Naive WebMD Symptom Checker:

Team 4's Final Project Report

Introduction:

The goal of this project is to calculate and return a set of five relevant diseases to the user based on the symptoms that they include as a comma-separated query. The program shall process all symptoms and compare these symptoms to our data set, which is a bunch of diseases and a related list of symptoms. Using one of the various models, the program will return a list of five diseases that it deems relevant. Given our five relevant diseases, the program will further tell the user, which of the three categories, namely- "emergency" , "seek medical" or "wait", each disease falls under. The idea behind classifying the diseases is that some diseases like heart failure is an immediate emergency, whereas a patient suffering from e.g. depressive mental disorder, will require to seek medical attention periodically and someone suffering from e.g. dehydration does not need medical attention and can wait for the symptoms to dissipate. We can check the results via numerous evaluation metrics to see how close our models are compared to each other.

We want to tell the user what ailment they have, but without actual example cases, we cannot test if the ground truth that we arrive at is the correct one. Therefore, our aim is to make this program return ailments as close as we possibly can to the truth. Using a training and test data set, we can avoid overfitting.

We chose to use a baseline similarity of BM25, as this method was already implemented in Lucene and is easily applicable. For the classification part, we implemented a custom Naive Bayes classifier following the Bernoulli method. Bernoulli method was chosen over Multinomial method because unlike Multinomial, it generates indicators for each term in the vocabulary(1 if present in the document, 0 otherwise). This is more in line with the type of problem and data we are dealing with, where term frequency is not that relevant, as for a particular disease, a symptom occurs in the data set once.

In terms of the TF-IDF variants, term frequency isn't very useful as each symptom (or in the context of TF-IDF, each term) either appears zero or one times in the symptom list for each disease. However, a symptom could appear in more than one disease, so document frequency is a more suitable measurement to adjust. We decided on bnn.bnn, btn.btn, and bpn.bpn as the three variants to use as they all explore different ways to interact with the document frequency.

Each unigram language model variant that was touched on in Assignment 4 was added in our project as well. We felt like language models were a great fit for our project because there were not a lot of different 'marbles' in our 'urn'; the amount of symptoms was fairly limited in comparison to the entire English language. This allowed us to easily determine the values needed for different smoothing algorithms.

To train and test our custom classifier model, we first divided the diseases in our dataset into three categories - "emergency", "seekmedical" and "wait". We then took the first 70% of our dataset as our training set and the remaining 30% as our test set. Although, the diseases in each dataset are different but the three categories or classes in the dataset depends on the symptoms of the diseases and not on the name of the disease itself. Therefore, the training of our classification model is carried out using the symptoms of the diseases from the training set and the testing is also carried out using the symptoms of the diseases in the test set.

Related Work:

We're unsure how we can post related work for most of our methods, as they are from class material and notes. However, we can talk about certain resources we used in our project.

The program that we used to create this project is Apache Lucene (link: https://lucene.apache.org/). We have been using this program throughout the class in order to complete all of our programming assignments. Lucene is used to build search engines, which is exactly what our project is about. A full documentation can also be found online, explaining the different objects and methods found.

For a ground truth, we used a website called WebMD (link: https://www.webmd.com/) to compare. Since our program is a smaller version of WebMD, the symptom checker on the web site gives us a way to compare our results with a more professional product. For the five queries that we tested, we were able to mostly match our symptoms with symptoms on their websites. Also, our dataset did not have all of the diseases that WebMD takes into account, and vice versa. Even though there are some complications, we were able to find 3-5 relevant diseases for each query for our qrel file.

Finally, our data set came from this link: http://people.dbmi.columbia.edu/~friedma/Projects/DiseaseSymptomKB/index.html

We originally found this same dataset in Kaggle. However, the data set was in German. We attempted to use Google Translate and other translating methods at first, but we found the data set to be too large to do so efficiently. We luckily found the English version from their sources. The data set as it is in the link is in a very poor format for Lucene, so we overhauled its structure; more on data cleaning will be stated in the next topic.

