Cassandra's Twin: What Does the Data Predict?

Ioana Boier
ioanaboier@gmail.com

September 5, 2016

Abstract

This paper examines what type of insights can be extracted from a given dataset. This analysis is intended to encourage those collecting data to engineer their collection processes with data usability in mind. It is also meant to caution researchers against the pitfalls of trying to squeeze out more information from data than the data actually warrants. The ideas presented are illustrated through a concrete example of using CDC's Behavioral Risk Factor Surveillance System (BRFSS) dataset to identify risk factors associated with hypertension. A complete end-to-end predictive modeling process is described in detail from data conversion and pre-processing, to model building, and interpretation of results. Python code to process the data as well as Matlab code to build, train, and evaluate the models are included in full and are sufficiently general in nature to be used for other analyses beyond hypertension.

1 Introduction

Most often, when given a problem to solve, we gather relevant data from various sources to come up with a solution. In this paper we address the reverse question of what type of insights could be extracted from a prescribed dataset. This analysis serves two main purposes: (a) as guidance for those collecting data (through surveys or other generic means) to consider the usability of the information collected and (b) to caution researchers against the temptation to squeeze out more information than the data actually warrants.

To make the illustration concrete, let us consider the topic of predictive modeling in the context of the Behavioral Risk Factor Surveillance System (BRFSS) dataset [5]. The goal is to demonstrate that although it may be possible to design predictive models that can be trained successfully (as measured by standard scientific criteria) on general-purpose data, such models may be of limited use in practice without additional problem-specific information.

The paper is organized as follows: Section 2 provides details regarding the nature of the data and it includes a discussion of the challenges it raises in the context of prediction; Section 3 combines a review of literature with a categorization of the main types of predictive modeling that have been performed on such data; a concrete prediction model for hypertension is developed and analyzed in detail in Section 4; final remarks conclude the paper in Section 5.

2 The BRFSS Data

Administered by the Centers for Disease Control and Prevention (CDC), the BRFSS survey is designed to collect prevalence data from adult U.S. residents pertaining to risk behaviors and preventive practices that could impact their health. It comprises an annual component (i.e., questions asked every year), a biannual one (i.e., question asked only every other year), plus optional information that is gathered at the discretion of each state. The main goal of the survey according to the CDC is to collect data on actual behaviors that could be used in the development and evaluation of health promotion and disease prevention programs.

After collection and final aggregation by the CDC, the data appears in tabular format with each row representing a survey record (i.e., a sample) and each column corresponding to a reported variable. The

values of the variables are either directly coded based on the participant responses or calculated from values of other directly coded variables. For example, $\_RFBMI5$ is a calculated categorical variable that can take values $1, 2$, or 3 depending on whether another real-valued variable $\_BMI$ falls in the overweight range (greater than 25.0) or not or it is missing. In its own turn, the $\_BMI$ variable derives its values from the weight and the height reported by each survey participant.

Certain characteristics of the data are important for predictive modeling purposes:

1. **Recall bias:** the data is only as accurate as the recollections or estimates given by people participating in the survey. This implies that all data, including the calculated measures (such as the BMI in the previous example) can be hampered by inaccuracies. The degree of noise can vary within subgroups (e.g., women tend to underreport their weight by a larger percentage than men [10]).

2. **Selection bias:** the sample is selected from the adult, civilian, non-institutionalized population of the U.S. which means that some information such as the number of sick days could be underestimated. Also, any conclusions drawn from this data may be difficult to generalize to groups living outside the U.S.

3. **Inconsistency:** data may not be directly comparable between surveys for a variety of reasons (e.g., change in weighting methodology of the survey itself, change in the wording of questions, measurement unit discrepancies (e.g., fruit intake, vegetable servings), collection lags (some data is collected annually, some every other year).

4. **Memorylessness:** since there is no identification information, the progression of specific individuals or groups over time cannot be monitored based on this data.

3 Predictive Modeling Based on the BRFSS Data

The scientific literature abounds with BRFSS data analyses of various kinds, including predictive models. The topics broadly fall into the following categories:

1. **Diagnostic tools:** a significant number of studies have tackled the problem of identifying the most relevant risk factors that can "predict" a particular outcome. The outcomes can be considered at the individual level (e.g., what health, environmental, or socio-demographic factors are most associated with cancer [11] or cancer survival [19], diabetes [24], depression [25], hypertension [23], obesity [10, 8]) or at the level of communities (e.g., level of emergency preparedness [1]). These are typically implemented using supervised learning methods by defining the response variable(s) as the answer to one or a combination of survey questions and using a subset of the remaining questions/answers as input variables.

2. **Trend analysis:** some demographic aspect such as education levels or disease prevalence [14, 7] are typically predicted at state level. Subsequent analysis is typically conducted to uncover the endogenous characteristics of states that outperform their predicted values by significant amounts and ultimately, to understand the environmental, policy, or other factors that set them apart so they could be replicated and applied elsewhere.

3. **Gap filling and data correction:** survey and/or self-reported data have deficiencies, as previously discussed. Yet, when supplemented with additional information or corrected for bias, they can drive powerful insights. For example, in [15] the authors use BRFSS data on fruit and vegetable consumption to determine percentage of state populations that meet federal recommended intakes. Models are not built directly from the BRFSS data which used different units over time, but from a separate CDC database of dietary recalls (NHANES [17]) and is subsequently applied to BRFSS information to fill in gaps and allowing monitoring state-level progress towards national goals. Another example pertaining to BRFSS data correction is provided in [10] where regression models that can estimate the bias between self-reported and professionally measured weight and height are developed using the NHANES dataset and subsequently used to correct the BRFSS information and predict obesity prevalence.

4. **Collective supervision:** some authors are using the BRFSS data to correlate social media messages with survey outcomes and to transform unsupervised topic models into collectively supervised, i.e., using existing aggregated survey data to inform the inferred topics [3].

5. **Scenario simulation:** models based on the BRFSS data are developed to simulate policy outcomes (e.g., how would changing Medicare insurance eligibility criteria affect various states [2]).

4 Hypertension: Pervasive and Dangerous

According to the World Heart Federation [22], hypertension is defined as systolic blood pressure at or above 140mmHg and/or diastolic blood pressure at or above 90mmHg. There are almost a billion people worldwide who suffer from it and this number is estimated to increase by roughly 50% over the next decade.

Once it sets in, hypertension is straightforward to measure and diagnose. It is known or suspected to be a major risk factor in cardiovascular and other serious conditions. Yet, its own risk factors are not fully understood. For example, *essential hypertension* could be related to genetic predispositions, lifestyle choices (stress, diet, exercise) and other factors that have not been clearly identified at this point. *Secondary hypertension* can be caused by factors like diseases, medications, etc., also not fully understood.

4.1 Problem Statement

The main goal of this exercise is to investigate the opportunities and limitations offered by the BRFSS survey data to identify risk factors associated with hypertension. An additional goal is to provide a general analysis blueprint for anyone wishing to make similar characterizations of other conditions based on this data.

To solve this problem, we first build a prediction model similar to the diagnostic-type tools mentioned in the previous section. Second, we analyze its usefulness and limitations.

4.2 Data Curation

There are several steps in the data curation process:

1. Raw data retrieval: there are many years worth of data in the BRFSS database. Due to differences in survey methods, data prior to 2011 and after are not directly comparable. Data is provided in two formats (SAS and TXT) which may require conversion. A small R script useful for converting data from SAS format to CSV is shown in Appendix A.

We consider two BRFSS annual survey datasets from 2013 and 2015 (the 2014 dataset does not include the high blood pressure question used to define the response variable). The 2015 set is used for model building and testing and the 2013 one for additional out-of-sample testing. There are 491773 records of 336 variables in the 2013 data. The raw 2015 data consists of 441456 records of 330 variables each. Unless otherwise noted, any further references to BRFSS data in this section pertain to the 2015 dataset. The 2015 codebook published by the CDC contains details regarding the values and ranges of each variable [4].

2. Feature selection and cleanup: From a machine learning standpoint, our problem is a binary classification one. To solve it, we need to define a set of predictor variables that will be used to predict the class of the response variable (in this case, whether a person has high blood pressure or not). Such a formulation may seem odd at first since one could just consider measuring a person's blood pressure to determine whether they are hypertensive or not. However, the actual goal is hypertension prevention and in this context, the hope is that finding the factors associated with the condition could offer some insight into what may lead to it. The Python notebook used to load the CSV files containing the raw data and to select and process the features of interest is also included in Appendix A.

Response variable: corresponds to the calculated variable $\_RFHYPE5$, i.e., "Adults who have been told they have high blood pressure by a doctor, nurse, or other health professional". For readability, we

recode its name as $HYPERT$ for the remainder of the paper. The original values for this variable were 1 - NO, 2 - YES, 9 - Refused or missing.

Input variables: are selected in steps, an initial manual one and one or more automatic ones. In a first pass, taking domain knowledge into account, we reduce the original 330 variables to 31 corresponding to raw or calculated information that is suitable to use for the problem at hand (see Table 1). Specifically, we choose to ignore outright several types of data:

(a) Irrelevant: variables pertaining to date and time of the survey, geographic information, landline or cellphone, healthcare provider, household composition, wears seatbelt, etc.

(b) Duplicate: between calculated variables and raw ones, only one type referring to a particular topic is kept (e.g., BMI is used, but $WEIGHT$ and $HEIGHT$ are left out; $ALCOHOL$ is kept, but average alcohol consumption, binge drinking, and related ones are left out).

