Maria DaRocha, 300399718
Assignment 1, Due: 29/07/2019
COMP309, Machine Learning Tools and Techniques

[Core]

Dataset: Cleveland Heart Disease
Expectation: 4 Experiments (one per tribe)

UCI File Selection Process:

During the dataset review and file selection period, I decided on using the processed.cleveland.data file and then converted it to a .csv file. I knew to select this data set based on the WARNING.txt uploaded to the project folder (available on the UCI Machine Learning Repository website). The warning file contained information about the cleveland.data file being corrupted. Additionally, upon reviewing the state of processed.cleveland.data.csv it was possible to see that missing values were handled appropriately (substituted with '?' character) and that the data was in a suitable format to be read into Weka after first adding a header for the 14 unique attributes.

1.) Connectionist

Approach: Multilayer Perceptron

Artificial Neural Networks, ANN's, focus on classifying instances given labelled examples. Uses adjusted weights to learn a decision boundary. One perceptron can only handle linear decision boundaries, so we use a multilayer perceptron to learn more complex functions. In most cases there is an input layer accompanied by one or more hidden layers completely connected to the next [output] layer, (which may be the input layer for another, and so on). Hence the feedforward approach is a *connectionist* implementation of machine learning.

Representation: Neural Networks

- The ANN serves as a basic representation of the multilayer perceptron by mathematically relating these layers. Through back propagation, we minimize the training error $J = ½ ||d-y||^2$ between desired output x and actual output y. Each connection has a weight (which can also be zero). The weights are adjusted based on the training examples provided to the learner. This mathematical relationship connects one layer to another, creating a multilayer ANN in the process.

Evaluation: Mean Squared Error (MSE)

- The mean squared error function is the most common method of producing an error function which represents the difference between the actual and desired outputs over the entire set of inputs.

- $E(w) = \frac{1}{2N} \sum_{p=1}^{N} (t^p - o^p)^2$ ...

  Where N is the number of patterns,
  t^p is the output target for pattern p, and
  o^p is the output obtained for pattern p

Optimization: Gradient Descent

- ***In summary***, feed forward the present pattern at the input layer (propagate the forwards activations) for all nodes, then calculate the error for the output neurons and propagate backwards, calculate the partial derivatives, repeat for all patterns, and then sum.

  (Expanded:) A technique used for minimizing the error function. Uses derivatives to adjust input values. Because we need to adjust several weights in our multilayer perceptron, we apply partial derivatives, (derive with respect to one variable while holding the others constant).
  - *dy/dx > 0* implies that y increases as x increases. (Thus to find the minimum y, we reduce x)
  - *dy/dx < 0* implies that y decreases as x increases. (Thus to find the minimum y, we increase x)
  - *dy/dx = 0* implies that we are at a local minimum or maximum

  We then apply a standard logistic function (sigmoid activation function) to gain the output of neuron *i* for pattern *p.* The sigmoid function gets its name from its "S"-shaped curve.

  *Aside: ***Sigmoid Function***; "It is used in neural networks to give logistic neurons real-valued output that is a smooth and bounded function of their total input. It also has the added benefit of having nice derivatives which make learning the weights of a neural network easier." -*Quora*

  We then update output neurons using the generalized delta rule:

$$\Delta w_{ij} = \eta \, \delta_i^p \, x_{ij}$$

$$\delta_i^p = (t_i^p - o_i^p) f'(u_i)$$

  ...and account for hidden nodes with the function:

$$\delta_i^p = \sum_k w_{ki} \, \delta_k f'(u_i)$$

Multilayer Perceptron: Unsplit results

=== Summary ===

Correctly Classified Instances	63	96.9231 %
Incorrectly Classified Instances	2	3.0769 %
Kappa statistic	0.9615	
Mean absolute error	0.0263	

Root mean squared error	0.0967
Relative absolute error	8.2084 %
Root relative squared error	24.1827 %
Total Number of Instances	65

=== Confusion Matrix ===

```
  a  b  c  d  e   <-- classified as
 13  0  0  0  0 |  a = 0
  0 13  0  0  0 |  b = 1
  0  0 12  0  1 |  c = 2
  0  0  0 13  0 |  d = 3
  0  0  0  1 12 |  e = 4
```

Model Assessment:

Overall, this is a decent model as it classifies with ~97% accuracy, but has some problematic shortcomings. The four categories that we are attempting to classify distinguish the presence (values 1,2,3,4) from absence (value 0) of a cardiac blockage. *(Value 0: < 50% diameter narrowing, Value 1: > 50% diameter narrowing).*

A complete blockage (e) is the most life threatening and is tied for the most misclassified case. Because this dataset is small and there is no splitting, I believe that it only performs this well as a result of the small sample population (after class balancing). Before class balancing, it performs with a ~93.3% accuracy. Therefore, this model is equally likely to run the risk of a false negative on a complete cardiac blockage, as it is to run a false positive in other cases. This is obviously problematic for health and safety reasons should the model be used to predict blockage level on a larger sample population. This level of inaccuracy is likely to be a result of only a small number of *e* instances out of our 303 samples, further reduced to 65 instances after balancing.