Approach:

Lucene's BM25 was used as our "control" in our experiments. We found that it performed well enough for our task assuming the user inputted terms that existed in the corpus and that the user inputted enough symptoms to produce specific results. If the user inputs fewer symptoms, recall is high but precision is likely very low. A good example is the common illness symptom "vomiting". Many illnesses have this as a symptom from stomach flu to AIDS, but all of these results do not help the user determine the cause or severity of their illness. It performed better using the standard Lucene analyzer than using our custom analyzer because it was able to analyze each individual word in the query and corpus rather than forcing users to input terms with the exact correct spacing to return results.

To get a similarity score based on tf-idf, we implemented three different variants, bnn.bnn, btn.btn and bpn.bpn. As mentioned before, for both a query or a document(disease) in our dataset, a term(symptom) will occur only once and therefore, rather than considering term frequency, it is more relevant to consider it as a boolean. However, multiple documents(diseases) can have the same terms(symptoms). Due to this, in the case of document frequency, we considered three variants or scenarios- how does it affect our score when document frequency is not considered(bnn.bnn) and when considered, how does it affect if the variant used is an inverse document frequency(btn.btn) or a probability of inverse document frequency(bpn.bpn). For all of the three variants, to calculate the weight of the term weight from the query, for a given document, if a term occurs in a query or document then we assign it a weight of 1 else it is 0. For bnn.bnn, since the document frequency is not considered, it is always set as 1. In case of btn.btn the inverse document frequency(idf) for a particular term t is calculated as $t(idf) = logN/df(t)$, where N is the number of documents in the corpus and df(t) is the document frequency for the term t. For bpn.bpn, we calculated the probability of inverse document frequency of a term t as $prob\ of\ idf = max\{0, log[\ (N - df(t))\ /\ df(t)]\}$. Since no normalization was carried out the normalization value was always set as 1.

Language models were another method that we decided to implement in our program. The basic premise of a language model is that a disease is scored based on how likely it is to randomly pick a term from its list of symptoms, and the chosen term be a part of the query. The basic equation is the term frequency divided by the document length, for each term in the query. Each of these probabilities are multiplied together to get the final score. However, if a query term does not appear in the document, then the score becomes zero because we are multiplying the score by zero. To avoid this, we implemented three different types of smoothing functions. These make sure that we are never multiplying the score by zero. The three equations we used are listed below.

Unigram Language Model with Laplace Smoothing:

*Score = Score * ((termFrequency + 1) / (docLen + vocabLength));*

Unigram Language Model with Jelinik-Mercer Smoothing:

*Score = Score * (((0.9) * (termfrequency / docLen)) + ((0.1) * (relevantTermFrequency / relevantVocabLength));*

Unigram Language Model with Dirchelet Smoothing:

*Score = Score * ((termFrequency + (1000 * (relevantTermFrequency / relevantVocabLength)) / (docLen + 1000));*

For clarification, termFrequency is the term frequency of a term in a document, relevantTermFrequency is the term frequency of a term in all relevant documents, docLen is the document length in terms of tokens, and relevantVocabLength is the length of the vocabulary of all relevant documents. 1, 0.9, and 1000 are all constants we chose.

For our Naive Bayes classifier the Bernoulli method was used. To train and test the classifier, we created two separate methods. The train method has the vocabulary list (V) and the total number of diseases (N) from our training set. Now for each of the three classification categories (c), it gets the total number of diseases in each category (Nc) and the prior probability for each category is calculated as *prior[c] = Nc/N*. For each term (t) in our vocabulary list (V), it counts the number of documents under that particular class containing a term (Nct). The conditional probability of a term under a given class is calculated as *condpron[t][c] = (Nct + 1)/(Nc + 2)*. The test method contains our prior probability and conditional probability, which was calculated in train method. Now, given a list of diseases from our training dataset, for each disease in the list, it extracts it's corresponding list of symptoms from the dataset. For each category(c), it sets the score for that category as *score[c] = log(prior[c])*. For each term (t) is our vocabulary list (V), if t exists in vocabulary list (Vd) then *score[c] += log condprob[t][c]*, else *score[c] += log(1 - condprob[t][c])*. Once the score for all the categories have been calculated, we pick the category with the highest score.