(c) Medically similar or presumed unrelated: conditions for which hypertension is a known risk factor like stroke and myocardial infarction are removed from the analysis. General statements regarding the health of a person like poor health or good health are also removed. All cancer-related variables are left out (several other types of illnesses are kept, but we eliminate cancer due to the complex relationships between malignancy and hypertension [16]) which would require more careful model engineering. See also Section 4.4 for discussion regarding prediction of one condition in the presence of another.

(d): Undersampled: variables for which the overwhelming frequency of response was "Not asked or missing" (e.g., estimated intensity of first activity).

Cleanup: before proceeding to the automated variable selection and modeling phases, the data corresponding to the selected variables (input and output) is cleaned as follows:

YES–NO responses: all values representing the answer YES to a survey question are mapped to the value 1 and all values representing the answer NO to a survey question are mapped to the value 0. For example, as mentioned previously, the response variable's original values were 1 - NO, 2 - YES, 9 - Refused or missing. After the cleanup, they become 1 - YES (meaning 'has hypertension'), 0 - NO, and 9 - Refused or missing. This is more in line with the binary classification best practice of encoding the presence of something we're looking for (e.g., hypertension) with 1 and its absence with 0. While not necessary, it allows for easier inspection of the data without having to constantly refer to the codebook.

Unavailable responses: all numerical values corresponding to DON'T KNOW, REFUSED, MISSING, or NOT ASKED (typically 7 or 9 in the raw data) are mapped to the value 9. All blank entries in the raw data are mapped to -99.

Categorical variables: with the exceptions noted in the previous two categories, all categorical values are kept as in the original data.

Numerical variables: with the exception of missing values which are coded as previously described, all values are kept in original form. Note that the only numerical variables used in this example are the ones pertaining to food intake; they are all reported in the same unit (times per day) and take values in the same range $(0 - 9999)$.

Following the standardization of variable values, we proceed to remove all records that have any blank entries in their composition. We also remove all records that have missing information for the hypertension question. This leaves a total of 335583 records from the original set. An alternative would have been to keep missing values and handle them at modeling time, but for the purpose of this exercise we decided to operate without missing data from the start.

After cleaning the data, the model building can proceed. As we will see in the next section, the models themselves can be used to automatically perform further feature selection.

Index	BRFSS Name	New Name
1	_AGE_G	AGE
2	SEX	SEX
3	_RFBMI5	BMI
4	MARITAL	MARITAL
5	_EDUCAG	EDUC
6	_RACE	RACE
7	_INCOMG	INCOME
8	_TOTINDA	EXERCISE
9	_RFCHOL	CHOLEST
10	_RFDRHV5	ALCOHOL
11	_ASTHMS1	ASTHMA
12	MEDCOST	MEDCOST
13	CHCCOPD1	COPD
14	ADDEPEV2	DEPRESSION
15	RENTHOM1	RENTOWN
16	EMPLOY1	EMPLOY
17	INTERNET	INTERNET
18	SMOKE100	SMOKE
19	FTJUDA1_	JUICE
20	FRUTDA1_	FRUIT
21	BEANDAY_	BEAN
22	VEGEDA1_	VEGGIE
23	DIABETE3	DIABETES
24	HLTHPLN1	HEALTHPLAN
25	HAVARTH3	ARTHRITIS
26	CHCKIDNY	KIDNEY
27	VETERAN3	VETERAN
28	BLIND	BLIND
29	DECIDE	COGNITION
30	DIFFWALK	DIFFWALK
31	USENOW3	TOBACCO

Table 1: Input features selected by manual process. New names are assigned for readability.

4.3 Model Building

The choice of learning model always entails trying out multiple methods. Each has its own assumptions and performance characteristics and it makes sense to explore several such methods to produce the best one (or combination) for a given dataset. In the context of binary classification, two good candidates are logistic regression and decision trees. While logistic regression assumes a hyperplane decision boundary that separates the data, decision trees partition the feature space into axes-aligned hyper-boxes. Both methods are quite fast and easy to implement, making them suitable benchmarks for more complex models. In this section, we illustrate both.

4.3.1 Logistic Regression

In logistic regression, the hypothesis function has the form:

$$h_\beta(x) = g(\beta^T x)$$

where g is the logistic function:

$$g(z) = \frac{1}{1+e^{-z}}$$

The output of the hypothesis $h_\beta(x)$ can be interpreted as the estimated probability that $y = 1$ given a new input data point x. Imagine that for some new record x in the 31−dimensional space of the features defined in Table 1 we have $h_\beta(x) = 0.7$. This number would indicate that the probability that the corresponding person has hypertension is 70%, i.e., $p(y = 1|x; \beta) = 0.7$. Training such a model means finding the values of the parameter vector β that maximize the log-likelihood function:

$$\ell(\beta) = \sum_{i=1}^{n} (y_i log p(x_i) + (1 - y_i) log(1 - p(x_i)))$$

with $p(x) = p(y = 1|x; \beta) = h_\beta(x)$ and x_i samples in the training data. The parameter vector β has $m + 1$ components (coefficients), where m is the number of predictor variables ($m = 31$ in this example) and one additional coefficient for the intercept:

$$h_\beta(x) = g(\beta^T x) = g(\beta^0 + \beta^1 x^1 + \cdots \beta^m x^m)$$

Statistical packages like R and Matlab that implement logistic regression typically return not only the estimated values of these coefficients, but also other statistics useful in the analysis. In particular, each coefficient comes with a t-statistic to test the null hypothesis that the coefficient is zero against the alternative that it is different from zero, as well as a corresponding p-value. Hence, if one of these p-values is greater than 0.05, it can be concluded that the corresponding feature is not significant at the 5% significance level given the other features in the model. As shown in Section 4.4, this information can be used to automatically select relevant features. Even when logistic regression is not the final predictive model, it can be used as a first step to reduce the feature space.

4.3.2 Decision Trees

An important aspect not mentioned in the previous section was the linear dependence implied by the hypothesis function. While it is possible to enhance the dependence formulation to include higher order terms, such enhancement is manual and requires knowledge of the data to avoid blindly adding a large number of terms. Decision trees make no linearity assumptions about the data. They are appealing as they accommodate features of various kinds (numerical, categorical, etc.) without explicit need for extensive pre-processing. Additionally, they offer an intuitive representation that is easier to grasp compared to regression coefficients.

The leaf nodes in the tree correspond to class labels $(0, 1$, i.e., hypertension or no hypertension, for our example). Each interior node corresponds to a predictor variable and a binary decision rule on the possible values of that variable (e.g., obese or not, fruit intake above or below a threshold). The most relevant feature with respect to some measure of relevance appears at the root of the tree. The tree can be mapped to a set of rules starting at the root and following each branch towards a leaf. Once a decision tree is built, new data can be classified based on its features by simply following the flow of control defined by the tree. Decision trees can also be used for feature screening by simply considering the most important features as being the ones appearing closest to the root (no coefficients or p-values to analyze in this case) so they too, can be used for feature selection.

4.4 Results and Discussion

4.4.1 Logistic Regression

Matlab code for implementing a logistic regression classifier given training data in matrix format (rows correspond to samples and columns to variables; the response variable appears in the last column) is shown in Appendix B. The function *trainLogisticClassifier()* returns a trained classifier and its validation accuracy based on k-fold cross-validation. Besides the data itself, the function also takes as input the name of the variables ($headers$), corresponding indicators of which variables are categorical and which are not, and the value k for the k-fold validation. The trained classifier can subsequently be used to classify new data. In particular, it is applied to data from another year (2013).

The coefficients β of the classifier are calculated and returned along with the corresponding p-values. The regression is performed in several iterations to remove insignificant features:

1. Use the initial 31 features manually selected and evaluate the p-values for each of the corresponding regression coefficients. Eliminate features with p-values above 0.05. This step leads to a dataset of 329016 samples each with 24 features that are used in the second iteration.

2. Use the reduced set of features from the previous iteration to compute new regression coefficients and p-values. If necessary, repeat this step several times until all p-values are below 0.05.

The coefficients obtained after two iteration and their p-values are shown in Appendix C. Note that the categorical features in the data are automatically encoded as indicator variables with their first value used as reference. For the indicator variables showing insignificant coefficients at the 5% level, we can either perform additional lumping of categories (e.g., from six marital status categories down to two: married nor not) or we can completely remove the original variable if many of its indicator components are deemed insignificant.

The final set of 24 variables is shown in Table 2.

Index	BRFSS Name	New Name
1	_AGE_G	AGE
2	SEX	SEX
3	_RFBMI5	BMI
4	MARITAL	MARITAL
5	_EDUCAG	EDUC
6	_RACE	RACE
7	_TOTINDA	EXERCISE
8	_RFCHOL	CHOLEST
9	_RFDRHV5	ALCOHOL
10	MEDCOST	MEDCOST
11	CHCCOPD1	COPD
12	ADDEPEV2	DEPRESSION
13	INTERNET	INTERNET
14	SMOKE100	SMOKE
15	FTJUDA1_	JUICE
16	FRUTDA1_	FRUIT
17	DIABETE3	DIABETES
18	HLTHPLN1	HEALTHPLAN
19	HAVARTH3	ARTHRITIS
20	CHCKIDNY	KIDNEY
21	VETERAN3	VETERAN
22	BLIND	BLIND
23	DIFFWALK	DIFFWALK
24	USENOW3	TOBACCO

Table 2: Remaining input features selected automatically after logistic regression.

