CS229 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 a Naive Bayes, Logistic Regression, Random Trees, and Cosine Similarity Model Many of the techniques we employed were aimed at eliminating over-fitting and improving generalization.

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. This is a supervised learning task because we know the disease each document is describing. Below are some example inputs and outputs. The model outputs the top five most likely disease with their probabilities.

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

2 Related Work

In the work most related to ours, researchers from 3M built an automated tagging scheme, which takes clinician notes and predicts a standardized disease code [15]. 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. An 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. Most major reviews for medical classification provide a wide overview of different learning methods [8][12][4]. These reviews suggest that these methods to diagnostics across all of medicine. However, almost all their examples only to binary classification of a single disease. That is, oncologists build cancer detectors, cardiologists build heart disease detectors[14], 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.

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 [5]. 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 one. If we treat our document counts as a model of speech, then cosine similarity (which performs normalization across every feature vector in the training set) can be seen as performing the same desired effect. This method performed very well on our dataset.

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 [7]. Other papers in the field have suggested the use of priors to help over-fitting [13]; we implemented a set of priors for all our classes and found it was actually detrimental to our results.

3 Data

Our data consists of free text we collected from Mayo Clinic, Freebase and Wikipedia. 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. We unified our data by exploiting Freebase's list of available aliases for each disease. After unifying our sources, there are 486 different possible disease classes. For each class we have 5 examples, two from Freebase (descriptions and symptoms), two from Mayo Clinic (descriptions and symptoms), and one from Wikipedia bring us to a total of 2430 examples. 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, 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. 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. We also created a set of 80 handwritten testing examples which model what a human might input into the system; these handwritten examples consistent our best example of real user input.

3.1 Priors

We could not find a cumulative list of occurrences for all diseases so we decided a way to gauge the likelihood of a disease would be the number of results in a Google search (ex: "Alzheimer's" Medical). 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

We extract features from plain text data sources by building a document-term matrix. Each row represents a document (a disease article), each column represents a term (a word), and each element represents the number of times the term (column) appears inside the document/disease article (row). User queries are vectorized the same way we generate a row for the document-term matrix. Terms in user queries which were not present in any of the documents are discarded. All of our feature engineering is aimed at reducing the feature space because we identified this problem as having high variance (prone to overfitting) for most of our learning algorithms due to the training error being much higher than test error. We implemented five techniques: Tf-idf weighting, Stemming, Stop words removal, Document frequency based removal, and Non-alphabetical removal.

Tf-idf is a weighting scheme for word counts in the document-term matrix. Term frequencies are weighed according to:

$$tf_w = \begin{cases} 1 + log(count_w) \text{ if } count_w > 0 \\ 0 \text{ otherwise} \end{cases}$$

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 words that do not mean anything they only contribute to statistical noise. 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 Methods

We implemented several learning algorithms for this problem: Multinomial Naive Bayes, Logistic Regression, Cosine Similarity, and RandomTrees. The implementations of these algorithms are described below. We also implemented an SVM, Bayesian Network, Gradient Boosted Trees, and Artificial Neural Network, but these algorithms were implemented for the CS 221 portion of this project and are not described in this paper.

5.1 Naive Bayes

Naive Bayes algorithms are a set of supervised learning algorithms based on applying Bayes' theorem with the naive assumption of independence between every pair of features. Bayes theorem gives us:

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

Where y is the class label we predict and x is a feature vector we want to classify. Since we want to predict the class label with the highest probability given our feature vector, we have:

$$y = \arg\max_y p(y|x) = \arg\max_y \frac{p(y)p(x|y)}{p(x)} = \arg\max_y p(y)p(x|y)$$

The Naive Bayes assumption that the probability of a word appearing in a document is conditionally independent of any other words appearing in the document gives us:

$$y = \arg\max_y p(y)p(x|y) = \arg\max_y p(y)\prod_{i=1}^{n} p(x_i|y)$$

where x_i is the value of the i^{th} feature in x, which in our case is the occurrence of a given word in the document. We compute the likelihood of each disease and rank them based on their probability.

