CS221 Project: Doctor Bayes

Brandon Beckhardt
(beb619)

Leonid Keselman
(leonidk)

Anthony Perez
(aperez8)

Abstract

In order to alleviate the impact of healthcare spending on the the "last mile" of medicine, we built a system to predict a user's illness simply based on a description of their symptoms. Electronic medical records and clinician notes are difficult to access and so we explore online medical data bases as data sources. From the data on the Freebase, Mayo Clinic, and Wikipeida data bases, we trained several artificial neural network models and a bayesian network model. While handling multiple online data sources we found that a large part of the challenge was consolidating information across data sources in a consistent matter. We explore several techniques for automatic document matching across data bases of documents, with the goal of finding matching documents across Freebase, Mayo Clinic, and Wikipedia. We define matching documents to be articles which describe the same disease, then we found matching documents between databases with 91% accuracy.

1 Introduction

According to recent OECD estimates, healthcare expenditures for developed countries account for 10% to 20% of their national economies. In order to alleviate the impact of healthcare spending on the the "last mile" of medicine, we built a system to predict a user's illness simply based on a description of their symptoms. Originally we'd hoped to develop a classification system on top of electronic medical records, clinician notes, and the ICD-10 disease classification system. We were unable to find publicly available EMRs so instead we created Doctor Bayes, an illness classification system whose education comes only from reading about medicine on the web.
There are many interactive disease classification applications available but we wanted to see how well we could classify diseases solely based on plain text collected from the web and generated by a user. The input to our algorithm is raw, unprocessed, plain text, like the text that you are reading now. This text would be generated by a user. The output is the name of a disease, in text form, selected from a list of diseases known by and supplied to the algorithm. We implemented many algorithms to transform the input to an output, of which a Naive Bayes model was most effective. Given an input, each algorithm ranks the disease from most likely to least likely. We recorded many error metrics, but the one we used for algorithmic purposes and the one we thought was most informative was top five accuracy, the percentage of time the correct disease was in the top five ranked diseases. This is a supervised learning task because we know the disease each document is describing.
One of the large problems we discovered about this task, was that different data sources have inconsistent naming conventions for their articles, meaning that the same disease might be in multiple sources under different names. We tackled the problem of automatically pairing documents across sources by disease. In other words, for each disease present in any of our data sources, we needed to automatically find all the documents in all the data sources that were descriptions of this disease. Again, this is a left learning task because we know the disease each document is describing.
We shared the data set and feature extraction techniques of this project with another project we were all working together on in CS229. All models described in this document were unique to this project. The exploration of document similarity is also unique to this project. The CS 229 project explores Naive Bayes, Logistic Regression, Random Trees, and Cosine Similarity models for predicting disease class from a feature vector.

```
>> Redness, itchiness and difficulty opening eyes in the morning
1, Conjunctivitis (Pink Eye)
6.6e-08, Glaucoma
7.7e-10, Dermatitis
1.6e-10, Blepharitis
8.3e-11, Sleepwalking
```

```
Answer: Conjunctivitis

>> Fever headache and with a rash on my wrists.
1, Rocky Mountain spotted fever
3.4e-05, Meningitis
2e-06, Lyme disease
1.8e-06, Rheumatic fever
1.4e-06, Dengue fever
Answer: Rocky Mountain spotted fever

>> I have to go to the bathroom a lot and my stomach hurts all the time.
1, Social anxiety disorder
0.0012, Generalized anxiety disorder
0.00021, Bulimia nervosa
0.00016, Gastroenteritis
4.7e-05, Osteoarthritis
Answer: Irritable Bowel Syndrome
```

2 Related Works

The work most related to ours, researchers from 3M built an automated tagging scheme, which takes clinician notes and predicts a standardized disease code [12]. For their testing and training, they were able to use Electronic Health Records (EHRs) and found that regularized logistic regression could give them a 56% accuracy. While we originally wanted to pursue a similar task with labeled EHRs, we were unable to find any publicly available datasets of medical records or clinician notes (the 3M researchers used data from 3M's proprietary database). Thus, we had to find and create our own datasets. Our work on this is specified in our section on Data. The interesting differentiation is that our mismatch of datasets puts a much larger emphasis on our system's ability to generalize.

There is a long history of machine learning being applied to medical classification. For a historical retrospective, [6] covered the use of various learning algorithms (kNN, Decision Trees, and Bayesian methods). Their review suggests many learning methods applied to many medical fields, however, almost all their examples apply only to binary classification of a single disease. That is, oncologists build cancer detectors, cardiologists build heart disease detectors[11], and so on. Our system is different in that it is a general diagnostics system, relevant to first-pass classification of any illness (as long as it's our automated catalog). Similar sentiment can be found in modern overviews of learning methods in medicine [9]. Some of this more recent work [2] focuses and recommends the use of SVMs. However, in our case, we used plain-text features and found SVMs to perform poorly.