4.4.2 Decision Trees

Matlab code for implementing a decision tree classifier given training data in the same matrix format as before is shown in Appendix B. Similar to the logistic case, the function *trainTreeClassifier()* returns a trained classifier and its validation accuracy based on k-fold cross-validation. Besides the data itself, the function also takes as input the variable *headers*, the categorical flags, and the value k for the k-fold validation. The trained classifier is subsequently applied to the 2013 dataset.

The function *cvLoss()* in Matlab is used to determine the largest pruning level that achieves a value of the classification error close (within some predefined threshold) to the error returned by the full tree and we proceed to prune the tree to this level. Figures 1 and 2 show the classification trees before and after pruning.

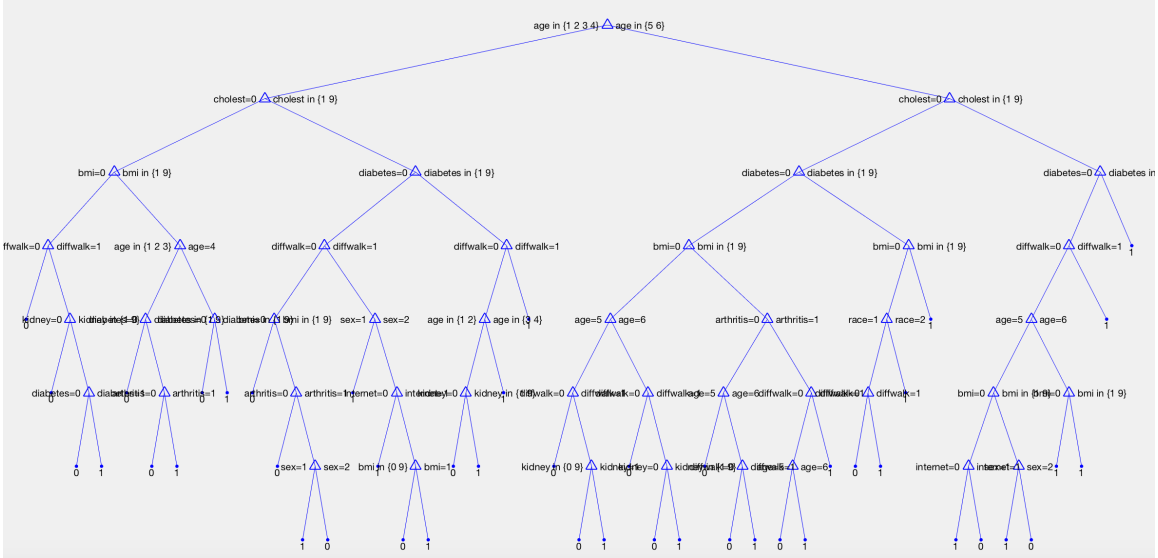

Figure 1: Full classification tree using 24 predictor features as input.

4.4.3 Performance Measures

Validation accuracy from 10-fold cross-validation on the 2015 data is summarized in Table 3; confusion matrices and related measures for the 2013 test set are shown in Fig 3.

Method	# Features	Accuracy (%)
Logistic	31	71.7
Logistic	24	71.5
Logistic	7	70.9
Tree 22 levels	24	70.9
Tree 12 levels	24	70.9

Table 3: Accuracy of classification models based on 10-fold cross-validation using 2015 data.

In terms of overall prediction accuracy, the methods presented here are comparable to those reported in recent literature [23, 12]. Note the larger specificity values compared to recall ones – these are likely a result of the class imbalance in the data (there are a total of 164023 samples in the hypertensive class, whereas the healthy class contains 26% more, i.e., 207304 samples). More sophisticated sampling of the original data would help alleviate this issue. Also note AUC or area under the Receiver Operator Characteristic (ROC) curve at over 70%. As pointed out in the Framingham study **??**, while no specific standard exists many screening tools in use have this statistic at or above 70%. For comparison, the AUC for the more robustly setup Framingham score was reported at 78.8% **??**.

4.4.4 Further Analysis

Up to this point we have two relatively simple models of hypertension prediction that perform similarly to one another and at the same level or not much worse than more complex models (e.g., neural networks) reported in the literature [23]. We observe age to be a main discriminator based on the tree model, along with cholesterol, diabetes, and BMI. Difficulty walking, arthritis, and gender also seem to have some predictive powers according to these models, although it is not clear whether some, like difficulty walking, are consequences of hypertension rather than risk factors.

A next step could be to drill further down into the selected features and find out if more sophisticated models would be better suited to capture the true decision boundaries in the data. Yet, another

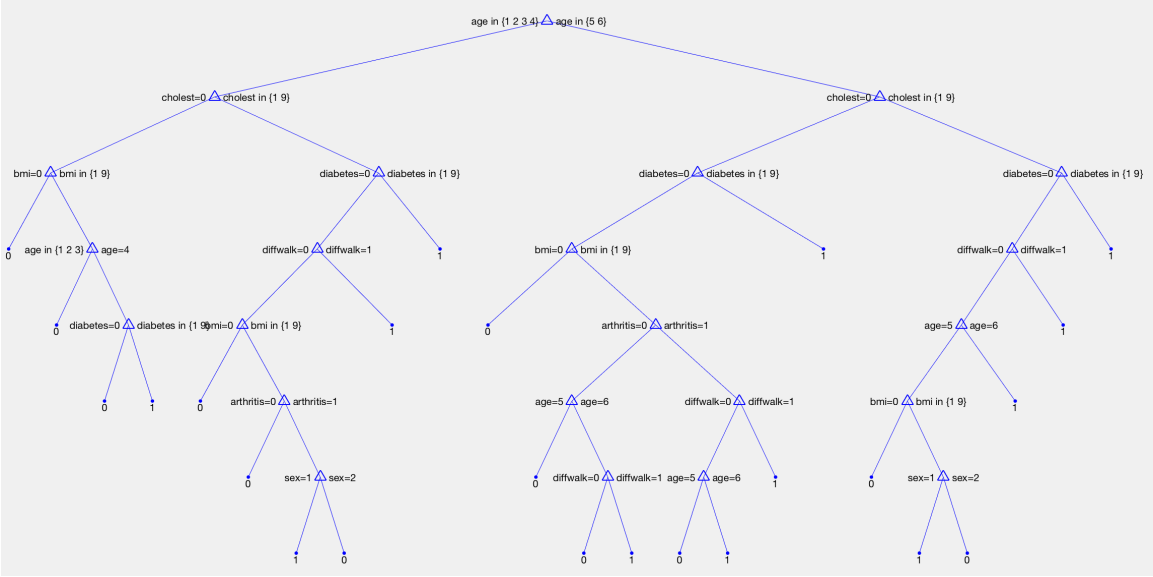

Figure 2: Pruned classification tree.

possibility would be to pause and ask if the BRFSS data by itself without additional information can ever supply us with the necessary information to build a usable model of risk factors for hypertension.

One of the first hits that comes up in Google when researching hypertension-related predictive models is the Framingham Heart Study [9]. It includes an online risk score calculator for the incidence of hypertension which is based on an earlier study [18]. The calculator takes into account age, gender, BMI, blood pressure, smoking, and parental hypertension information to predict a person's risk of hypertension within the next $1, 2$, and 4 years. The fact that four of their six predictor variables overlap with the ones that surfaced from our models is encouraging (the other two are not available in the BRFSS data). Interestingly, however, their study is based on non-hypertensive individuals aged 20 to 69 *without diabetes at baseline*. In our setup, diabetes appears as an important discriminator in the decision tree. Thus, it would be natural to wonder about the relationship between hypertension and diabetes. Why was it left out in the Framingham study? Should we have done the same? It turns out that the two frequently occur together and that there is actually substantial overlap in their etiology [6]. While a full understanding of common causes and disease mechanisms remains an open problem, it seems reasonable to question any model that would use diabetes as a "predictor" of hypertension. Hence, our next step is to see what happens if we switch the roles of the two in the tree model: make DIABETES the response variable and HYPERT one of the inputs. The results are shown in Figure 4. Sure enough, hypertension appears as the top discriminating feature in the new tree and the prediction accuracy goes up to 85.8%!

Next, let us return to the original hypertension model (i.e., use HYPERT as the response variable) and remove from the data all records of individuals who are diabetic (for whom DIABETES $= 1$). In other words, we would like to see how the predictions would change if we controlled for diabetes just like the Framingham study did. As expected, the top features no longer include diabetes, but age, cholesterol, and BMI remain among the most discriminative. The 10-fold cross-validation accuracy is 70.2% for this tree (25 levels) and 70.3% for the pruned version (9 levels). The latter is shown in Figure 5.

Finally, we can also use the 7 features of the tree in Figure 2 as inputs to the logistic regression. These are: AGE, CHOLEST, BMI, DIABETES, DIFFWALK, ARTHRITIS, and SEX. The performance of the logistic classifier remains virtually unchanged (71.0%) while the feature set is much reduced. The coefficients of this model are shown in Appendix C.