We implemented the Multinomial version of Naive Bayes which models $p(x|y)$ as a multinomial distribution. The distribution is parametrized by the vectors $\theta_y = \{\theta_1, ..., \theta_n\}$, where n is the number of words in our dictionary and y represents a disease. θ_{yi} represents the probability of word i occurring in a description for a given disease. By maximum likelihood, we estimate our parameters as $\hat{\theta_{yi}} = \frac{N_{yi}+\alpha}{N_y+\alpha n}$, where N_{yi} is the number of times feature i appears in descriptions of disease y in the training set and N_y is the total number of all words that occur in the descriptions of diseases y. α is a smoothing parameter and was empirically chosen to be one. The priors, $p(y)$, were not learned from the data set in our case, but rather were supplied as described in the "Priors" section, or were assumed to be uniform.

5.2 Logistic Regression

To use Logistic Regression in a multiclass classification problem we used a one versus rest scheme. Let C be the number of classes in the classification problem. Under this scheme, we train C Logistic Regression models, one for each class, each of which provides a probability that the input vector belongs to its class. We can order classes by probability from most likely to least likely and choose the most likely for prediction.
Each Logistic Regression model is trained with ”l2” regularization where $\lambda = \frac{1}{10}$. λ was chosen empirically. The mathematics behind each of the models is derived below.
Each model learns a parameter vector θ with the same length as the input vector. The probability each model predicts is given by:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + exp(-\theta^T x)}$$

Where x is the input feature vector. To find the value of θ we maximize likelihood of the data:

$$\ell(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m} (h_\theta(x))^{y^{(i)}} (1 - h_\theta(x))^{1-y^{(i)}} = \sum_{i=1}^{m} y^{(i)} log(h_\theta(x)) + (1 - y^{(i)}) log(1 - h_\theta(x))$$

Applying regularization:

$$\ell(\theta) = -\frac{\lambda}{2}\theta^T\theta + \sum_{i=1}^{m} y^{(i)} log(h_\theta(x)) + (1 - y^{(i)}) log(1 - h_\theta(x))$$

Now we derive the stochastic gradient ascent rule to find the maximum likelihood..

$$\begin{aligned}\frac{d}{d\theta_j}\ell(\theta) &= (y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})\frac{d}{d\theta_j}g(\theta^T x) - \lambda\theta_j \\ &= (y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})g(\theta^T x)(1-g(\theta^T x))\frac{d}{d\theta_j}\theta^T x - \lambda\theta_j \\ &= (y(1-g(\theta^T x)) - (1-y)g(\theta^T x))x_j - \lambda\theta_j = (y - h_\theta(x))x_j - \lambda\theta_j\end{aligned}$$

This gives us the stochastic gradient ascent rule:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} - \alpha\lambda\theta$$

5.3 Cosine Similarity

Cosine Similarity can be thought of as a nearest neighbors method. Cosine Similarity stores the document term matrix. Then, when the user enters a query, that query is also transformed into a vector of word counts. To predict a disease from a query, for each document (documents and diseases are interchangable in this model) the algorithm calculates a score for the document query pair, given by:

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

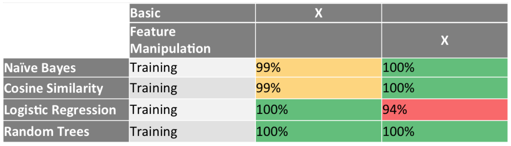
Where $\vec{d}$ is the vector representation of document d, $\vec{q}$ is the vector representation of the query.
This score represents the cosine of the angle between the query and the document vector. The algorithm sorts the documents by this score and returns this list as the result, with higher score meaning the document is a closer match to the query. The top scoring document is given by:

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

In order to avoid having multiple documents map to the same disease in the result, documents that map to the same diseases are combined in the document-term matrix before normalizing the document vectors. This is done by adding the document vectors element wise.