As our classification methods focus on using plain text analysis to predict diseases, we also reviewed modern methods in text classification. Of specific interest was work on Baysian pragmatic reasoning, where a speaker model is used to help shape the probabilities of hearing certain utterances [3]. The key concept of that work is that, if three things can be described as "cold" but two of them are "cold and wet", then a speaker saying "cold" is more likely to be referring to the not wet ones. If we treat our document counts as a model of speech, then the cosine similarity (which performs normalization across every feature vector in the training set) can be seen as performing the same desired effect.

One of our proposed models uses a convolutional neural network for text classification. Unlike existing methods applying convolutional networks to text [15] [4], which build a temporal model and or are done on one-hot or character-level features, we apply ours to a tf-idf word count of an entire document. We propose a novel method for providing a consistent ordering of features across such a large document-level feature vector.

One of our primary and most successful methods for classification came from using multinomial Naive Bayes, which have a long history of success when used with other methods like tf-idf [5]. Other papers in the field have suggested the use of priors to help over-fitting [10]; we implemented a set of priors for all our classes and found it was actually detrimental to our results.

3 Data and Infrastructure

Our data consists of free text we collected from Mayo Clinic, Freebase and Wikipedia. We collected this data by crawling the three websites using Scrapy, a python library for web crawling. With the

known structure of the Mayo Clinic and Freebase databases, we were able to extract both a list of symptoms and a description for each disease from the two sources. With over 12,000 different disease articles, many of which did not match across sources, we had to come up with a way to unify our data sources so that we would have an article for each disease from each data source. The Freebase data base contains a list of aliases for each disease which provides a list of alternative names for each disease. We used this alias list to unify our sources. We did this by having our final set of diseases be the intersect between the three data sources. In other words, to create this final data set we iterated over the disease entiries in Freebase and found the matching entries in both Mayo Clinic and Wikipedia by matching the titles in Mayo Clinic and Wikipedia to either the name of the entry in Freebase, or an alias of the entry in Freebase. After unifying our sources, there are 486 different possible disease classes and after splitting Freebase and Mayo Clinic into symptoms and description data sets, we had 2430 examples total across all diseases. This is the data set we use in this paper for algorithms predicting disease from user given symptom descriptions.
Data Sets:

- Freebase Symptoms
- Freebase Descriptions
- Mayo Clinic Symptoms
- Mayo Clinic Descriptions
- Wikipedia

We use the Wikipedia examples as a test set and the other examples as a training set. The reason we do this instead of cross validation is because we cannot split any of the data sets with the following reasoning. All of the five example sets mentioned are significantly different from one another, and we cannot split the data within an example set for cross validation, because each example set contains only one example of each disease. Therefore, any testing scheme is going to pit one example set against the others in terms of testing and training. We chose Wikipedia as the testing set due to its more free form nature which would better reflect informal human writing and because it contains information that is unrelated to the symptoms of the disease, which would add noise if it were a training set. To create a validation set, we divide the Wikipedia, data in half, using one half as the test set and the other as a validation set. When generating a test example, we extract either the paragraphs describing the symptoms of the disease or if such a paragraph is not present, we use the paragraphs in the main section of the article (the section directly after the title of the article). In order to avoid giving away the correct label to our learning algorithm, we remove all instances of the disease name from the Wikipedia articles when we generate the testing data. We also created a set of 60 handwritten testing examples which model what a human might input into the system.
However, we wanted to explore better automated ways of finding matching disease articles across sources, going from 12,000 diseases to 483 seems like a far too conservative estimate for the number of diseases shared across data sources. To constructed these automated ways of finding similar documents, we used the Freebase, Mayo Clinic, and Wikipedia data as a list of documents, and Freebase's alias section as the ground truth of which documents match to which other documents. Hence we have 3 sets of 483 documents, one from each data source, and we know the truth of which documents are matching. Each document describes exactly one disease and for each of the 483 diseases, each data sources has a document describing it.

3.1 Priors

For many of our algorithms which predict disease from user input, knowing priors, meaning the frequency of occurrence of each disease, increases the performance of the algorithm. A useful metric for generating priors would be the number of occurrences per year seen for a disease. We could not find a cumulative list of occurrences for all diseases and manually searching for 486 diseases would not be a good use of time for this project, so we decided a way to gauge the likelihood of a disease would be its relevance to society as captured by the number of results in a Google search (ex: ”Alzheimer's” Medical). The idea is that the more articles related to a disease, the more people it affects. Each prior was calculated as $\frac{\text{Number of results for diseases}}{\text{Total number of results}}$. Generating priors in this manner turned out to be a failure and actually decreased our results in both the top 1 prediction and top 5 prediction.