To a certain extent, we can view these last results as a quick, back-of-the-envelope confirmation of the *association* between hypertension and these other factors that have also been highlighted by more complex, more in-depth prior studies. However, we cannot confuse association with causation and we cannot say, based on these models alone, that they have the power to predict the incidence of

Confusion Matrix Structure		
TN	FP	TN+FP
FN	TP	FN+TP
TN+FN	FP+TP	TN+FN+TP+FP

Tree, 24 features		
160,824	46,480	207,304
61,147	102,876	164,023
221,971	149,356	371,327

Logistic, 24 features		
162,097	45,207	207,304
60,042	103,981	164,023
222,139	149,188	371,327

Accuracy	71.0%
Precision	68.9%
Recall	62.7%
Specificity	77.6%
AUC	70.1%

Accuracy	71.7%
Precision	69.7%
Recall	63.4%
Specificity	78.2%
AUC	70.8%

Figure 3: Confusion matrices for classification with 24 features and associated measures using 2013 data.

hypertension, no matter how sophisticated the underlying learning algorithm becomes. The BRFSS data simply does not support such a conclusion. Even if parental history and blood pressure (the missing predictors present in the Framingham scoring method) were included in the survey, we would still lack a critical piece: the ability to construct an "aging" model for those who develop hypertension versus those who don't. Anyone who's read or watched Moneyball [13] can recognize that the player stats crunched into predictive models by Billy Beane and his team were not generic: they were associated with identifiable people whose performance over time, in different environments (e.g., Fenway Park vs. Dodger Stadium) was traceable.

We contend that we need more such data to develop usable clinical models. The actual identity of the people surveyed need not be known, but interviewing the same person over time and being able to follow their data is critical.

5 Conclusions

An ever increasing number of institutions and individuals collect and make available data for almost all measurable aspects of life. When observational data is used to make predictions, especially health-related ones, it is important for model developers to understand the limitations of the data and clearly state the effects of those limitations on the capabilities of their models. In particular, prediction and etiology must not be mistaken for one another [21]. Predictive models of the kind highlighted in this paper or in the references surveyed are purely descriptive in nature. If they are good, they can be used to "guess" that a certain outcome is dependent on a set of input variables by learning from training data. Such models do not have the power to establish causality and the language used to present them should carefully avoid such claims. Additionally, when multiple conditions are present, domain-knowledge is necessary to control for each condition independently. Finally, the incidence of a medical condition is difficult to predict from static, point-in-time snapshots that do not provide a traceable time dimension to the data.

According to Greek mythology, beautiful Cassandra could predict the future with hundred percent accuracy and yet, nobody believed her. While that seems like a very frustrating situation, we cannot help but wonder whether the opposite could be equally or even more harmful. Imagine Cassandra's twin who makes much less accurate predictions, yet everybody believes her. In many domains, this is a situation we can hardly afford.

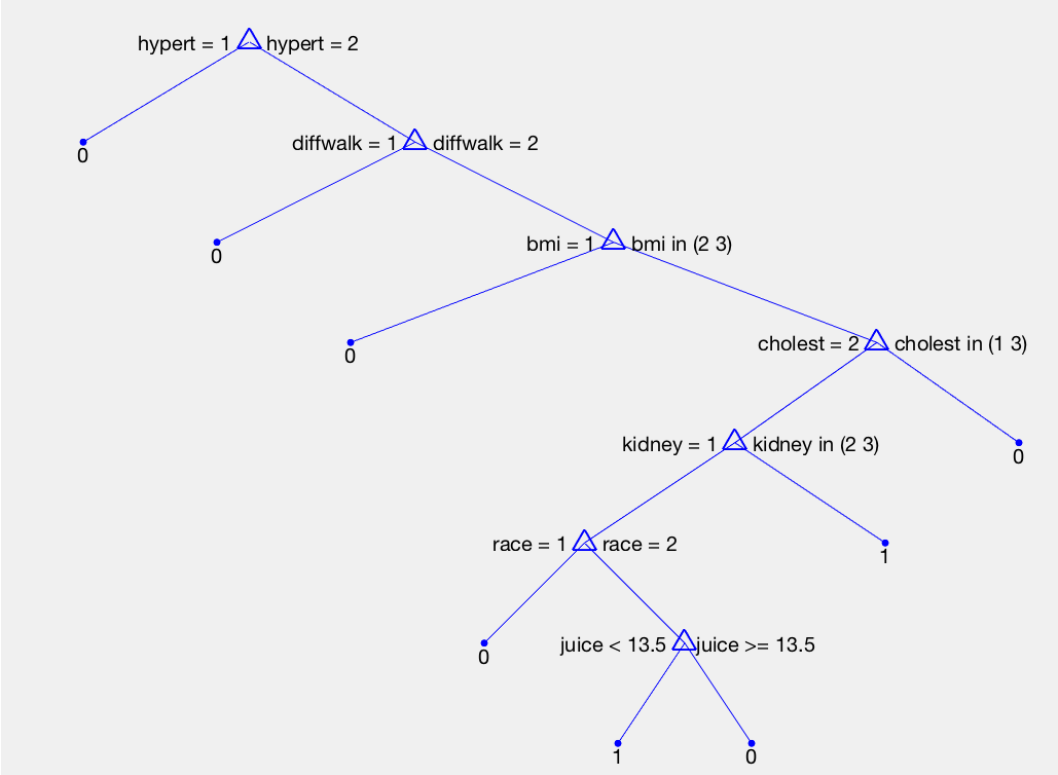

Figure 4: Pruned classification tree after reversing the roles of hypertension and diabetes variables. This tree predicts diabetes labels from a set of input features including hypertension.

References

[1] Ablah, E. et al., "Factors Predicting Individual Emergency Preparedness: a Multi-state Analysis of 2006 BRFSS Data", Biosecur Bioterror., 2009, 7(3):317-30.

[2] http://slideplayer.com/slide/4941002/

[3] , Benton, A. et al., "Collective Supervision of Topic Models for Predicting Surveys with Social Media", Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[4] http://www.cdc.gov/brfss/annual_data/2015/pdf/codebook15_llcp.pdf, "BRFSS 2015 Codebook Report".

[5] http://www.cdc.gov/brfss/, "Behavioral Risk Factor Surveillance System".

[6] Cheung, B.M and Li, C., "Diabetes and Hypertension: Is There a Common Metabolic Pathway?", Curr Atheroscler Rep., 2012, 14(2): 160–166.

[7] , Fan, A.Z. et al., "State Socioeconomic Indicators and Self-Reported Hypertension Among US Adults, 2011 Behavioral Risk Factor Surveillance System", Prev Chronic Dis, 2015,12:140353.

[8] , Finkelstein, E.A. et al., "Obesity and Severe Obesity Forecasts Through 2030", American Journal of Preventive Medicine, 2012, 42(6): 563–570.

[9] framinghaheartstudy.org, "Framingham Heart Study".

[10] Jain, R.B., "Regression Models to Predict Corrected Weight, Height and Obesity Prevalence from Self-reported Data: Data from BRFSS 1999–2007", International Journal of Obesity, 2010, 34:1655–1664.

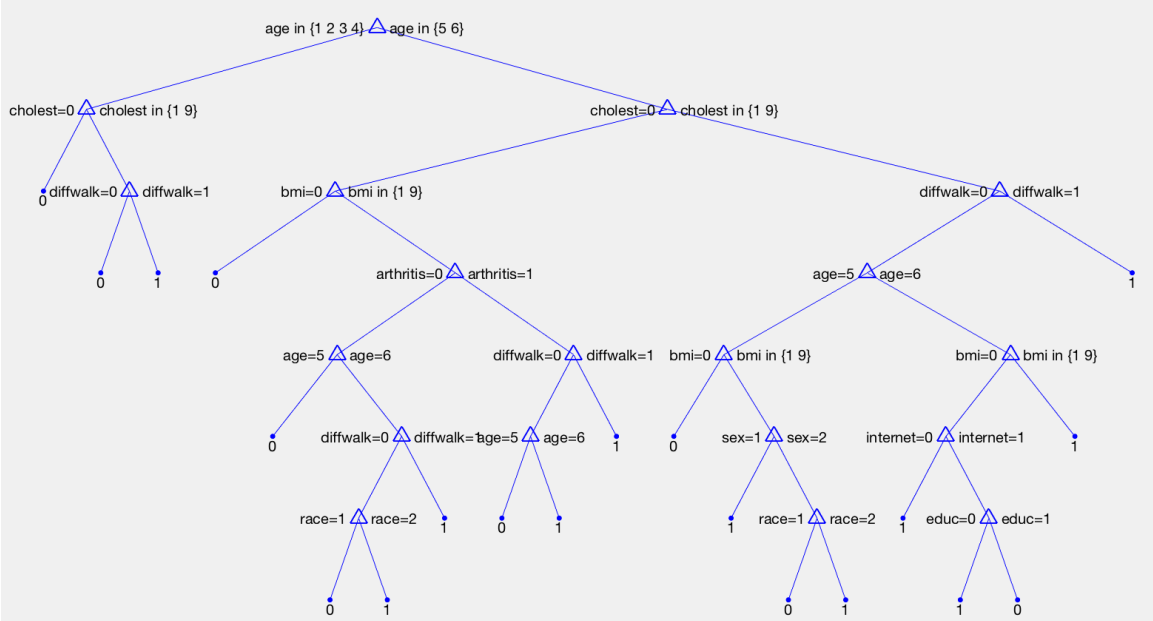

Figure 5: Hypertension classification tree after controlling for diabetes.

[11] Jerant, A. F. et al., "Age-related Disparities in Cancer Screening: Analysis of 2001 Behavioral Risk Factor Surveillance System Data", Annals of Family Medicine, 2004, 2(5), 481–487.

[12] , Lee, B.J. and Kim, J.Y., "A Comparison of the Predictive Power of Anthropometric Indices for Hypertension and Hypotension Risk", PLoS One, 1014, 9(1):e84897.

[13] Lewis, M., " Moneyball: The Art of Winning an Unfair Game", 2003.

[14] Luo, W. et al., "Is Demography Destiny? Application of Machine Learning Techniques to Accurately Predict Population Health Outcomes from a Minimal Demographic Dataset", PLoS One., 2015, 10(5): e0125602.

