Virtual Doctor

Ali Leino

December 18, 2016

1 Introduction

In this project we try to predict a patient's disease or condition given a list of symptoms. The diseases and conditions are referred to as *causes* in the rest of the document. The training data used is the MayoClinic database (http://www.mayoclinic.org/).

We do not have patient data available, so a variety of approaches were done to generate the validation data. Splitting the training data into testing and training sets does not work here - each class (cause) has exactly one training example. The same is true for cross-validation mehods.

It should be noted that in general, generating validation data is a bad idea - the results of the models depend entirely on the generation method chosen. Also, even the fact of knowing how the validation data is generated will influence how the model is chosen. Most of the time you can't make very specific assumptions about the data used for the prediction task, but in this case we know exactly what kind of data we are trying to predict.

The goal of this work is to find a model that gives good prediction results for unknown patient data. We assume (because we have to guess something) that a good model gives positive prediction results, given a variety of validation data sets.

2 Data gathering and preparation

The training data was scraped from MayoClinic by determining for each symptom what causes said symptom. Causes may be things other than diseases and conditions, for example tests and procedures. All causes other than diseases and conditions were left out of the data set used for training. The reasoning for this was that for example predicting that ankle pain resulted from ankle surgery does not usually make much sense in terms of usefulness for the patient.

Each training example consists of a symptom vector and a cause. The symptom vectors are transformed into indicator vectors, where each symptom is 1 if the symptom is associated with the cause, and 0 if not. The causes are also transformed into class labels starting from 0. The inverse transform of the symptoms and causes is also made possible, so that for a given indicator matrix, a natural language symptom list can be acquired.

The number of causes (and therefore examples) in the training set is $n = 490$, and the number of symptoms (features) is $m = 86$.

2.1 Validation data

Validation data was generated using four different generators. The *exact* generator simply samples examples from the training data. Usually this is frowned upon, but in this case the assumption that any future data is exactly the same is as good as any other assumption - we simply don't

know. The *one missing* generator samples examples from the training data and removes exactly one symptom from all of them. The *one addition* generator does the same, but adds one exactly one symptom. We generated 10^4 data points for each run of every algorithm, so that random fluctuations have less of an effect.

The *triangular* generator chooses a number from triangular distribution in the range $[-\sum_i x, \sum_i x]$. That is, it selects a number with the mean 0 and minimum and maximum the (negative) and positive number of symptoms in x. Then it either removes (negative number) or adds that number of symptoms. In the extreme cases it removes all the symptoms or doubles the number of symptoms.

3 Modeling

Since the data seemed most likely to be best suited for the kNN algorithm, it is discussed in detail here. We also discuss Naive Bayes classifiers and support vector machines in the results section. ## k-Nearest Neighbours k-Nearest Neighbours algorithm considers for each predicted its closest k neighbours. The distance can be defined using various measures, which are usually metrics. The algorithm classifies an instance to class if the majority of its k nearest neighbours are in the class. For $k > 1$ we may also define weights for the neighbours using some function of their distance.

If our validation data only contains exact instances of the training data, then clearly kNN is the best possible algorithm for the prediction task when $k = 1$. The closest neighbour is then always exactly at distance 0 from the predicted vector. The accuracy of this approach is 0.62 and not 1.0, because the training data has causes with the same symptom vectors.

Because for each cause there is exactly one training example, we don't gain anything by using $k > 1$. Let's first assume neighbours have uniform weighting. The effect of $k > 1$ is simply to add more tie-breaking situations to the algorithm, since each class may be in the neighbourhood only once. Then if we assume weighting scheme used is distance-based, that is neighbours with higher distance have smaller weight attached to their class. Then only the nearest neighbour counts, because the next one has a smaller weight, and each class can be only once in each neighbourhood. That's why we choose $k = 1$ for all the trials of the algorithm.

3.0.1 Distance measure

If we assume each of our indicator vectors x we are predicting have missing symptoms, and never extra symptoms with respect to the true cause, we can define a better distance measure for this exact problem than Euclidean or Manhattan metrics. We'll call this new distance measure M^*.

$$M^*(x, y) = \sum_{x_i > y_i} 1 + \frac{1}{m+1} \sum_{x_i < y_i} 1 \qquad \text{(where } x, y \in \mathbb{R}^m\text{)}$$

The sum $\sum_{x_i > y_i} 1 = 0$ when $x_i \leq y_i \forall i$, that is, for symptom vectors with no extra symptoms with respect to y. Each extra symptom in x raises the distance measure by 1. The second sum $\sum_{x_i < y_i} 1$ is incremented by 1 for each missing symptom in x. It is then scaled by $1/(m+1)$ so that its maximum value is $m/(m+1) < 1$. If x has no missing symptoms, both sums are therefore 0 and the distance is 0.