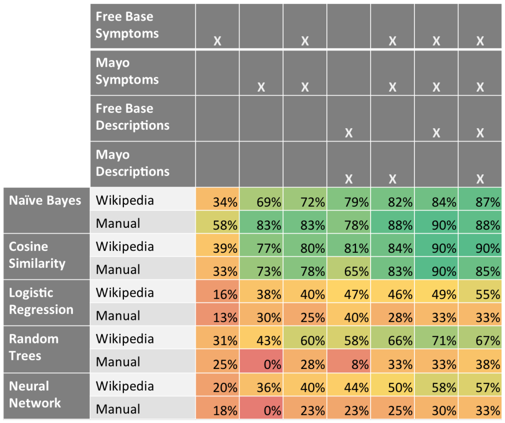
5.4 Random Trees

We used the scikit-learn implementation of random trees. We did some hyper-parameter tuning and found results stopped improving after about 100 trees. We also found that having no fewer than four examples in a leaf node was a good way to reduce tree depth and improve generalization. In addition, we tried a variant of Gradient Boosted Trees that has recently been popular in Kaggle competitions [2]. We found that the results were generally no better than the random forests we were already using, and the long training time (on the order of hours) discouraged us from trying to parameter tune an improved variant.



	Basic	X	
	Feature Manipulation		X
Naïve Bayes	Training	99%	100%
Cosine Similarity	Training	99%	100%
Logistic Regression	Training	100%	94%
Random Trees	Training	100%	100%

(a) Error on Training Set: Training on both Freebase and Mayo Clinic data and testing on the Freebase descriptions, these are our top five accuracy results.

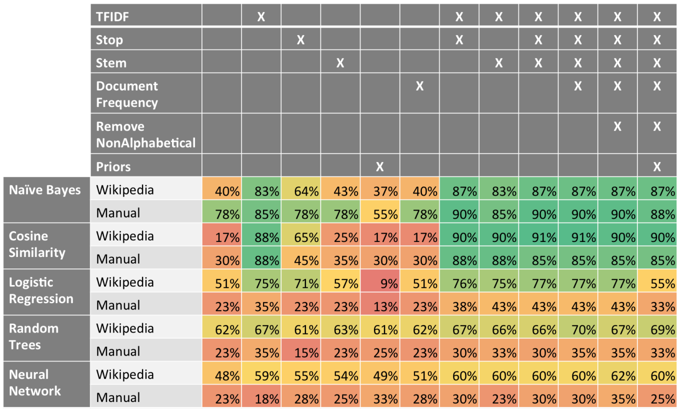


	Free Base Symptoms	X		X		X	X	X
	Mayo Symptoms		X	X		X	X	X
	Free Base Descriptions				X		X	X
	Mayo Descriptions				X	X		X
Naïve Bayes	Wikipedia	34%	69%	72%	79%	82%	84%	87%
	Manual	58%	83%	83%	78%	88%	90%	88%
Cosine Similarity	Wikipedia	39%	77%	80%	81%	84%	90%	90%
	Manual	33%	73%	78%	65%	83%	90%	85%
Logistic Regression	Wikipedia	16%	38%	40%	47%	46%	49%	55%
	Manual	13%	30%	25%	40%	28%	33%	33%
Random Trees	Wikipedia	31%	43%	60%	58%	66%	71%	67%
	Manual	25%	0%	28%	8%	33%	33%	38%
Neural Network	Wikipedia	20%	36%	40%	44%	50%	58%	57%
	Manual	18%	0%	23%	23%	25%	30%	33%

(b) Data Set Error Analysis: Using all of our feature manipulations with the given training data, these are the top five accuracy results. To provide a sense of significance of these results, Random Guessing would have a 0.021% top one accuracy while physicians average a 55.3% top one accuracy [9]. The best Naive Bayes model achieves a 63% top one accuracy on the manual test data and 67% top one accuracy for Wikipedia test data.

6 Results

As seen in figure 1b, the algorithm that yielded the overall the best results for the manual test set was Naive Bayes. TF-IDF was the feature manipulation that dramatically increased the results for almost all algorithms. We consider the manual test sets the best metric for success because they best represent an input that a user of this application would input.