4 Feature Extraction

At its core, we extract features from our plain text data sources by building a document-term matrix. This is a matrix where each row represents a document (a disease article in our case), each column

represents a term (a word), and each element is a number representing the number of times the term (given by the column) appears inside the document/disease article (given by the row). The terms used are determined by the words present in our data sources and without any of the modifications we implemented each word would be its own term in the document-term matrix. By its nature the document-term matrix tends to be sparse, and was in our case. It is important to note that we partitioned our data sources such that there were four rows corresponding to each disease, each of which is labeled identically. Each feature vector fed to our learning algorithm is a row from the document-term matrix and the corresponding label fed to our learning algorithm is the label given to the row. We store the document-term matrix as a compressed sparse row matrix to increase the computational efficiency of our algorithms.
User queries, which we must classify by assigning a disease, are vectorized in the same way we generate a row for the document-term matrix. Thus each query is turned into a vector of word counts. Terms in user queries which were not present in any of the documents are discarded.

Given the document-term matrix set up, there are two concepts we explored for manipulating our features to increase the performance of our system. Reducing the number of columns in the matrix (i.e. reducing the number of terms we record) and adjusting the word counts of our elements. All of our feature engineering is aimed at reducing the feature space because we identified this problem as having high variance (being prone to overfitting) for most of our learning algorithms. In later sections we justify our claim of high variance by noting that our training error is near 0% for most models, but our testing error is much higher. The extremely high dimension and low sample size nature of our task contribute to overfitting and we also see increasing the data we give our models reduces test error, which is expected with a high variance problem.

We implemented five techniques:

- Tf-idf weighting
- Stemming
- Stop words removal
- Document frequency based removal
- Non-alphabetical removal

Where tf_w is the new term frequency for element w and $count_w$ is the observed term frequency from each document for element w (note this is element wise).
The inverse document frequency weighting is give by:

$$idf_w = log(N/(1 + N_w))$$

Where idf_w is the idf weight of element w, N is the number of documents, and N_w is the number of documents containing the term corresponding to the column of element w.
For each document, the raw count of each feature/term is passed through the term frequency weighting function, then multiplied by the inverse document frequency weighting. Tf-idf weighting has two big effects: words which are common to many documents have a very low weight and are essentially eliminated if they are too common because the weighting is so low (reducing our feature space), and a disease with a single word appearing many times in its articles is not tied so closely to that word.
Stop words are a human made list of word that carry no information, which are removed from the documents. Document frequency based removal refers to removing words that are in more than 90% of documents. Non-alphabetical removal refers to removing words that do not contain alphabetical characters. The idea behind these techniques is that the words they remove are not predictive of which disease a user might have because they are either words that do not mean anything, punctuation, or numbers. These words only contribute to statistical noise and are thus better off removed. With these techniques we reduced our number of features from 11067 to 10740.
Stemming is a process that reduces a word to a more basic form. For example, the words 'reduces', 'reduced', 'reducing', and 'reduce' map to 'reduc' after stemming. Stemming tries to capture the meaning of a word but discards its grammatical form. By applying stemming we reduce our number of terms from 10740 to 7577 (with all other techniques applied).

5 Document Similarity

Having an automated method of matching documents describe the same disease across data sets was not attempted in the CS229 version of this paper. To perform the function of matching documents, we explored four techniques: Cosine Similarity, SVD, Latent Semantic Analysis, and Latent Dirichlet Allocation.

Model	Dimensions	Top 1	Top 5
Dot Product	7,379	14.20%	21.06%
Cosine Similarity	7,379	90.67%	97.46%
Latent Dirichlet Allocation	128	8.71%	19.82%
SVD	128	21.06%	28.67%
SVD-S	128	21.12%	28.67%
SVD-N	128	86.01%	96.43%
SVD-NS	128	86.01%	96.43%
SymSVD	128	20.58%	28.40%
SymSVD-N	128	70.58%	91.63%
SymSVD-S	128	20.85%	29.08%
SymSVD-NS	128	84.84%	96.91%

Table 1: Document Similarity Results. N means normalization. S means the singular values are taken to the 0.5 power. Sym means we use symmetric SVD as defined in [7]

5.1 Baseline

The baseline method to find document similarity was taking the dot product of our our TF-IDF weighted document term matrices for each disease. Taking the dot product of two matrices uses both the angle and and magnitude of the two to compare them. We compared descriptions of symptoms from each source against all other sources and ranked the results of the dot products. The baseline results were: we got 14.20% of comparisons correct in the first result, and 21.06% of comparisons correct in the top 5 results.

5.2 Cosine Similarity

We implemented Cosine Similarity on our document matrices, which measures the similarity between two vectors by the cosine of the angle between them. To predict a disease from a query, for each document the algorithm calculates a score for the document query pair, given by:

$$score_d = \frac{\vec{d} \cdot \vec{q}}{(||\vec{d}||_2||\vec{q}||_2)}$$

Where $score_d$ is the score of document d, $\vec{d}$ is the vector representation of document d, $\vec{q}$ is the vector representation of the query.
This is the dot product of the normalized vectors of the document and the query. This represents the cosine of the angle between the query and the document vector (which monotonically decreases as the angle increases), hence our description as a nearest neighbors method. The top scoring document (with higher score meaning the document is a closer match to the query), and hence the predicted disease is given by:

$$d = \arg\max_d \frac{\vec{d} \cdot \vec{q}}{(||\vec{d}||_2||\vec{q}||_2)}$$

Since each document corresponds to a disease, the result of this model can be used to order the disease by likelihood. We tested our results using both the dot product of the un-normalized vectors as well as Cosine Similarity. Cosine Similarity gave us the best results out of any document similarity method, as seen in the ”Results” section.

