work on results first. Besides the fact that I did not succeed the ultimate goal of being able to extract stability and solubility mentions from

scientific papers, I have multiple positive results. I was able to incorporate the new JAPE [14] rules and gazetteer [15] into the program successfully. I was also able to print results to the CSV file based on annotations from JAPE rules, they were just not the correct annotations. I also evaluate my work on improvement, time and effort. Looking at my gantt chart, it is obvious that I started off by putting little effort into this course. Through time I picked up my slack and did better towards the end. I am unhappy with how I started but happy with how I finished it when it so I made great improvement. I also evaluate myself on difficulty of the challenge and if I learned something new. This is my first time working on a project with over 35 files and some of them containing over 3000 lines of code, so it seemed overwhelming at first. Once I knew which classes I had to work with, I got more comfortable with the code. This led to me accomplishing the majority of my goals and learn how to approach a big project without being so overwhelmed in the future.

Future

This is the easy part for me since the final goal of the project is yet to be accomplished. I was successfully able to integrate the JAPE [14] rules and gazetteers [15] into Java, but the results were not there because I did not properly extract and save the annotations.

The most important design improvement currently is to get the proper results for the stability directions printed on the CSV file.

Some additional data sets are still needed. My goal is to text mine over stability and also solubility data, so additional data sets would be solubility ones. After successfully getting stability direction results, the rest should be easy by following the identical steps.

The people who should know about these results once they are as desired are research scientists who work with soluble and stable proteins.

Conclusion

The finished project result could mean a lot for the research in science and pharmaceutical area. It would be less cost effective and less time efficient than their current methods to do this on a large-scale. Finishing this application would mean that research scientists and drug companies would be able to text mine scientific papers to extract stability and solubility mentions on large corpus.

The contributions made by me to the state-of-the-art is that when the program is ran, we get confirmation that the JAPE [14] rules and gazetteer [15] are added successfully. There's a new column in the result CSV for stability, this Stability Direction column currently produces an output.

My prototype is premature and not ready for use until we can at least extract the stability directions and save them so that we can write it to the CSV file to see the desired results. This was the desired main goal.

Critical Commentary

I learned how to approach bigger projects. I started by reading every file one by one, and when you've got over 35 files and some over 3000 lines of code, this can be time consuming when there is not much time. Instead, in the future I'd search for the file that has the content I need and the files that directly interact with that file. This would help me get straight to the point. Watching back over the gantt chart I also learned to do the hard work as soon as I can because leaving it for later is gonna end up in me doing it at some point regardless with less time. There is no point to leave work for later. Setting daily goals and forcing myself to finish them is what let me to a significant increase of hours in the gantt chart, and I will take this lesson with me in the real world.

I worked with the GATE [2] the software tool to text mine over documents. I worked with the Java [9] programming language with a goal in mind to do what the GATE software can do in text mining over documents but also extracting and saving the annotations. I also worked with CSV files where I wrote the results to. These were my main tools in the project.

I designed customized outputs (new Stability Direction tab in the CSV file); we just need to figure out how to make the correct input work so that we can extract the correct annotations and print results based on the actual Stability Directions. The methods I followed were in the already written script. There were examples on how to extract the annotations and write to the CSV file, I simply attempted to follow them.

To get started and advance on this project, you need the skills mentioned in my DOGMA section which are: Java programming experience since there is a lot of understanding to go through with 35+ files and some of them containing over 3000 lines of code. Knowledge on CSV is also necessary as we write the results to a CSV file, and you need to grasp understanding of the GATE java-doc API and of how the GATE [2] software works.

Updated References

1. Damborsky, Jiri (2019/02/01). Computational Design of Stable and Soluble Biocatalysts. *ACS Catalysis*, 9, 1033-1054. doi: 10.1021/acscatal.8b03613
2. Cunningham, et al. Developing Language Processing Components with GATE Version 8. University of Sheffield Department of Computer Science. 17 November 2014.
3. Klein, Artjom et al. "Benchmarking infrastructure for mutation text mining." *Journal of biomedical semantics* vol. 5,1 11. 25 Feb. 2014, doi:10.1186/2041-1480-5-11
4. H. Cunningham, V. Tablan, A. Roberts, K. Bontcheva (2013) Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open-Source Text Analytics. *PLoS Comput Biol* 9(2): e1002854. doi:10.1371/journal.pcbi.1002854 — <http://tinyurl.com/gate-life-sci/>
5. Man, T. P. (2018, September 18). Methods of Determining Protein Stability. <https://info.gbiosciences.com/blog/methods-of-determining-protein-stability>.
6. SoluProtMutDB. (n.d.). <https://loschmidt.chemi.muni.cz/soluprotmutdb/>
7. Jiri Hon, Loschmidt Laboratories. "About." About — SoluProt 1.0 – Prediction of Soluble Protein Expression in Escherichia Coli. Web. 16 July 2021. <https://loschmidt.chemi.muni.cz/soluprot/?page=about>
8. Han, Xi et al. "Improving protein solubility and activity by introducing small peptide tags designed with machine learning models." *Metabolic engineering communications* vol. 11 e00138. 22 Jun. 2020, doi:10.1016/j.mec.2020.e00138
9. Arnold, K., Gosling, J., Holmes, D. (2005). *The Java programming language*. Addison Wesley Professional.
10. Baker, Christopher Kanagasabai, Rajaraman. (2011). Extraction and Grounding of Protein Mutations via Semantic Integration of Text and Sequence Information. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*. 556-563. 10.1109/AINA.2011.112.
11. GATE JavaDoc, July 16, 2021 <https://gate.ac.uk/releases/gate-8.3-build5704-ALL/doc/javadoc/index.html>
12. "IntelliJ Idea: The Capable Ergonomic Java Ide by JetBrains." Web. 16 July 2021. <https://www.jetbrains.com/idea/>
13. "Visual Studio Code - Code Editing. Redefined." RSS. Microsoft, 14 Apr. 2016. Web. 16 July 2021. <https://code.visualstudio.com/>
14. Thakker, D., Osman, T., Lakin, P. "GATE JAPE Grammar Tutorial." Feb 27, 2009. <https://gate.ac.uk/sale/thakker-jape-tutorial/GATE>
15. "Sale/tao/splitch13.html." GATE. 10 Mar. 2021. Web. 16 July 2021. <https://gate.ac.uk/sale/tao/splitch13.html>