	TFIDF		X					X	X	X	X	X	X
	Stop			X				X		X	X	X	X
	Stem				X				X	X	X	X	X
	Document Frequency						X				X	X	X
	Remove NonAlphabetical											X	X
	Priors					X							X
Naïve Bayes	Wikipedia	40%	83%	64%	43%	37%	40%	87%	83%	87%	87%	87%	87%
	Manual	78%	85%	78%	78%	55%	78%	90%	85%	90%	90%	90%	88%
Cosine Similarity	Wikipedia	17%	88%	65%	25%	17%	17%	90%	90%	91%	91%	90%	90%
	Manual	30%	88%	45%	35%	30%	30%	88%	88%	85%	85%	85%	85%
Logistic Regression	Wikipedia	51%	75%	71%	57%	9%	51%	76%	75%	77%	77%	77%	55%
	Manual	23%	35%	23%	23%	13%	23%	38%	43%	43%	43%	43%	33%
Random Trees	Wikipedia	62%	67%	61%	63%	61%	62%	67%	66%	66%	70%	67%	69%
	Manual	23%	35%	15%	23%	25%	23%	30%	33%	30%	35%	35%	33%
Neural Network	Wikipedia	48%	59%	55%	54%	49%	51%	60%	60%	60%	60%	62%	60%
	Manual	23%	18%	28%	25%	33%	28%	30%	23%	30%	30%	35%	25%

Figure 2: Feature Manipulations Error Analysis: Using all of our data sources with the given feature manipulations, these are the top five accuracy results.

7 CS221 Project

In general, we'd used this concept of disease classification across both our CS229 project and CS221 project. Here we provide a terse summary of the methods explored in our CS221 project. In general, our CS221 project focused on exploring relationships across our dataset and designing novel classification methods. We implemented LSA [3], and LDA [1] to perform automated document similarity, to automatically match articles across our datasets. Similarly, we used word2vec [10] to train our own word embeddings on our corpus and compared those to the results from GloVe [11] for symptom alias detection. Additionally, we implemented a Bayes Net for classification and a

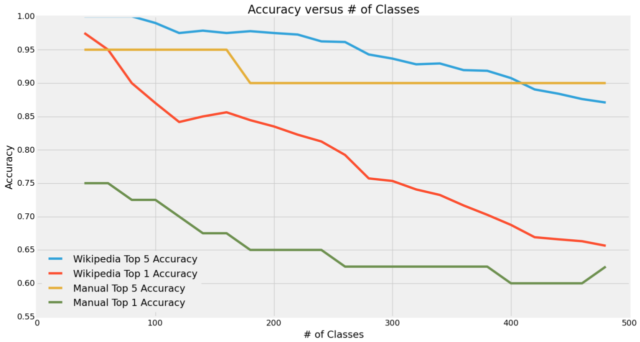


Figure 3: Using Naive Bayes with all of our feature extraction techniques, we train the model on a subset of the diseases and test only on diseases in that subset to produce the plot below (classes are equivalent to diseases).

novel method for applying convolution neural networks to text classification (by using our word embeddings to order our feature vectors, unlike previously published work on text CNNs [16] [6]).

8 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 55.3% success rate seen by physicians. This was done in two months without medical records or heavy clinical testing and we still had problems we wanted to tackle, such as finding priors and reducing medical jargon.

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] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *the Journal of machine Learning research* 3 (2003), pp. 993–1022.

[2] Tianqi Chen. "Higgs Boson Discovery with Boosted Trees". In: May 2014 (2015), pp. 69–80.

[3] 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`.

[4] 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.

[5] 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`.

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

[7] 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.

[8] 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.

[9] 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.

[10] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[11] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. 2014, pp. 1532–1543. URL: `http://www.aclweb.org/anthology/D14-1162`.

[12] 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.

[13] 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.

[14] 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`.

[15] 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.

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