5.3 SVD

We used Singular Value Decomposition to factor our document term matrix. The Singular Value Decomposition decomposes the document term matrix into matricies U, S, and V, such that U * S * V recovers the original matrix. Using the U matrix's rows as vectors representing document concepts allows us to find similar documents[1] .

5.4 Latent Semantic Analysis

Latent Semantic Analysis is widely used in natural language processing to analyze relationships between documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA uses Singular Value Decomposition then runs Cosine Similarity. The reason LSA didn't perform better than Cosine Similarity is because the dimensions of the matrix after SVD are compressed, so data is lost. LSA is a useful tool when working with extremely large amounts of data, but was not as beneficial with the amount of data we used. An example of 2D representations of word vectors used with LSA can be found in figure [1].

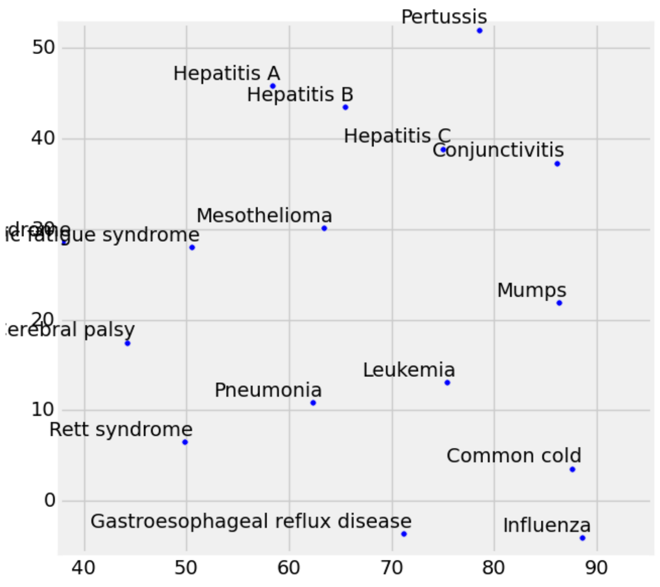
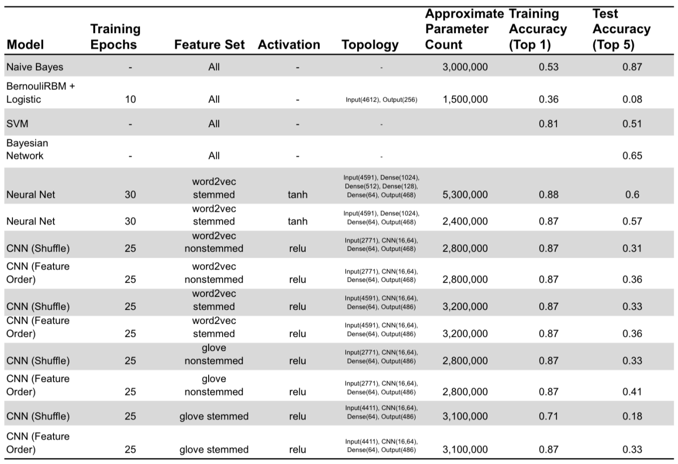


Figure 1: 2D t-SNE plot of the representation of Word Vectors using Latent Semantic Analysis. Clustering like the cold and flu are visible, is the cluster of Heptatises.



Model	Training Epochs	Feature Set	Activation	Topology	Approximate Parameter Count	Training Accuracy (Top 1)	Test Accuracy (Top 5)
Naive Bayes	-	All	-	-	3,000,000	0.53	0.87
BernouliRBM + Logistic	10	All	-	Input(4612), Output(256)	1,500,000	0.36	0.08
SVM	-	All	-	-		0.81	0.51
Bayesian Network	-	All	-	-			0.65
Neural Net	30	word2vec stemmed	tanh	Input(4591), Dense(1024), Dense(512), Dense(128), Dense(64), Output(468)	5,300,000	0.88	0.6
Neural Net	30	word2vec stemmed	tanh	Input(4591), Dense(1024), Dense(64), Output(468)	2,400,000	0.87	0.57
CNN (Shuffle)	25	word2vec nonstemmed	relu	Input(2771), CNN(16,64), Dense(64), Output(468)	2,800,000	0.87	0.31
CNN (Feature Order)	25	word2vec nonstemmed	relu	Input(2771), CNN(16,64), Dense(64), Output(468)	2,800,000	0.87	0.36
CNN (Shuffle)	25	word2vec stemmed	relu	Input(4591), CNN(16,64), Dense(64), Output(486)	3,200,000	0.87	0.33
CNN (Feature Order)	25	word2vec stemmed	relu	Input(4591), CNN(16,64), Dense(64), Output(486)	3,200,000	0.87	0.36
CNN (Shuffle)	25	glove nonstemmed	relu	Input(2771), CNN(16,64), Dense(64), Output(486)	2,800,000	0.87	0.33
CNN (Feature Order)	25	glove nonstemmed	relu	Input(2771), CNN(16,64), Dense(64), Output(486)	2,800,000	0.87	0.41
CNN (Shuffle)	25	glove stemmed	relu	Input(4411), CNN(16,64), Dense(64), Output(486)	3,100,000	0.71	0.18
CNN (Feature Order)	25	glove stemmed	relu	Input(4411), CNN(16,64), Dense(64), Output(486)	3,100,000	0.87	0.33