Our dataset's lists of symptoms mostly contained the scientific terms for different symptoms (e.g. polydipsia instead of excess thirst), so we realized we needed to use thesaurus-based query expansion in order for regular users to get results from more common terms for symptoms. Probability-based query expansion was considered, but it was not used because the system trying to determine similar words based on probability in a raw list of terms would likely create estimations of symptoms that are incorrect and would produce incorrect results. Though it is faster to create an automatic thesaurus using co-occurrence, we couldn't do this with our particular corpus because the document texts were just raw lists of symptoms with no other words to gain context. We attempted to create a manual thesaurus, but doing this in Lucene was quite difficult, particularly with mapping one scientific term to a short phrase to

describe the word. It may have been more effective to alter the corpus to have less scientific terms, or add the non-scientific terms into a document along with the scientific terms. We also could have trained our system using another corpus of documents with more words other than symptoms such as medical documents using co-occurrence so the system could gain enough context to build a thesaurus that would result in increased recall in our system.

As we mentioned above, the data set that we acquired online was not in the correct format. All of the diseases and symptoms had a strange prefix, and not all of the symptoms were labelled with their corresponding diseases. A picture of the old data set can be found below.

Disease 2	Disease 2 2	Symptoms
UMLS:C0020538_hypertensive disease	Hy…	UMLS:C0008031_pain chest
		UMLS:C0392680_shortness of breath
		UMLS:C0012833_dizziness
		UMLS:C0004093_asthenia
		UMLS:C0085639_fall
		UMLS:C0039070_syncope
		UMLS:C0042571_vertigo
		UMLS:C0038990_sweat^UMLS:C0700590_sweating increased
		UMLS:C0030252_palpitation
		UMLS:C0027497_nausea
		UMLS:C0002962_angina pectoris
		UMLS:C0438716_pressure chest
UMLS:C0011847_diabetes	Dia…	UMLS:C0032617_polyuria
		UMLS:C0085602_polydypsia
		UMLS:C0392680_shortness of breath
		UMLS:C0008031_pain chest
		UMLS:C0004093_asthenia
		UMLS:C0027497_nausea
		UMLS:C0085619_orthopnea
		UMLS:C0034642_rale
		UMLS:C0038990_sweat^UMLS:C0700590_sweating increased
		UMLS:C0241526_unresponsiveness
		UMLS:C0856054_mental status changes
		UMLS:C0042571_vertigo
		UMLS:C0042963_vomiting
		UMLS:C0553668_labored breathing

In order to create a more suitable file to use, we utilized JMP's Recode platform. JMP is the statistical software we used to manipulate our data. Recoding let us change all matching cells in a column into a different string, and then put the new strings in a different column. For each of the 134 diseases, we got rid of the prefix and capitalized all of the words. We did that for over 500 symptoms as well, and put those in a different column as well. It took us one hour to recode the diseases, and three hours to recode the symptoms, because we had to do this manually.

While this solved our prefix problem, it did not solve our format issue. In order to solve that, we first copied the recoded diseases and sorted them alphabetically in a column of a new data table. Then, we copied all of the symptoms into a text editor, put all symptoms on the same line and separated them by column. We then put the newly formed list into the row corresponding to their disease. Since this was also done manually, this took about two hours to complete, tallying our total up to six hours.

As a visual example, here's a screenshot of our newly formatted data set.