The distance measure defined here is not a metric, because it is not symmetric. This means that for scikit's implementation of the kNN-algorithm, we can't select internal algorithms that rely on the distance measure to be a metric. However, brute force approach works for non-metric distance measures.

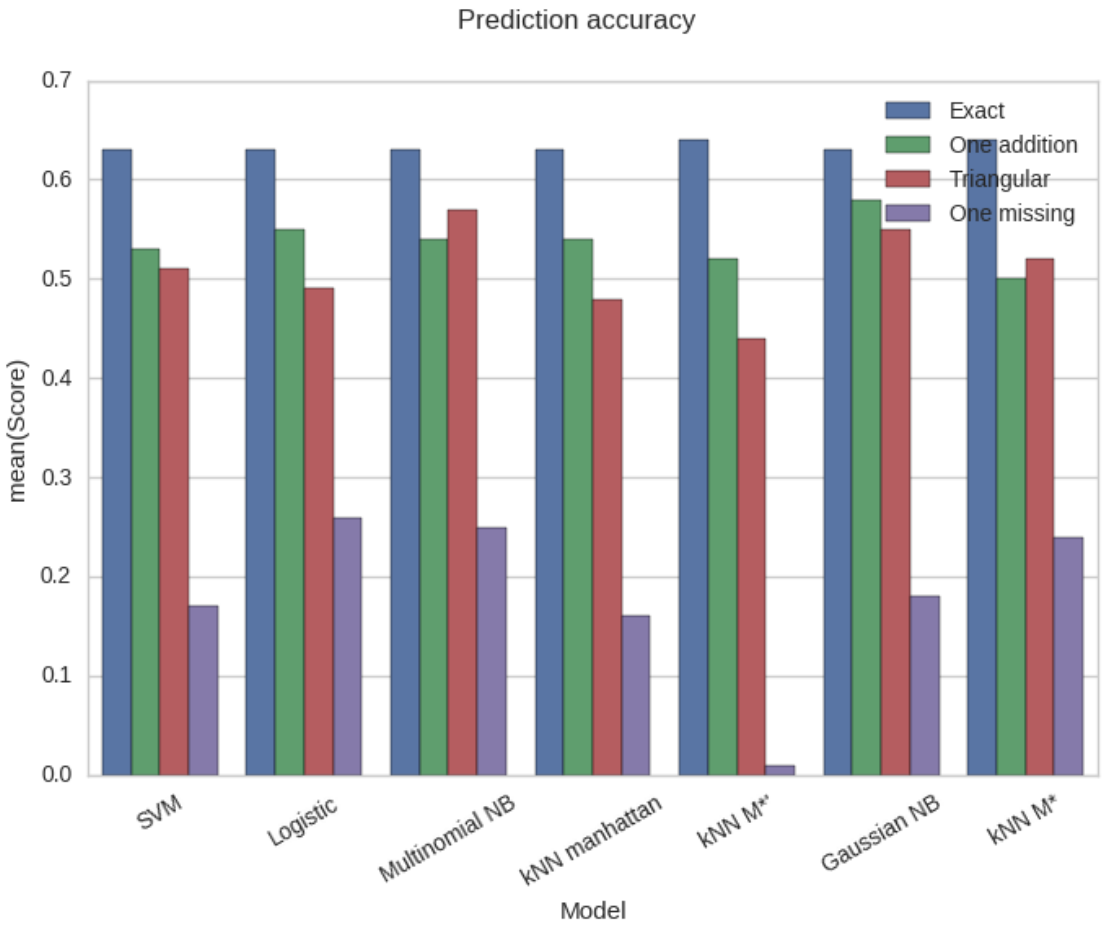
If we assume the inverse, that all symptom vectors have only extra symptoms and never missing ones, we can simply change the term that is scaled by $1/(m+1)$. We'll call this distance measure $M*'$. However, if the symptom vectors contain both missing and extra symptoms, and if additionally missing symptoms have the same value as extra symptoms, then Manhattan distance should be a better metric for the task.

If we assumed that the number of missing or extra symptoms is some exact value n, we could develop another distance measure that would be best suited for the problem. Generally it's trivial to construct models when the details of the validation data are known.

4 Results

In the following chart various algorithms were trained and validated using the previously defined validation generators.

The results of the various distance measures for kNN were close to what was expected. M^* did well for exact and one missing validations as expected. It also didn't suffer too much for one addition validation. Surprisingly $M*'$ measures results were not symmetric with respect to $M*$. But this is explained by the fact that true causes typically have a lot less symptoms than the number of features, so there are a lot more missing symptoms.