Figure 2: Results from neural network testing

5.5 Latent Dirichlet Allocation

Latent Dirichlet allocation is a generative model used in natural language processing that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. LDA had the worst performance out of all the models we tried.

5.6 Results and Error Analysis

Referencing 1, we see that Cosine Similarity had the highest results in both Top 1 and Top 5 results, while Latent Dirichlet Allocation performed the worst. This is likely due to the fact that Cosine Similarity has over 7000 dimensions where as any other algorithm (aside from the basic dot product) has just over 100 dimensions. Although Cosine Similarity did well, it is difficult to perform when there is a lot of data to handle. Normalized SVDs, which performed particularly well are a good alternative to Cosine Similarity. Although it was 4% lower in prediction Top 1, it requires far fewer dimensions and may be the algorithm to use with big data sets.

6 Word Vectors

6.1 Clustering Symptom Words

One application of clustering is finding the associations and meaning between words, so we tried a few different ways to cluster words. There are many option for the vectorization of words that would allow clustering to take place. One option was to create a vectorization of words that is based on the patterns found in our Freebase, Mayo Clinic, and Wikipedia free text. We tried doing this by using both Tensor Flow's and Gensim's Word2Vec libraries, which are deep learning neural nets that process text. Below is a visual of the word a 2D representation of the word vectors that were generated by Tensor Flow. This plot was made using TSNE, which took the high dimensional vector space of each word and made it into a 2D representation (seen in figure 3).

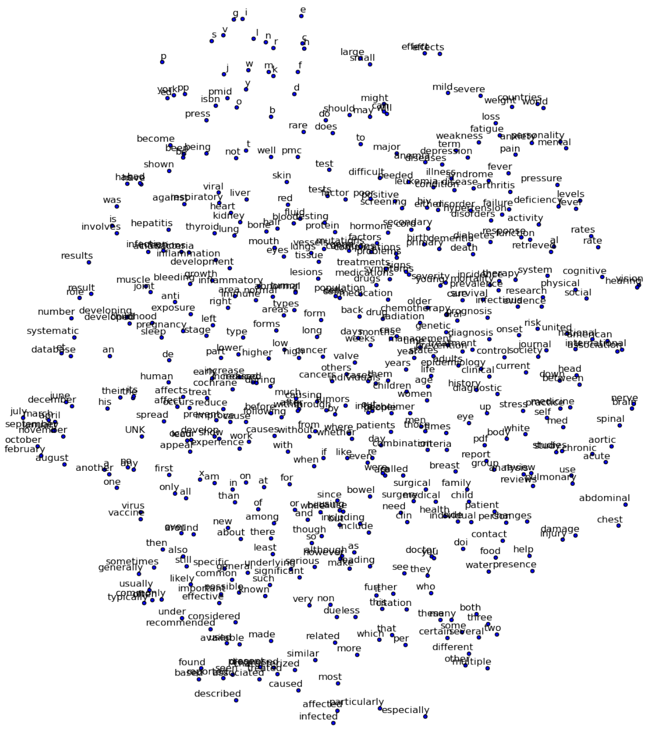


Figure 3: 2D Representation of Tensor Flow Word Vectors, clusters similar words like damage and injury, or all the months are clearly visible

The clustering we got from Tensor and Gensim was not very meaningful - finding actual associations in the clusters was near impossible and running the vectorization algorithms tooks hours. For this reason, we wanted to see if we could get better clusters using pretrained word vectors from GloVe, which runs an unsupervised learning algorithm to obtain vector representations for words. The model we used was trained on Wikipedia 2014 and Gigaword 5, which was our best option since some of our data is from Wikipedia. We ran K-Means on the vectors with cluster sizes of 5, 10, 15 and 20 and only used words that appear in our symptom descriptions. 5 clusters was not enough, as the clusters didn't have much meaning while 20 clusters was too much, with each cluster not containing many words. 10 and 15 clusters had the best results, with 15 being the most meaningful. The general breakdown is as follows:

- Clusters 0,1,5,11,12,14,15 consisted of very specific words that provide insight into a document into the disease and area of the body it is affecting, but are words that a user would not use to describe their symptoms. These words would be very useful if we replaced them with their simple aliases.
- Clusters 1,6,7,8 were meaningless words that a user would never input and provide no added information. 9 contained no information.
- Clusters 3, 13 consisted of common words that a user would certainly input into the system. These words did give insight yet also appear commonly in the english language, so perhaps would not hold as much weight relative to other words.