Disease Full Dataset

	Column 1	Column 2
1	Accident Cerebrovascular	Dysarthria, Asthenia, Speech Slurred, Facial Paresis, Hemiplegia, ...
2	Acquired Immuno-Deficiency ...	Fever, Night Sweat, Spontaneous Rupture of Membranes, Cough, ...
3	Adenocarcinoma	Mass of Body Structure, Lesion, Decreased Body Weight, ...
4	Adhesion	Flatulence, Pain, Large-For-Dates Fetus, Para 1, Vomiting, Lung ...
5	Affect Labile	Extreme Exhaustion, Sleeplessness, Enuresis, Patient Non ...
6	Alzheimer's Disease	Drool, Agitation, Nightmare, Rhonchus, Consciousness Clear, Pin-...
7	Anemia	Chill, Guaiac Positive, Monoclonal, Ecchymosis, Tumor Cell ...
8	Anxiety State	Worry, Feeling Suicidal, Suicidal, Sleeplessness, Feeling Hopeless, ...
9	Aphasia	Clonus, Egophony, Facial Paresis, Aphagia, Muscle Twitch, ...
10	Arthritis	Pain, Hemodynamically Stable, Sleeplessness, Asthenia, Syncope, ...
11	Asthma	Wheezing, Cough, Shortness of Breath, Chest Tightness, Non-...
12	Bacteremia	Fever, Chill, Flushing, Unresponsiveness, Indifferent Mood, ...
13	Benign Prostatic Hypertrophy	Mental Status Changes, Cachexia, Blackout, Orthostasis, ...
14	Bipolar Disorder	Feeling Suicidal, Energy Increased, Suicidal, Irritable Mood, ...
15	Bronchitis	Cough, Wheezing, Shortness of Breath, Chest Tightness, Fever, ...
16	Carcinoma	Mass of Body Structure, Pain, Lesion, Tumor Cell Invasion, ...
17	Cardiomyopathy	Shortness of Breath, Orthopnea, Hypokinesia, Jugular Venous ...
18	Cellulitis	Erythema, Pain, Swelling, Redness, Fever, Abscess Bacterial, ...
19	Cholecystitis	Moan, Nausea, Pain Abdominal, Murphy's Sign, Flatulence, Colic ...
20	Cholelithiasis and Biliary Calculus	Vomiting, Nausea, Pain Abdominal, Pain, Cushingoid Facies and ...
21	Chronic Alcoholic Intoxication	Tremor, Hallucinations Auditory, Suicidal, Hoard, Irritable Mood, ...
22	Chronic Kidney Failure	Vomiting, Orthopnea, Hyperkalemia, Oliguria, Jugular Venous ...
23	Chronic Obstructive Airway Disease	Shortness of Breath, Wheezing, Cough, Dyspnea, Distress ...
24	Cirrhosis	Ascites, Fall, Splenomegaly, Pruritus, Pain Abdominal, Tumor Cell ...
25	Colitis	Fever, Thicken, Green Sputum, Vomiting, Nausea and Vomiting, ...
26	Confusion	Seizure, Enuresis, Lethargy, Speech Slurred, Fall, Consciousness ...
27	Coronary Arteriosclerosis and ...	Pain Chest, Angina Pectoris, Shortness of Breath, Hypokinesia, ...
28	Decubitus Ulcer	Systolic Murmur, Frail, Fever
29	Deep Vein Thrombosis	Swelling, Pain, Ecchymosis, Shortness of Breath, Pain in Lower ...

Unfortunately, there was one more thing we had to add: classes. Since we were implementing Naive Bayes, we needed to sort the diseases into three categories. The three classes were “emergency,” “seek medical attention”, and “wait”, which corresponded to the severity of the disease. These were added manually as well to every disease. Finally, here is a look at our completed dataset.

```
Manic disorder:emergency:Energy Increased, Suicidal, Hypersomnia, Feeling Suicidal,
Blanch, Hallucinations Auditory, Hallucinations Visual, Elation, Verbal Auditory
Hallucinations, Feeling Hopeless, Difficulty, Decompensation, Verbally Abusive Behavior,
Suicidal, Feeling Suicidal,
Melanoma:emergency:Mass of Body Structure, Paraparesis, Fever, Gravida 0, Pain, Pruritus,
Mass In Breast, Vomiting, Diarrhea,
Migraine Disorders:wait:Ambidexterity, Vomiting, Dizziness, Numbness, Nausea, Fever,
Splenomegaly,
Mitral Valve Insufficiency:emergency:Shortness of Breath, Dyspnea on Exertion,
Asymptomatic, Hypokinesia, Dyspnea, Syncope, Thicken, Left Atrial Hypertrophy,
Palpitation, Fatigue, Vomiting, Pain, Cardiomegaly, Chest Discomfort,
Myocardial Infarction:emergency:Pain Chest, St Segment Elevation, Sweat Sweating
Increased, Shortness of Breath, St Segment Depression, Hypokinesia, Angina Pectoris,
Pressure Chest, T Wave Inverted, Orthopnea, Rale, Chest Tightness, Presence of Q Wave,
Palpitation, Dyspnea, Chest Discomfort, Bradycardia, Syncope,
```

Evaluation:

The following outlines our evaluation paradigm.