[15] Moore, L. et al., "Using Behavioral Risk Factor Surveillance System Data to Estimate the Percentage of the Population Meeting US Department of Agriculture Food Patterns Fruit and Vegetable Intake Recommendations", Am. J. Epidemiol., 2015.

[16] Mouhayar, E. and Salahudeen, A., "Hypertension in Cancer Patients", Tex Heart Inst J., 2011, 38(3): 263265.

[17] http://www.cdc.gov/nchs/nhanes/nhanes_questionnaires.htm, "National Health and Nutrition Examination Survey".

[18] Parikh, N.I. et al., "A Risk Score for Predicting Near-term Incidence of Hypertension: the Framingham Heart Study", Ann Intern Med. 2008,148(2):102–10.

[19] , Rohan, E.A. et al., "Health Behaviors and Quality of Life Among Colorectal Cancer Survivors", J Natl Compr Canc Netw., 2015, 13(3): 297302.

[20] , Seirawan, H., Parsimonious prediction model for the prevalence of dental visits. Community Dentistry and Oral Epidemiology, 2008, 36: 40–408.

[21] Zalpuri, S. et al., "Association vs. Causality in Transfusion Medicine: Understanding Multivariable Analysis in Prediction vs. Etiologic Research", Transfus Med Rev., 2013, 27(2):74–81.

[22] world-heart-federation.org

[23] Wang, A. et al., "Predicting Hypertension without Measurement: A Non-invasive, Questionnaire-based Approach", Expert Systems with Applications, 2015.

[24] Wojcik, K. et al., "Identifying Populations at High Risk for Diabetes With the Behavioral Risk Factor Surveillance System, Rhode Island, 2003", Prev Chronic Dis., 2010, 7(4): A86.

[25] , Yoon, S. et al., "Using a Data Mining Approach to Discover Behavior Correlates of Chronic Disease: A Case Study of Depression", Stud Health Technol Inform., 2014, 201: 7178.

Appendix A: Data Preprocessing Code

I used an R script to convert data in the original SAS format to CSV:

```
brfss <- sasxport("BRFSS_2015.XPT")
write.csv(brfs, file="brfss2015.csv")
```

I used Python to load the CSV data file and perform operations on the variables. Below is an extract of the Python notebook I used to select features and standardize their values.

```
#-----------------------------------------------------------------------
# python notebook to process BRFSS data in CSV format for use in
# the predictive modeling of hypertension risk factors
#
# Author: Ioana Boier
# Date: Sep 2016
#-----------------------------------------------------------------------

import pandas as pd

# Flags for data generation
predictHyp = 1 # 1 - predict hypertension, 0 - predict diabetes
controlDiab = 0
iteration = 1 # 1 - all variables; 2 - variables after logistic elimination
contribute = False # output data to file or not

# read raw data from CDC BRFSS file (previously converted from SAS to csv format)
year = '2015'
root = ~/Projects/CDC/data/brfss'
df = pd.read_csv(root + year + '.csv', encoding='cp1252')

# output some raw data info
print("Number of rows:", len(df))
print("Number of columns (variables in the survey):", len(df.columns))
print(df.columns)

# keep some columns of interest (potential predictors of hypertension)
if (year == '2015'):
    drinkColumn = 'x.rfdrhv5'
else:
    drinkColumn = 'x.rfdrhv4'
if (predictHyp == 1):
    inp = 'diabete3'
    response = 'x.rfhype5'
else:
    inp = 'x.rfhype5'
    response = 'diabete3'

if iteration == 1:
    raceColumn = 'x.race'
    columnsKeep = df[['x.age.g', 'sex', 'x.rfbmi5', 'marital', 'x.educag', raceColumn, 'x.incomg',
'x.totinda', 'x.rfchol', drinkColumn, 'x.asthms1', 'medcost', 'chccopd1', 'addepev2',
'renthom1', 'employ1', 'internet', 'smoke100', 'ftjuda1.', 'frutda1.', 'beanday.',
'vegeda1.', inp, 'hlthpln1', 'havarth3', 'chckidny', 'veteran3', 'blind', 'decide',
'diffwalk', 'usenow3',
```

```
                    response]]
else:
    raceColumn = 'x.raceg21'
    if (controlDiab == 0):
        columnsKeep = df[['x.age.g', 'sex', 'x.rfbmi5', 'marital', 'x.educag', raceColumn,
'x.totinda', 'x.rfchol', drinkColumn, 'medcost', 'chccopd1', 'addepev2',
                    'internet', 'smoke100', 'ftjuda1.', 'frutda1.',
                    inp, 'hlthpln1', 'havarth3', 'chckidny', 'veteran3', 'blind', 'diffwalk', 'usenow3',
                    response]]
    else:
        columnsKeep = df[['x.age.g', 'sex', 'x.rfbmi5', 'marital', 'x.educag', raceColumn,
'x.totinda', 'x.rfchol', drinkColumn, 'medcost', 'chccopd1', 'addepev2',
                    'internet', 'smoke100', 'ftjuda1.', 'frutda1.',
                    'hlthpln1', 'havarth3', 'chckidny', 'veteran3', 'blind', 'diffwalk', 'usenow3',
                    response]]

# function to re-map data values from src_values to val_new
# and to replace missing (NaN) values with na_val
def cleanColumn(df, val_src, val_new, na_val):
    df.replace(val_src, val_new, inplace=True)
    df.fillna(na_val, inplace=True)
    print(df.value_counts())
    return df

# function to control for diabetes in a given variable
def controlDiabetes(df, colDiab, na_val):
    df.loc[colDiab==1] = na_val
    print(df.value_counts())
    return df

# process each variable according to its values
naVal = -99
oVal = 9;

# process AGE (x.age.g) column
variable = 'x.age.g'
# nothing to do

# process SEX (sex) column
variable = 'sex'
# nothing to do

# process BMI (x.rfbmi5) column
variable = 'x.rfbmi5'
if (variable in columnsKeep):
    columnsKeep[variable] = cleanColumn(columnsKeep[variable], [1, 2, 9], [0, 1, oVal], naVal)

# process MARITAL (marital) column
variable = 'marital'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] = cleanColumn(columnsKeep[variable], [9], [oVal], naVal)
    else:
        columnsKeep[variable] =
```

```
cleanColumn(columnsKeep[variable], [1,2,3,4,5,6,9], [1,0,0,0,0,0,naVal], naVal)

# process EDUCATION (x.educag) column
variable = 'x.educag'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] = cleanColumn(columnsKeep[variable], [9], [oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1,2,3,4,9], [0,0,0,1,naVal], naVal)

# process RACE (x.race) column
variable = raceColumn
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] = cleanColumn(columnsKeep[variable], [9], [oVal], naVal)
    else:
        columnsKeep[variable] = cleanColumn(columnsKeep[variable], [9], [naVal], naVal)

# process INCOME (x.incomg) column
variable = 'x.incomg'
if (variable in columnsKeep):
    columnsKeep[variable] = cleanColumn(columnsKeep[variable], [9], [oVal], naVal)

# process EXERCISE (x.totinda) column
variable = 'x.totinda'
if (variable in columnsKeep):
    columnsKeep[variable] = cleanColumn(columnsKeep[variable], [2, 9], [0, oVal], naVal)

# process CHOLESTEROL (x.rfchol) column
variable = 'x.rfchol'
if (variable in columnsKeep):
    columnsKeep[variable] = cleanColumn(columnsKeep[variable], [1, 2, 9], [0, 1, oVal], naVal)

# process HEAVY DRINKER (x.rfdrhv5) column
variable = drinkColumn
if (variable in columnsKeep):
    columnsKeep[variable] = cleanColumn(columnsKeep[variable], [1, 2, 9], [0, 1, oVal], naVal)

# process ASTHMA (x.asthms1) column
variable = 'x.asthms1'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] = cleanColumn(columnsKeep[variable], [9], [oVal], naVal)
    else:
        columnsKeep[variable] = cleanColumn(columnsKeep[variable], [1,2,3,9], [1,1,0,naVal], naVal)

# process MEDICAL COST (medcost) column
variable = 'medcost'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
```

```
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process COPD/BRONCHITIS (chccopd1) column
variable = 'chccopd1'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process DEPRESSION (addepev2) column
variable = 'addepev2'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process OWN/RENT (renthom1) column
variable = 'renthom1'
if (variable in columnsKeep):
    columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 3, 7, 9], [1, 2, 3, oVal, oVal], naVal)

# process EMPLOYMENT (employ1) column
variable = 'employ1'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [9], [oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1,2,3,4,5,6,7,8,9], [1,2,3,3,5,oVal,7,8,oVal], naVal)

# process INTERNET (internet) column
variable = 'internet'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process SMOKER (smoke100) column
variable = 'smoke100'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
```

```
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process DIABETES (diabete3) column
variable = 'diabete3'
if (variable in columnsKeep):
    columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [2, 3, 4, 7, 9], [0, 0, 0, oVal, oVal], naVal)

# process HEALTH PLAN (hlthpln1) column
variable = 'hlthpln1'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process ARTHRITIS (havarth3) column
variable = 'havarth3'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process CHCKIDNY (chckidny) column
variable = 'chckidny'
if (variable in columnsKeep):
    columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)

# process VETERAN (veteran3) column
variable = 'veteran3'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process BLIND (blind) column
variable = 'blind'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
```

```
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process DECIDE (decide) column
variable = 'decide'
if (variable in columnsKeep):
    columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)

# process DIFFWALK (diffwalk) column
variable = 'diffwalk'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 7, 9], [1, 0, naVal, naVal], naVal)

# process TOBACCO (usenow3) column
variable = 'usenow3'
if (variable in columnsKeep):
    if iteration == 1:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 3, 7, 9], [1, 2, 3, oVal, oVal], naVal)
    else:
        columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 3, 7, 9], [1, 2, 3, naVal, naVal], naVal)

# process HYPERTENSION (x.rfhype5) column
variable = 'x.rfhype5'
if (variable in columnsKeep):
    columnsKeep[variable] =
cleanColumn(columnsKeep[variable], [1, 2, 9], [0, 1, naVal], naVal)
if controlDiab:
    columnsKeep[variable] =
controlDiabetes(columnsKeep[variable], df['diabete3'], naVal)

# process FRUIT JUICE (ftjuda1.) column
variable = 'ftjuda1.'
if (variable in columnsKeep):
    columnsKeep[variable].fillna(naVal, inplace=True)
    #print(columnsKeep[variable].value_counts())

# process FRUIT (frutda1.) column
variable = 'frutda1.'
if (variable in columnsKeep):
    columnsKeep[variable].fillna(naVal, inplace=True)
    #print(columnsKeep[variable].value_counts())

# process BEANS (beanday.) column
variable = 'beanday.'
if (variable in columnsKeep):
    columnsKeep[variable].fillna(naVal, inplace=True)
```

```
    #print(columnsKeep[variable].value_counts())

# process GREENS (grenday.) column
variable = 'grenday.'
if (variable in columnsKeep):
    columnsKeep[variable].fillna(naVal, inplace=True)
    #print(columnsKeep[variable].value_counts())

# process ORANGES (orngday.) column
variable = 'orngday.'
if (variable in columnsKeep):
    columnsKeep[variable].fillna(naVal, inplace=True)
    #print(columnsKeep[variable].value_counts())

# process VEGETABLES (vegeda1.) column
variable = 'vegeda1.'
if (variable in columnsKeep):
    columnsKeep[variable].fillna(naVal, inplace=True)
    #print(columnsKeep[variable].value_counts())

# remove rows with missing values
cleanData = columnsKeep
for col in columnsKeep.columns:
    cleanData = cleanData[cleanData[col] != naVal]

# write clean data to .csv file
if contribute:
    if (predictHyp == 1):
        if (controlDiab == 1):
            suffix = '_hycd'
        else:
            suffix = '_hy'+str(iteration)
    else:
        suffix = '_db'+str(iteration)
    cleanData.to_csv(root + year + suffix + '_extra.csv', sep=',', header=False)
```