Using pretrained word vectors to create clusters did provide some insight, but it did have some downfalls that may have been avoided if we were able to better train our word vectors using our own data. One obvious pitfall of using generic, pretrained, vectors is that our clusters tend to group medical words together. Most of the words we use are medical words so clustering them on that criteria doesn't add much insight.
In an ideal scenario we could train word vectors based on our medical free text and create clusters that can provide insight as to how words in symptom descriptions can be weighted and what the presence or absence of certain words says about the disease.

7 Disease Prediction

In the CS229 version of this paper, we explored Naive Bayes, Logistic Regression, Cosine Similarity, and Random Trees models as models used to predict a disease from user input. See the results section of the other paper for more information on how well these models did. For the CS221 project we implemented a Bayesian Network, an SVM, an Artificial Neural Network, and a Convolutional Neural Network.
One of the challenges we faced was that we have over 7500 features but less than two thousand examples. As a result, our models tend to have a very large number of parameters and thus tend to overfit the data.

7.1 Baseline

As our baseline we chose to implement unregularized Logistic Regression (the Logistic Regression we implemented for CS229 was regularized). The Logistic Regression model achieved a 18% top one accuracy and 38% on top five accuracy on the manual test data and a 42% top one accuracy and 67% top five accuracy for Wikipedia test data. Change to Cosine similarity with no feature extraction techniques.

7.2 Oracle

A study done in ... reports that physicians average a 55.3% success rate in diagnosing their patients [8]. This corresponds can be compared to the top one accuracy of our model. In order to get a better judge of top five accuracy for our models (percentage of time the correct disease is ranked among the top five most likely diseases by the model), we chose to use our Naive Bayes model from the CS229 version of this paper, which was the best performer. The Naive Bayes model achieved a 63% top one accuracy and 88% on top five accuracy on the manual test data and a 67% top one accuracy and 87% top five accuracy for Wikipedia test data.

7.3 Bayesian Network

Our Bayesian Network model has two layers. One layer has a node for each feature and the other layer has a node for each disease. We call these layers the feature layer and disease layer respectively. Each node in the feature layer is connected to each node in the disease layer. We restrict each feature to take on binary values (zero or one), and disease nodes naturally take on a binary value of present (one) or not present (zero). Using this mental model and Bayes rule we can derive the following:

$$p(disease|features) = \frac{p(disease, features)}{p(features)}$$

To simplify this model, we assume each feature is independent from one another giving us:

$$p(disease|features) = \frac{\prod_i p(disease, features_i)}{p(features)}$$

To predict a disease from a feature vector x, we predict the probability of each disease being present and use this to order the diseases from most likely to least likely. This is given by:

$$p(disease = 1|features = x) = \frac{\prod_i p(disease = 1, features_i = x_i)}{p(features = x)}$$

To use this model we need to estimate $p(disease = 1, features_i = x_i)$ for each disease and feature and we need to estimate $p(features = x)$. We estimate $p(features = x)$ as $\prod_i p(features_i = x_i)$, again making the assumption our features are independent. We estimate these values directly from our data.

$$p(features_i = x_i) = \frac{\text{\# of times } features_i = x_i}{\text{\# of times } features_i \text{ was observed}}$$

$$p(disease = 1, features_i = x_i) = \frac{\text{\# of times } features_i = x_i \text{ and } disease = 1}{\text{\# of times } features_i \text{ was observed when } disease = 1}$$

The idea behind using only binary indicators for features was to address the over-fitting problem prevalent in many of our models. Sadly, the model does not generalize well in the sense that it has poor performance on the manual test results.

7.4 SVM

Support vector machines are a commonly used algorithm for medical classification tasks. However, in our case SVMs did very poorly. To perform multiclass prediction, the SVM model is setup in a one versus rest scheme, meaning one SVM is trained for each class and each SVM provides a score for how likely a test point is to be its class (based on distance from separating hyper plane). This means each SVM was operating in a very high dimensional space with very unbalanced classes. We suspect this is why the SVM did so poorly. The SVM model had reasonable success on the wikipedia test data (51%) but failed on the manual test data, getting only a 0.5% top 5 accuracy for manual examples.

7.5 Convolutional Neural Network

In a traditional neural network, a number of fully-interconnected layers are used to minimize residual errors from a top layer. In classification, the top layer is typically a softmax normalized output layer that is the number of layers in size. A one-hot feature vector is then used to perform classification training via error propagation through the network. Additionally each layer has a non-linear activation function which maps input to output.