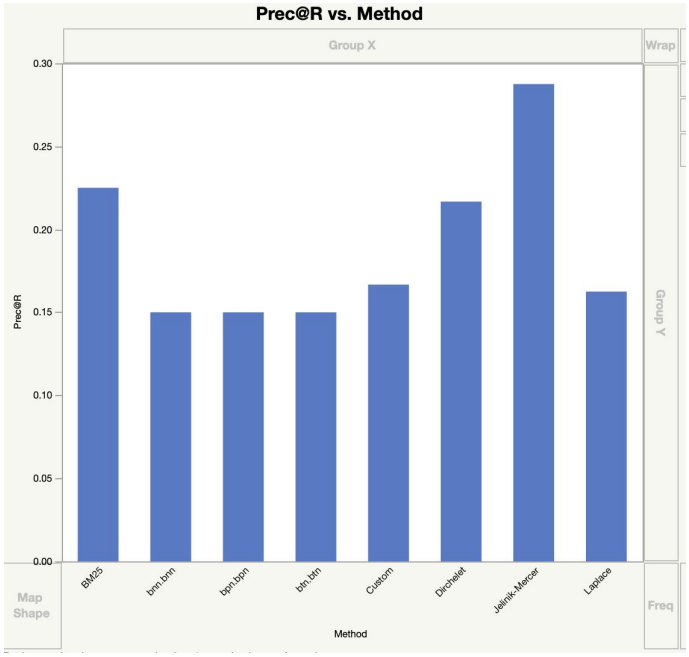
The corpus will be documents with an ID of the name of the disease and a comma-delimited list of symptoms as the text. Queries will search this corpus to find the disease documents that have the most matching symptoms.

Queries will be entered by the user as a comma-delimited list of symptoms. The corpus will then be searched for documents that have the most similar symptoms as the query. The user will get the top 5 results.

The ground truth is what disease the user is actually suffering from. The disease that matches what the user has is the most relevant. In theory, we would have data from many doctors who treated patients, recorded their symptoms, and tested them for a given disease. They would use these results to determine the relevance of different diseases compared to a list of symptoms. They may also use the process of elimination to determine the rank of lower-ranking documents. When a doctor examines a person, they may assume a few different illnesses before testing and being certain of what they actually have. The similar illnesses can be seen as relevant, but less relevant than the actual disease. We can increase relevance by number of matching symptoms.

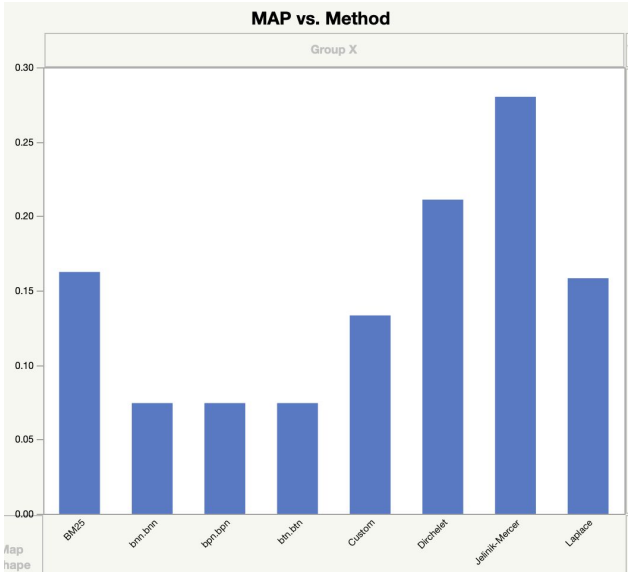
At the moment, we will be using WebMD as our relevance data that we will test against. WebMD is a more advanced website with the same function as ours, so it is a good benchmark for determining what ailment is relevant and what is not relevant. Of course, it is possible that the ailment that we find for our program is not classified as an ailment in the WebMD server, but as long as there are some matches between our data and theirs, we can create our own qrel file.

Here are the plots of MAP, Prec@R, and NDCG@5.