Appendix B: Matlab Classification Code

```
% --------------------------------------------------------------------------------
% trainLogisticClassifier(trainingData, headers, isCategoricalPredictor, KFolds)
%   returns a logistic classifier and its accuracy
%
%   Input:
%       trainingData: the training data in matrix format with the
%           variables in columns and the response variable in the last
%           column
%       headers: the labels of the variables in the data in the same order
%           as they appear in the data matrix
%       isCategoricalPredictor: an array of boolean flags one for each
%           predictor variable to indicate if the variable is categorical
%           or not
%       KFolds: the number of folds to be used for cross-validation when
%           computing the validation accuracy of the model
%
%   Output:
%       trainedClassifier: a struct containing the trained classifier
%           containing various fields with information about the classifier.
%
%       trainedClassifier.predictFcn: a function to make predictions
%           on new data. It takes an input in the same matrix format as
%           the training code and returns predictions for the response.
%
%       validationAccuracy: (1 - mean squared error) between the observations
%           in a fold when compared against predictions made with a tree
%           trained on the out-of-fold data.
%
%   Author: Ioana Boier, Sep 2016
% --------------------------------------------------------------------------------

function [trainedClassifier, validationAccuracy] = ...
    trainLogisticClassifier(trainingData, headers, isCategoricalPredictor, KFolds)

% Extract predictors and response, check input dimensions
inputTable = array2table(trainingData, 'VariableNames', headers);
n = size(trainingData, 2);
if (n ~= size(headers, 2))
    error('Error: training data and the headers must have the same number of columns.')
end;
predictorNames = headers(1:n-1);
predictors = inputTable(:, predictorNames);
response = trainingData(:, n);
if (size(isCategoricalPredictor, 2) ~= n-1)
    error('Error: number of predictors must be equal to number of categorical flags')
end;

% Create and train the classifier
% For logistic regression response values must be converted to zeros
% and ones as the responses are assumed to follow a binomial distribution
% 1 or true = 'successful' class
% 0 or false = 'failure' class
```

```
% NaN - missing response.
successClass = double(1);
failureClass = double(0);
missingClass = double(NaN);
successFailureAndMissingClasses = [successClass; failureClass; missingClass];
isMissing = isnan(response);
zeroOneResponse = double(ismember(response, successClass));
zeroOneResponse(isMissing) = NaN;
categoricalPredictorIndex = find(isCategoricalPredictor);
concatenatedPredictorsAndResponse = [predictors, table(zeroOneResponse)];

% Train using zero-one responses, specifying which predictors are categorical
GeneralizedLinearModel = fitglm(...
    concatenatedPredictorsAndResponse, ...
    'Distribution', 'binomial', ...
    'link', 'logit', ...
    'CategoricalVars', categoricalPredictorIndex);

% Convert predicted probabilities to predicted class labels and scores
convertSuccessProbsToPredictions = ...
@(p) successFailureAndMissingClasses( ~isnan(p).*( (p<0.5) + 1 ) + isnan(p)*3 );
returnMultipleValuesFcn = @(varargin) varargin{1:max(1,nargout)};
scoresFcn = @(p) [1-p, p];
predictionsAndScoresFcn = ...
@(p) returnMultipleValuesFcn( convertSuccessProbsToPredictions(p), scoresFcn(p) );

% Create the result object containing the prediction function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
logisticRegressionPredictFcn = @(x) predictionsAndScoresFcn( predict(GeneralizedLinearModel, x) );
trainedClassifier.predictFcn = @(x) logisticRegressionPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result object
trainedClassifier.GeneralizedLinearModel = GeneralizedLinearModel;
trainedClassifier.SuccessClass = successClass;
trainedClassifier.FailureClass = failureClass;
trainedClassifier.MissingClass = missingClass;
trainedClassifier.ClassNames = {successClass; failureClass};
trainedClassifier.About = 'Logistic classifier @ Ioana Boier, 2016';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new matrix, X, use: \n
yfit = trainedClassifier.predictFcn(X) \nX must contain only predictor columns in the same
order as the training data.');

% Perform cross-validation
cvp = cvpartition(response, 'KFold', KFolds);
validationPredictions = response;
numObservations = size(predictors, 1);
numClasses = 2;
validationScores = NaN(numObservations, numClasses);
for fold = 1:KFolds
    trainingPredictors = predictors(cvp.training(fold), :);
    trainingResponse = response(cvp.training(fold), :);
    foldIsCategoricalPredictor = isCategoricalPredictor;

    isMissing = isnan(trainingResponse);
```

```
    zeroOneResponse = double(ismember(trainingResponse, successClass));
    zeroOneResponse(isMissing) = NaN;
    categoricalPredictorIndex = find(foldIsCategoricalPredictor);
    concatenatedPredictorsAndResponse = [trainingPredictors, table(zeroOneResponse)];

    % Train on current fold
    GeneralizedLinearModel = fitglm(...
        concatenatedPredictorsAndResponse, ...
        'Distribution', 'binomial', ...
        'link', 'logit', ...
        'CategoricalVars', categoricalPredictorIndex);

    % Convert predicted probabilities to predicted class labels and scores
    convertSuccessProbsToPredictions = ...
@(p) successFailureAndMissingClasses( ~isnan(p).*( (p<0.5) + 1 ) + isnan(p)*3 );
    returnMultipleValuesFcn = @(varargin) varargin{1:max(1,nargout)};
    scoresFcn = @(p) [1-p, p];
    predictionsAndScoresFcn = ...
@(p) returnMultipleValuesFcn( convertSuccessProbsToPredictions(p), scoresFcn(p) );

    % Create the result struct with predict function
    logisticRegressionPredictFcn = ...
@(x) predictionsAndScoresFcn( predict(GeneralizedLinearModel, x) );
    validationPredictFcn = @(x) logisticRegressionPredictFcn(x);

    % Compute validation predictions and scores
    validationPredictors = predictors(cvp.test(fold), :);
    [foldPredictions, foldScores] = validationPredictFcn(validationPredictors);

    % Store predictions and scores in the original order
    validationPredictions(cvp.test(fold), :) = foldPredictions;
    validationScores(cvp.test(fold), :) = foldScores;
end

correctPredictions = (validationPredictions == response);
validationAccuracy = sum(correctPredictions)/length(correctPredictions);


% ------------------------------------------------------------------------------
% logisticModel.m: driver code to load a curated subset of the BRFSS 2015
%   data and perform logistic regression
%
%   Author: Ioana Boier, Sep 2016
% ------------------------------------------------------------------------------
clear;

% iteration can be 1 or 2 and indicates which pass of the regression is
% being performed:
%       pass 1: initial data with 31 features manually selected from BRFSS
%           dataset
%       pass 2: reduced data with 24 features after p-value base feature
%           selection
%       pass 3: reduced data with 7 features after tree selection
iteration = 2;
```

```
% setup data matrix, headers, and categorical flags
if (iteration == 1)
    data = csvread('data/brfss2015_hy1_extra.csv');
    headers = {'age'; 'sex'; 'bmi'; 'marital'; 'educ'; 'race'; 'income'; ...
        'exercise'; 'cholest'; 'alcohol'; 'asthma'; 'medcost'; 'copd'; ...
        'depression'; 'rentown'; 'employ'; 'internet'; 'smoke'; 'juice'; ...
        'fruit'; 'bean'; 'veggie'; 'diabetes'; 'healthplan'; 'arthritis'; ...
        'kidney'; 'veteran'; 'blind'; 'cognition'; 'diffwalk'; 'tobacco'; ...
        'hypert'};
    isCategoricalPredictor = [
        true; true; true; true; true; true; ...
        true; true; true; true; true; true; ...
        true; true; true; true; true; true; ...
        false; false; false; false; true; true; ...
        true; true; true; true; true; true; true];
elseif (iteration == 2)