In a convolutional network, the layers that directly interact with the input are replaced with convolutions. Instead of having a dense set of connections of size $inputSize \cdot layerSize$, each input element has a bank of convolutional kernels applied. These convolutions help reduce the parameter size of the network, as a small bank of filters is reused across the entire image. Such networks, and their associated benefits of fewer parameters, have been key in modern research in image classification [13].

However, in order to apply convolutions, one needs to first create some value in spatial locality, which is readily available in image networks or word sequences. In our case, however, input is simply a tf-idf feature vector of words.

To show the efficacy of CNNs on text input, we developed a novel method in which we create an ordering for our word count features, and then apply a CNN for classification. The topology can be seen in figure 4.

Although our overall results were still worse than our Naive Bayes classifier, we were able to demonstrate the value of using a word-ordering system. We believe that the weak results of our classifier were mostly due to our small training corpus and lack of time to try various interesting convolutional topologies (as each classifier took about an hour to test and train).

In order create a sensible ordered set of features, we applied the techniques discussed in the ”Document Similarity” section to generate word vectors for each of our features. These word vectors map

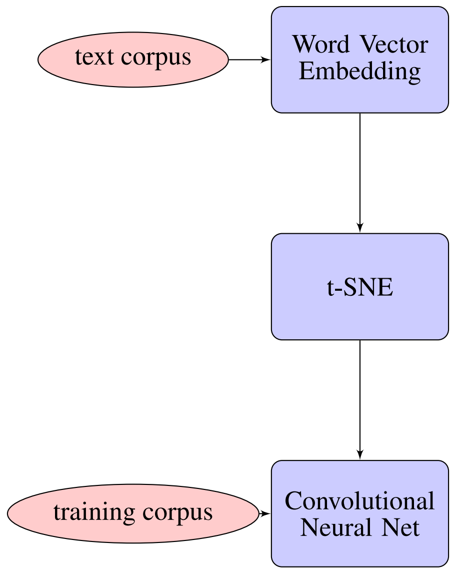


Figure 4: Overview of our method of generating a CNN label

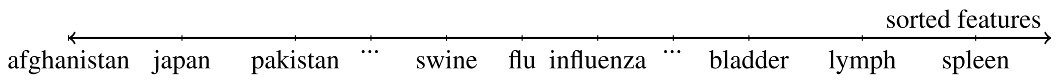


Figure 5: Overview of the ordering generated by using pre-trained GloVe vectors and a t-SNE ordering, as consumed by our CNN

the words to points in space where words with similar meaning are closer to one another. We then used t-SNE [14] on these word vectors to generate a one dimensional ordering of the words. The goal of this was to have similar terms grouped, so that convolutions could apply to our text feature vectors. Examples of groupings found can be seen in figure 5 .

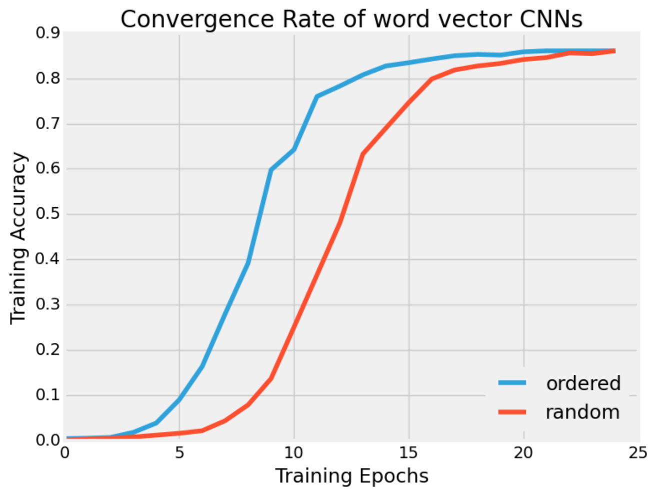


Figure 6: The benefits of our sorted word vectors in demonstrating CNN convergence.

The strongest result we had was that our sorted convolutional networks performed much better than convolutional networks with shuffled features. An example result can be seen in figure 6.

From the results in figure 2, we see that Convolutional Neural Networks did not perform to the standard we had hoped. There is clearly an overfitting problem given that the training accuracy is relatively high while the test accuracy is low. We had hoped that the convolutional neural nets would not suffer from this problem because they contain less parameters, but this turned out not to be the case. Surprisingly, the normal Artificial Neural Network had a better test accuracy with a similar training accuracy to the CNNs.
On the other hand, the feature ordering increased testing accuracy for CNNs, regardless of the architecture of the net. This is what we expected given that CNNs are designed to word on data where adjacent features ”share” information.

8 Challenges/Looking Forwarding

Lack of Clinical Data: None of our data sources exactly represent the two types of notes we'd like to learn from: actual clinician notes and people's descriptions of their own illnesses. Training our system on data which better represents our desired queries would greatly strengthen the quality of the model. Overconfidence: Our model tends to overfit, guessing either right or wrong with very high probability. We'd like to alleviate this by obtaining additional data and reducing the size of the feature space. Using only word referring to symptoms may be one solution. Quality Priors: It would be beneficial to access full data records to get the actual incidence of diseases as our current priors are unreliable.