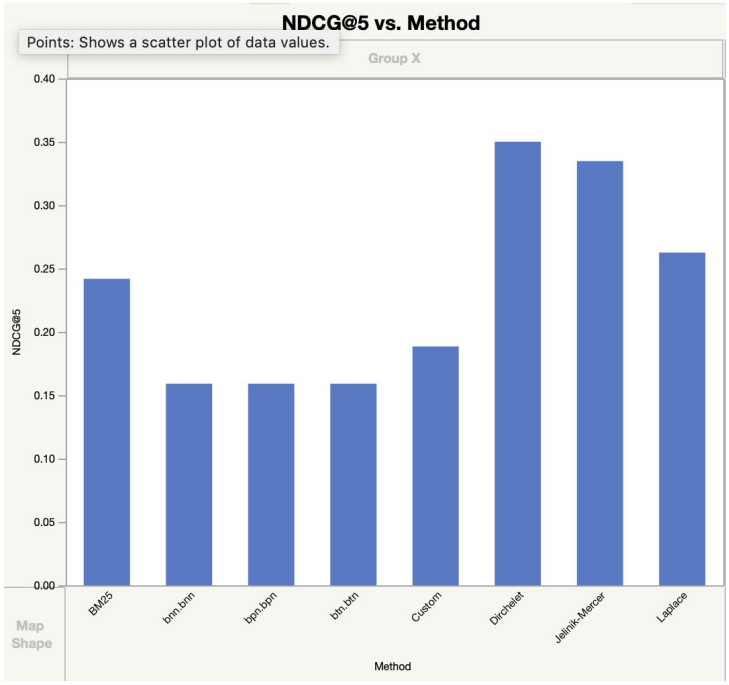


This is the plot of each Prec@R value for all of the methods. In this case, the Jelinik-Mercer smoothing seemed to have the highest value, followed by BM25. It's very rare that a method beats BM25, but this could be because our relevance information is very small. Using WebMD, we could only find, at a maximum, five relevant diseases for the queries. This

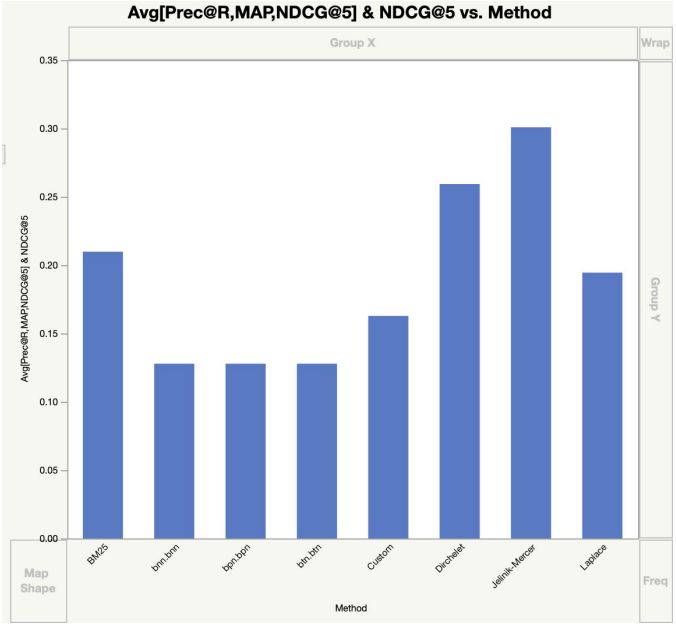
affected the methods using the relevant data (Jelinik-Mercer and Dirchelet) the most. The TF-IDF models did the worst in terms of Prec@R. This could be caused by an error in the code.



For MAP, we see a similar picture, except that Dirchelet has also edged out BM25. However, the reason could be the same as above, as Dirchelet uses our relevance information. It makes sense that MAP and Prec@R would have similar values, as they both rely on precision instead of recall.



Now, in terms of NDCG@5, we see that our final language model with Laplace smoothing has bypassed BM25. It could very well be that BM25 does not fare well with our data set, as we're not really searching through bodies of text; it's more like we're searching through lists of tokens. Also, Dirchelet has a higher NDCG@5 value than Jelinik-Mercer; we'll have to study the mean of all three evaluation metrics in order to determine which method was the best.



According to the mean of all three evaluation measures, it appears that Jelinik-Mercer was the best method that was implemented, followed by Dirchelet and BM25. However, it's hard to distinguish the exact values from this bar chart, and the standard error bars were not working, so here is our full data table with the results AND the standard error.