    data = csvread('data/brfss2015_hy2_extra.csv');
    headers = {'age'; 'sex'; 'bmi'; 'marital'; 'educ'; 'race'; 'exercise'; ...
        'cholest'; 'alcohol'; 'medcost'; 'copd'; 'depression'; 'internet'; ...
        'smoke'; 'juice'; 'fruit'; 'diabetes'; 'healthplan'; 'arthritis'; ...
        'kidney'; 'veteran'; 'blind'; 'diffwalk';'tobacco'; ...
        'hypert'};
    isCategoricalPredictor = [
        true; true; true; true; true; true; ...
        true; true; true; true; true; true; ...
        true; true; false; false; true; true; ...
        true; true; true; true; true; true];
else
    data = csvread('data/brfss2015_hy3_extra.csv');
    headers = {'age'; 'sex'; 'bmi'; 'cholest'; ...
        'arthritis'; 'diffwalk'; 'diabetes'; ...
        'hypert'};
    isCategoricalPredictor = [
        true; true; true; true; true; true; true;];
end;
[m, n] = size(data);

% invoke the classifier with 10-fold cross validation
[trainedClassifier, accuracyTrain] = ...
    trainLogisticClassifier(data(:,2:n), headers', isCategoricalPredictor', 10);

% display results for the training set
fprintf('\naccuracyTrain = %.4f', accuracyTrain);
trainedClassifier.GeneralizedLinearModel.Coefficients

% Perform out-of-sample validation on the 2013 BRFSS dataset
if (iteration == 1)
    testData = csvread('data/brfss2013_hy1_extra.csv');
elseif (iteration == 2)
    testData = csvread('data/brfss2013_hy2_extra.csv');
else
    testData = csvread('data/brfss2013_hy3_extra.csv');
end;
```

```
yfit = trainedClassifier.predictFcn(testData(:,2:n-1));

% calculate and display classification measures for the test set
posLabels = (testData(:,n) == 1);
negLabels = (testData(:,n) == 0);
TP = sum(yfit.*posLabels==1);
TN = sum(~yfit.*negLabels==1);
FP = sum(yfit.*negLabels==1);
FN = sum(~yfit.*posLabels==1);
Accuracy = (TP+TN)/(TP+TN+FP+FN);
Precision = TP/(TP+FP);
Recall = TP/(TP+FN);
Specificity = TN/(FP+TN);
AUC = 0.5*(TP/(TP+FN)+TN/(TN+FP));

fprintf('\nConfusion matrix:\nTN = %d, FP = %d\nFN = %d, TP = %d\n', TN, FP, FN, TP);
fprintf('\nTotal: %d', TN+FN+TP+FP);
fprintf('\nAccuracy: %.4f', Accuracy);
fprintf('\nPrecision: %.4f', Precision);
fprintf('\nRecall: %.4f', Recall);
fprintf('\nSpecificity: %.4f', Specificity);
fprintf('\nAUC: %.4f\n', AUC);


% ------------------------------------------------------------------------------
% trainTreeClassifier(trainingData, headers, isCategoricalPredictor, KFolds)
%   returns a tree classifier and accuracy-related information
%
%   Input:
%       trainingData: the training data in matrix format with the
%           variables in columns and the response variable in the last
%           column
%       headers: the labels of the variables in the data in the same order
%           as they appear in the data matrix
%       isCategoricalPredictor: an array of boolean flags one for each
%           predictor variable to indicate if the variable is categorical
%           or not
%       KFolds: the number of folds to be used for cross-validation when
%           computing the validation accuracy of the model
%
%   Output:
%       trainedClassifier: a struct containing the trained classifier
%           containing various fields with information about the classifier.
%
%       trainedClassifier.predictFcn: a function to make predictions
%           on new data. It takes an input in the same matrix format as
%           the training code and returns predictions for the response.
%
%       validationAccuracy: (1 - mean squared error) between the observations
%           in a fold when compared against predictions made with a tree
%           trained on the out-of-fold data.
%
%       validationPredictions: predicted class labels for every fold
%
```

```
%       validationScores: posterior probabilities of the classification for
%           every fold
%
%   Author: Ioana Boier, Sep 2016
% --------------------------------------------------------------------------
function [trainedClassifier, validationAccuracy, validationPredictions, validationScores] = ...
    trainTreeClassifier(trainingData, headers, isCategoricalPredictor, KFolds)

% Extract predictors and response, check input dimensions
inputTable = array2table(trainingData, 'VariableNames', headers);
n = size(trainingData, 2);
if (n ~= size(headers, 2))
    error('Error: training data and the headers must have the same number of columns.')
end;
predictorNames = headers(1:n-1);
predictors = inputTable(:, predictorNames);
response = trainingData(:, n);
if (size(isCategoricalPredictor, 2) ~= n-1)
    error('Error: number of predictors must be equal to number of categorical flags')
end;

% Create and train the classifier
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'CategoricalPredictors',isCategoricalPredictor,...
    'MaxNumSplits',100);

% Create the result object containing the prediction function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result object
trainedClassifier.ClassificationTree = classificationTree;
trainedClassifier.About = 'Tree classifier @ Ioana Boier, 2016';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new matrix, X, use: \n
yfit = trainedClassifier.predictFcn(X) \nX must contain only predictor columns in the
same order as the training data.');

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationTree, 'KFold', KFolds);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

% Compute validation predictions and scores
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% --------------------------------------------------------------------------
% treeModel.m: driver code to load a curated subset of the BRFSS 2015
%   data and perform decision tree classification
%
%   Author: Ioana Boier, Sep 2016
```

```
% --------------------------------------------------------------------------
clear;

% load the reduced data after the second iteration of logistic regression
data = csvread('data/brfss2015_hy2_extra.csv');
[m, n] = size(data);
headers = {'age'; 'sex'; 'bmi'; 'marital'; 'educ'; 'race'; 'exercise'; ...
        'cholest'; 'alcohol'; 'medcost'; 'copd'; 'depression'; 'internet'; ...
        'smoke'; 'juice'; 'fruit'; 'diabetes'; 'healthplan'; 'arthritis'; ...
        'kidney'; 'veteran'; 'blind'; 'diffwalk';'tobacco'; ...
        'hypert'};

isCategoricalPredictor = [
    true; true; true; true; true; ...
    true; true; true; true; true; ...
    true; true; true; true; false; false;  ...
    true; true; true; true; true; true; true; true];

% invoke the classifier with 10-fold cross validation
kfold = 10;
[trainedClassifier, accuracyTrain, ~, ~] = ...
    trainTreeClassifier(data(:,2:n), headers', isCategoricalPredictor', kfold);

% display results for the training set and the tree
fprintf('\naccuracyTrain = %.4f', accuracyTrain);
t1 = trainedClassifier.ClassificationTree;
view(t1, 'Mode','graph')

% prune the tree to optimal level
[~,~,~,bestlevel] = cvLoss(t1,'SubTrees','All');
t2 = prune(t1,'level',bestlevel);

% display results for the pruned tree
classError2 = kfoldLoss(crossval(t2, 'KFold', kfold), 'LossFun', 'ClassifError');
accuracyPrune = 1 - classError2;
fprintf('\naccuracyPrune = %.4f, bestlevel = %d', accuracyPrune, bestlevel);
view(t2, 'Mode','graph')

% Perform out-of-sample validation on the 2013 BRFSS dataset
testData = csvread('data/brfss2013_hy2_extra.csv');
yfit = trainedClassifier.predictFcn(testData(:,2:n-1));

% calculate and display classification measures for the test set
posLabels = (testData(:,n) == 1);
negLabels = (testData(:,n) == 0);
TP = sum(yfit.*posLabels==1);
TN = sum(~yfit.*negLabels==1);
FP = sum(yfit.*negLabels==1);
FN = sum(~yfit.*posLabels==1);
Accuracy = (TP+TN)/(TP+TN+FP+FN);
Precision = TP/(TP+FP);
Recall = TP/(TP+FN);
Specificity = TN/(FP+TN);
AUC = 0.5*(TP/(TP+FN)+TN/(TN+FP));
```

```
fprintf('\nConfusion matrix:\nTN = %d, FP = %d\nFN = %d, TP = %d\n', TN, FP, FN, TP);
fprintf('\nTotal: %d', TN+FN+TP+FP);
fprintf('\nAccuracy: %.4f', Accuracy);
fprintf('\nPrecision: %.4f', Precision);
fprintf('\nRecall: %.4f', Recall);
fprintf('\nSpecificity: %.4f', Specificity);
fprintf('\nAUC: %.4f\n', AUC);
```