8.1 Medical Records

The data we collected was from formal and academic descriptions of symptoms associated with a disease. The data used very specific medical words to describe a disease. A patient using this application would not use medical words such as ”lipodystrophy” or ”reticularis”, which are often found in the data we are training on. Training our data on electronic medical records, clinician notes, and ICD-10 information would introduce our algorithm to descriptions it would most likely be getting from a user. There was a dataset of medical records for a ”Practice Fusion” challenge on Kaggle, but unfortunately it was taken down before we could acccess it.

9 Conclusion/Thoughts for industry

There are many web applications that predict diseases, all of them using hand crafted metrics and experts in the field to construct the model. Using only free text from the web and no field knowledge, we were able to get results that were comparable to the 53% success rate seen by physicians. This was done in two months without medical records or heavy clinical testing. We hope the use of algorithms on free text can provide a new way of thinking about medical diagnoses from a purely medical practice to a practice of data analysis and manipulation.

One dataset we feel could have helped our project significantly, as well as anyone else working in the field, is a large set of anonymous set medical records. If we were able to analyze abundant patient descriptions as well as doctors notes, we feel we coudl've have gotten our algorithms to perform significantly better. Unfortunately, HIPAA laws make it difficult to release such sensitive information. We feel the availability of medical records could advance the field of medical diagnoses significantly, and we hope to see a solution to this problem in the years to come.

Although doctors are irreplaceable, it would be beneficial to the healthcare system to have a way to reliably get an assessment of health without human interaction. We see this application going beyond a plain text description of symptoms, to a fully interactive system that can measure full biometrics and gauge health in ways that a human simply can't. We are far from the day where we can reliably get a diagnoses and prognosis for medical conditions, but artificial intelligence will play a key role in the improvement of medical paradigms in the years to come.

References

[1] Scott Deerwester et al. “Indexing by latent semantic analysis”. In: *Journal of the American Society for Information Science* 41 (1990), pp. 391–407. ISSN: 0002-8231. arXiv: `arXiv:1403.2923v1`.

[2] Kenneth R Foster, Robert Koprowski, and Joseph D Skufca. "Machine learning, medical diagnosis, and biomedical engineering research - commentary". In: *BioMedical Engineering OnLine* 13.1 (2014), p. 94. ISSN: 1475-925X.

[3] M. C. Frank and N. D. Goodman. "Predicting Pragmatic Reasoning in Language Games". In: *Science* 336 (2012), pp. 998–998. ISSN: 0036-8075. DOI: `10.1126/science.1218633`.

[4] Rie Johnson. "Effective Use of Word Order for Text Categorization with Convolutional Neural Networks". In: (2015), pp. 103–112. arXiv: `arXiv:1412.1058v2`.

[5] Am Kibriya et al. "Multinomial naive bayes for text categorization revisited". In: *In AI 2004: Advances in Artificial Intelligence* (2005), pp. 488–499. ISSN: 03029743.

[6] Igor Kononenko. "Machine learning for medical diagnosis: history, state of the art and perspective". In: *Artificial Intelligence in Medicine* 23.1 (2001), pp. 89–109. ISSN: 0933-3657.

[7] Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225. ISSN: 2307-387X.

[8] Ashley N. D. Meyer et al. "Physicians' Diagnostic Accuracy, Confidence, and Resource Requests". In: *JAMA Internal Medicine* 173.21 (2013), p. 1952. ISSN: 2168-6106.

[9] Paul Sajda. "Machine Learning for Detection and Diagnosis of Disease". In: *Annual Review of Biomedical Engineering* 8.1 (2006), pp. 537–565. ISSN: 1523-9829.

[10] Karl-Michael Schneider. "Techniques for Improving the Performance of Naive Bayes for Text Classification". In: *Computational Linguistics and Intelligent Text Processing* i (2005), pp. 682–693.

[11] Jyoti Soni et al. "Predictive Data Mining for Medical Diagnosis: An Overview of Heart Disease Prediction". In: *International Journal of Computer Applications* 17.8 (2011), pp. 43–48. ISSN: 09758887. DOI: `10.5120/2237-2860`.

[12] Michael Subotin and Anthony R Davis. "A System for Predicting ICD-10-PCS Codes from Electronic Health Records". In: *Workshop on BioNLP* BioNLP (2014), pp. 59–67.

[13] C. Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *ArXiv e-prints* (Dec. 2015). arXiv: `1512.00567 [cs.CV]`.

[14] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.2579-2605 (2008), p. 85.

[15] Xiang Zhang and Yann LeCun. "Text Understanding from Scratch". In: *CoRR* abs/1502.01710 (2015). URL: `http://arxiv.org/abs/1502.01710`.