	Method	Prec@R	MAP	NDCG@5	Evaluation Mean	Evaluation Standard Dev	Evaluation Standard Error
1	BM25	0.225	0.1625	0.2419	0.2098	0.0341503538	0.019716716
2	Custom	0.1667	0.1333	0.1885	0.1628333333	0.0227005629	0.0131061761
3	bnn.bnn	0.15	0.0744	0.1591	0.1278333333	0.0379652765	0.0219192626
4	btn.btn	0.15	0.0744	0.1591	0.1278333333	0.0379652765	0.0219192626
5	bpn.bpn	0.15	0.0744	0.1591	0.1278333333	0.0379652765	0.0219192626
6	Laplace	0.1625	0.1583	0.2626	0.1944666667	0.0482080445	0.0278329275
7	Jelinik-Mercer	0.2875	0.2802	0.3348	0.3008333333	0.0242022497	0.0139731754
8	Dirchelet	0.2167	0.2111	0.3501	0.2593	0.0642459856	0.0370924371

These standard errors are very telling; the Jelinik-Mercer standard error interval would not overlap with the Dirchelet standard error interval, signalling that the difference between the two models may be statistically significant. On top of that, the difference between all methods (except for the TD-IDF methods) may be significantly significant, as the error intervals rarely overlap. We are not sure why this is the case.

The testing of our classification model was done in isolation. Once the model had been trained using our train set, we passed the list of diseases from our test set into our test method. The test method returns predicted categories for each of the test case diseases. We then compared the predicted categories against their actual counterparts. The results showed that out of the 39 diseases in the test case, the model was able to classify 38 of them accurately. Although, the result is highly optimistic, but we realize that the technique employed to evaluate the classification result is very crude and basic. Also, there is a possibility that the model could be overfitted, that is our model is too closely fit to our limited data.

Finally, we did notice a good amount of errors in our code that threw off our results. For one, we noticed too late that the scoring functions for the language models was incorrect; we had added the scores together instead of multiplied. We had already created all of the trec runfiles, and there was no time to remake them, but it is now currently fixed in our program for future use.

Also, the term frequencies were incorrect. Not all terms were accounted for in our list; some diseases even had 0 symptoms attached to them. We were not sure what caused such an error.

Conclusion/Future Work:

While our dataset lists each symptom as a token, it is very hard for a user to express the exact feeling and severity of their symptoms using just words, which could be why WebMD returns serious diseases for minor symptoms. While our program returns diseases from a given query, it only works with certain queries at the moment, so there is a lot of room for improvement and optimization.

Two things pop into mind for future work. First, a more optimal way to create relevance data should be explored. At the moment, we have to go into WebMD ourselves and look at their results. Some diseases will never be relevant in this case, as some diseases in our dataset will never appear in WebMD's results. Also, some symptoms in our dataset cannot be expressed correctly in WebMD, or our symptoms in our dataset lacks the details that WebMD asks for. To top it off, we can only use queries that we have relevance data for.

Creating the qrel and relevance text files for every possible query is next to impossible with these conditions, which limits the flexibility of our program. If there was a better way to create the relevance data quickly and more efficiently using our dataset, then we would love to know it. We contemplated using the results of BM25 as our relevance data, but we felt like professional input would've been more accurate, which might have led us down a dead end.

Second, an afterthought that occurred to us too late was that the symptoms were listed in order of strength of association. That means that some symptoms were more important than

others. It would be interesting to see if we can improve our search results by creating custom weights for each symptom in a disease; a symptom that is more prominent in a disease has a higher weight associated with it.

While our project might be slightly lacking, we have learned a lot from it. Since we all have the code for it, there is a possibility that we take a look at this program in the future in hopes of expanding it. There is a lot of room for expansion, and we've only scratched the surface of it.

Roles/Contributions:

Sarah's role in the project involved formatting the written documents, cleaning up the program in a proper style (such as proper spacing), collecting corpus metadata in the program for different search and evaluation methods, using trec_eval to output evaluation data, and organizing all resources in the group. She implemented a unique stemming method so that the system understood our input data, Spearman's Rank Correlation Coefficient, a manual thesaurus, and added classes to the dataset.

Ben's role in the project involved data set cleaning and mathematical insight for the methods used, as well as finding and creating relevance data. He implemented three unigram language models, each with a unique smoothing algorithm, and the main file output method. He also spent a good portion of his time condensing a large data set by hand into the form that was needed for the project, minus the classes.

Talha's role in the project involved evaluating the data, as well as dealing with anything related to training and testing sets in terms of evaluation. He implemented three variants of TF-IDF and his training and testing methods involving Bernoulli's Method.