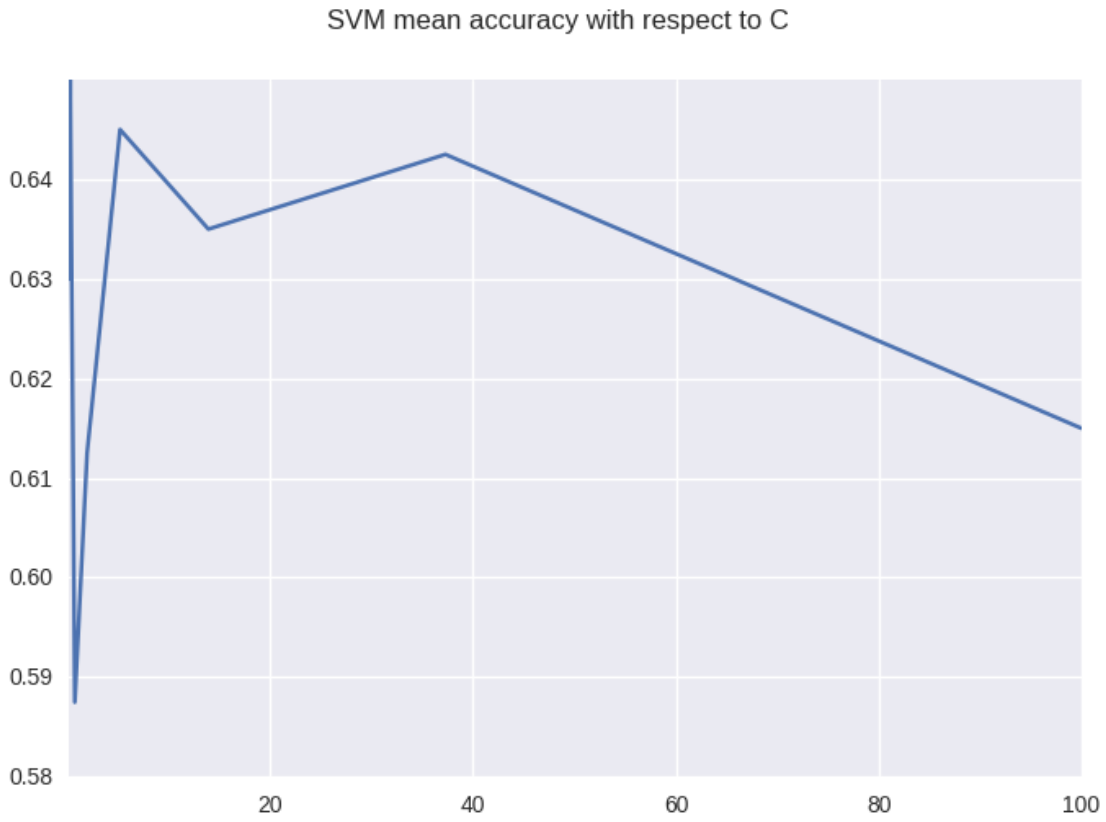


Model	Exact	One addition	Triangular	One missing	Mean
Logistic	0.6292	0.5403	0.5143	0.2806	0.491
kNN M*	0.6166	0.5276	0.5212	0.2758	0.485
Multinomial NB	0.6209	0.5269	0.5007	0.2819	0.483
kNN manhattan	0.6245	0.5659	0.4779	0.2068	0.469
SVM	0.6311	0.5699	0.4755	0.1946	0.468
Gaussian NB	0.6137	0.5626	0.478	0.2101	0.466
kNN M*'	0.624	0.5611	0.4266	0.0237	0.409

Here are the numerical results sorted by the mean of the different validation schemes. Surprisingly SVM and Logistic classifiers did better than kNN* with Manhattan metric for exact inputs from the training data. This should not be possible unless the two algorithms systematically make better choices in tie-braking situations.

Naive bayes models assume that the features (symptoms) are independent. In real life this is of course not true - many symptoms are dependent on one another. However, for the validation generators chosen multinomial naive bayes classifier does quite well. This is likely because given a symptom and a set of causes having the symptom, some causes may be more probable given the symptom. The random addition and removal of symptoms still may leave a lot of probabilistic information about the true symptoms and the true missing symptoms left. Also, the distribution of removal and addition more is closer to a multinomial distribution than a gaussian one. It's likely that if another validation scheme were added, using a gaussian distribution, then gaussian bayes classifier would do better.

The values for SVM are done using the regularization parameter $C = 1$. A plot of how C affects to the mean of SVM accuracy is depicted below. Due to time constraints, here prediction is only done for 100 symptom vectors.



5 Conclusions

Most of this work was heavily influenced by the fact that we know how the validation data is generated, and we could freely choose how it is done. In real life this is rarely the case. When our premise is so fundamentally strongly biasing our models, an interesting idea would be to use generated data for the training data as well. This would make certain machine learning algorithms better suited for the task. Many models depend on seeing many examples of the same classes.

Logistic regression had the best mean accuracy, but not by a very large margin. kNN with M* distance measure and multinomial NB were less than 1% behind in the mean accuracy. The rankings of the algorithms in terms of mean accuracy could be easily reversed were the validation generators chosen differently. It is not known if these results can be generalized outside of these validation methods.