Appendix C: Logistic Coefficients

After the first iteration of the logistic regression, the coefficients and their corresponding statistics:

	Estimate	SE	tStat	pValue
(Intercept)	-2.3481	0.065242	-35.99	1.1797e-283
age_2	0.45251	0.047173	9.5925	8.5974e-22
age_3	0.86719	0.046029	18.84	3.5482e-79
age_4	1.2919	0.045518	28.382	3.3928e-177
age_5	1.7119	0.045565	37.57	6.6133e-309
age_6	2.1479	0.046534	46.157	0
sex_2	-0.29563	0.0099745	-29.638	4.7992e-193
bmi_1	0.67489	0.0094574	71.361	0
bmi_9	0.60499	0.018925	31.968	3.0222e-224
marital_2	0.050066	0.012699	3.9424	8.0683e-05
marital_3	0.2369	0.013443	17.622	1.6591e-69
marital_4	0.072743	0.030601	2.3771	0.017449
marital_5	0.085139	0.015359	5.5432	2.9702e-08
marital_6	2.1694e-05	0.028995	0.00074822	0.9994
marital_9	-0.11977	0.060665	-1.9742	0.048354
educ_2	-0.016808	0.018806	-0.89374	0.37146
educ_3	-0.077797	0.019366	-4.0172	5.8893e-05
educ_4	-0.22156	0.019822	-11.177	5.2772e-29
educ_9	-0.13569	0.084719	-1.6016	0.10925
race_2	0.67382	0.016655	40.457	0
race_3	0.10205	0.036204	2.8187	0.0048219
race_4	0.15812	0.031501	5.0195	5.1796e-07
race_5	0.25412	0.079326	3.2035	0.0013575
race_6	-0.035004	0.06423	-0.54497	0.58577
race_7	0.15387	0.031603	4.869	1.1219e-06
race_8	-0.015395	0.017928	-0.85871	0.3905
race_9	-0.020136	0.035001	-0.57529	0.5651
income_2	-0.023421	0.018999	-1.2327	0.21767
income_3	-0.040778	0.020967	-1.9449	0.051785
income_4	-0.034793	0.020346	-1.71	0.087258
income_5	-0.1294	0.019455	-6.6512	2.9076e-11
income_9	-0.10931	0.019559	-5.5888	2.2867e-08
exercise_1	-0.13219	0.0096987	-13.63	2.6722e-42
exercise_9	-0.19201	0.05071	-3.7865	0.00015279
cholest_1	0.78986	0.0082853	95.334	0
cholest_9	0.46057	0.04475	10.292	7.6456e-25
alcohol_1	0.28489	0.018263	15.6	7.3065e-55
alcohol_9	0.10557	0.033802	3.1232	0.0017888
asthma_2	-0.0020918	0.024831	-0.08424	0.93287
asthma_3	-0.12338	0.014734	-8.3735	5.591e-17
asthma_9	-0.0084614	0.051869	-0.16313	0.87042
medcost_1	0.11153	0.015606	7.1465	8.8987e-13
medcost_9	-0.072574	0.09736	-0.74542	0.45602
copd_1	0.079539	0.01573	5.0565	4.2706e-07
copd_9	0.04288	0.061652	0.69551	0.48673
depression_1	0.07462	0.011304	6.6014	4.0722e-11
depression_9	-0.030499	0.067524	-0.45168	0.6515

rentown_2	0.0039891	0.012159	0.32806	0.74286
rentown_3	0.066852	0.025181	2.6549	0.0079343
rentown_9	-0.10814	0.061606	-1.7553	0.079201
employ_2	-0.14374	0.015764	-9.1187	7.6038e-20
employ_3	0.10138	0.02995	3.3851	0.00071155
employ_4	0.043232	0.032394	1.3346	0.18202
employ_5	0.04617	0.019296	2.3928	0.016721
employ_6	-0.057137	0.048208	-1.1852	0.23592
employ_7	0.1514	0.0127	11.922	9.1431e-33
employ_8	0.14017	0.019561	7.1659	7.7267e-13
employ_9	0.0088224	0.05805	0.15198	0.8792
internet_1	-0.13108	0.011862	-11.05	2.1943e-28
internet_9	-0.12618	0.10245	-1.2316	0.21809
smoke_1	0.018092	0.0086217	2.0984	0.035867
smoke_9	0.0041646	0.060341	0.069017	0.94498
juice	0.00023664	6.6849e-05	3.5399	0.00040033
fruit	-0.00044283	4.5984e-05	-9.6302	5.9624e-22
bean	-0.00018224	0.00010381	-1.7555	0.079173
veggie	0.0001223	5.6264e-05	2.1736	0.029733
diabetes_1	0.85461	0.012514	68.292	0
diabetes_9	0.37652	0.1167	3.2265	0.0012531
healthplan_1	0.10953	0.020141	5.4381	5.3843e-08
healthplan_9	-0.24119	0.096444	-2.5008	0.012391
arthritis_1	0.28375	0.0090663	31.297	5.1455e-215
arthritis_9	0.097987	0.054254	1.8061	0.070905
kidney_1	0.64544	0.023001	28.062	2.882e-173
kidney_9	0.55835	0.081454	6.8547	7.1467e-12
veteran_1	-0.10439	0.012934	-8.0708	6.9851e-16
veteran_9	-0.18235	0.13786	-1.3227	0.18593
blind_1	0.096247	0.019872	4.8434	1.2762e-06
blind_9	0.093665	0.085672	1.0933	0.27426
cognition_1	-0.0036906	0.015937	-0.23157	0.81687
cognition_9	-0.11467	0.060612	-1.8919	0.058511
diffwalk_1	0.30918	0.012443	24.847	2.7648e-136
diffwalk_9	0.16097	0.066634	2.4157	0.015707
tobacco_2	-0.16407	0.049022	-3.3469	0.00081708
tobacco_3	-0.26639	0.032764	-8.1305	4.2746e-16
tobacco_9	-0.13615	0.099293	-1.3712	0.1703

After the final iteration of feature selection based on p-values, the coefficients of the remaining features and their corresponding statistics:

	Estimate	SE	tStat	pValue
(Intercept)	-2.4974	0.056324	-44.34	0
age_2	0.4653	0.045279	10.276	9.022e-25
age_3	0.8886	0.043092	20.621	1.7714e-94
age_4	1.3152	0.04214	31.21	7.8529e-214
age_5	1.7696	0.041918	42.217	0
age_6	2.293	0.04204	54.544	0
sex_2	-0.24974	0.0096932	-25.765	2.1985e-146
bmi_1	0.68332	0.0094479	72.325	0
bmi_9	0.6134	0.019088	32.136	1.399e-226
marital_1	-0.16909	0.0086305	-19.592	1.7939e-85

educ_1	-0.19895	0.0089493	-22.231	1.7104e-109
race_2	0.29765	0.010902	27.303	3.8927e-164
exercise_1	-0.12781	0.0097539	-13.103	3.1649e-39
exercise_9	-0.15367	0.034999	-4.3907	1.13e-05
cholest_1	0.78983	0.0083214	94.916	0
cholest_9	0.43233	0.046257	9.3464	9.0721e-21
alcohol_1	0.26055	0.018329	14.215	7.4282e-46
alcohol_9	0.079003	0.03448	2.2913	0.021947
medcost_1	0.11718	0.01565	7.4874	7.0235e-14
copd_1	0.12959	0.015313	8.4624	2.6193e-17
depression_1	0.082318	0.010843	7.5917	3.1585e-14
internet_1	-0.17914	0.011223	-15.962	2.3359e-57
smoke_1	0.0242	0.0085968	2.815	0.0048773
juice	0.0003177	6.753e-05	4.7046	2.5435e-06
fruit	-0.00047948	4.484e-05	-10.693	1.0953e-26
diabetes_1	0.8641	0.012586	68.658	0
diabetes_9	0.45459	0.13189	3.4468	0.00056721
healthplan_1	0.12494	0.019964	6.2584	3.8899e-10
arthritis_1	0.30489	0.0090607	33.649	3.1912e-248
kidney_1	0.64996	0.023209	28.005	1.4097e-172
kidney_9	0.62254	0.087391	7.1237	1.0508e-12
veteran_1	-0.085156	0.012937	-6.5822	4.636e-11
blind_1	0.12262	0.01987	6.1713	6.772e-10
diffwalk_1	0.35333	0.012007	29.428	2.4238e-190
tobacco_2	-0.1541	0.04935	-3.1225	0.0017932
tobacco_3	-0.25514	0.033008	-7.7296	1.0787e-14

The coefficients of the logistic regression after tree-based feature selection:

	Estimate	SE	tStat	pValue
(Intercept)	-2.888	0.038805	-74.423	0
age_2	0.3794	0.042517	8.9235	4.5191e-19
age_3	0.75353	0.0403	18.698	5.1258e-78
age_4	1.1687	0.039337	29.71	5.7617e-194
age_5	1.5915	0.039017	40.789	0
age_6	2.1388	0.038858	55.042	0
sex_2	-0.19488	0.0079294	-24.577	2.2313e-133
bmi_1	0.67703	0.0087914	77.01	0
bmi_9	0.55845	0.016736	33.367	4.0656e-244
cholest_1	0.81295	0.0077752	104.56	0
cholest_9	0.54221	0.040133	13.511	1.3557e-41
arthritis_1	0.3509	0.0084059	41.744	0
diffwalk_1	0.58006	0.010437	55.575	0
diabetes_1	0.95654	0.011691	81.818	0
diabetes_9	0.57657	0.11085	5.2013	1.9792e-07