2.) Symbolic,

Approach: Random Forest

Symbolists believe that new knowledge is discovered by filling in the gaps of our existing knowledge. The random forest belongs to this tribe because it is a dispersed and logical sequence of decision trees. The decision trees themselves are logical rules which use recursion to split the domain and build a classifier. Symbolists emphasize the importance of deduction and induction of the model's basic rules, then chain them to form complex reasoning, and is precisely how a random forest operates. The output of a random forest is categorical and the learning type is supervised.

Representation: Logic

- As explained above, the representation of a random forest lies in the logical rules of the decision trees. During the building process, the model learns the questions that it needs to ask and how much weight they carry on the classification.

For new knowledge, each tree in the random forest begs a question of an instance, then further refines the classification based on the result.

Evaluation: Accuracy

- The accuracy of a random forest determines its efficacy as a classifier. In cases where models demonstrate 90-100% accuracy, the data used to train the model is often robust in both quality and quantity. Often the size of the data used to train the model leads to it being more time consuming than other approaches, but the trade-off is improved accuracy.

Optimization: Inverse Deduction

- Inverse deduction is induction, which is when you derive a general conclusion from a specific conclusion. Induction is when you prove an individual instance to be true, then use operators to extrapolate truth for all other instances as well.

Random Forest: Unsplit results

=== Summary ===

Correctly Classified Instances	303	100	%
Incorrectly Classified Instance	0	0	%
Kappa statistic	1		
Mean absolute error	0.0742		
Root mean squared error	0.1198		
Relative absolute error	28.6575 %		
Root relative squared error	33.3659 %		
Total Number of Instances	303		

=== Confusion Matrix ===

a	b	c	d	e	<-- classified as
164	0	0	0	0	a = 0
0	55	0	0	0	b = 1
0	0	36	0	0	c = 2
0	0	0	35	0	d = 3
0	0	0	0	13	e = 4

Model Assessment:

This model performs with 100% accuracy because it tests on the exact same instances used to train it. Unsplit, this model only says that it classifies the points that trained it with complete accuracy. This is unsurprising given the nature of a random forest. This model does not indicate how well it might do at classifying new data points.

3.) Bayesian

Approach: Naive Bayes

The Naive Bayes algorithm is based on Bayes Theorem which describes the probability of an event given prior knowledge.

Bayes Theorem: $P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$ where…

$P(A|B)$ is our posterior: How probable is our hypothesis given the observed evidence?

$P(B|A)$ is our likelihood: How probable is the evidence given our hypothesis is true?

$P(A)$ is our prior: How probable was our hypothesis before observing any evidence?

$P(A|B)$ is our marginal: How probable is the new evidence under all possible hypotheses? $P(B) = \Sigma\, P(B|A)\,P(A)$

Have: Numerous hypotheses about the data

Start: Start with ***prior***

1. As you begin to see evidence, update your belief in each hypothesis using the ***likelihood***.
2. If the hypothesis makes the evidence being seen likely, then conversely, the evidence also makes the hypothesis likely (***marginal***).
   **Aside:* The marginal is used to make the probability sum to 1.
3. Multiply the likelihood and the prior to obtain the ***posterior*** probability, which evolves with more evidence.

In summary… a Naive Bayes classifier operates under the assumption that one particular feature in a class is unrelated to any other (local Markov Assumption) and that *P* factorizes over some graph *G*, defined by its parents. It is from the Bayesian tribe because algorithm's premise is rooted in Bayes Theorem and probabilities of events. It is named "naive" bayes because it also assumes that each property independently contributes to the probability of a classification being true.

<u>Representation:</u> Graphical Models

- A naive bayes classifier can be represented as a directed acyclic graph where X is a set of random variables and X_i are all vertices in the directed graph.

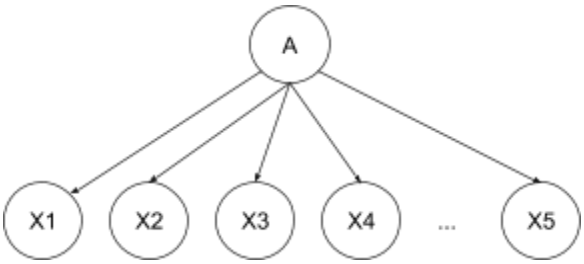
(Expanded:) $P(A,X) = P(A)\prod_{i=1}^{n} P(X_i|A)$ where…

A is a class variable,

X is the set of evidence variables $(X_1, X_2, X_3, ..., X_n)$

And we assume $(X_i \perp X_j | A)\ \forall\ X_i \subseteq X,\ X_{i\neq j} \subseteq X$ … which basically states that we assume each element in the set of evidence variables is independent of one another, related to class *A*, and are themselves a subset less than or equal to *X*.

Acyclic Graph for class A:



In summary, these graphical models demonstrate independencies through causal structures. By querying the model, you can gain different insights which involve prediction (downstream), evidential reasoning (upstream), and inter-causal reasoning (sideways between parents).

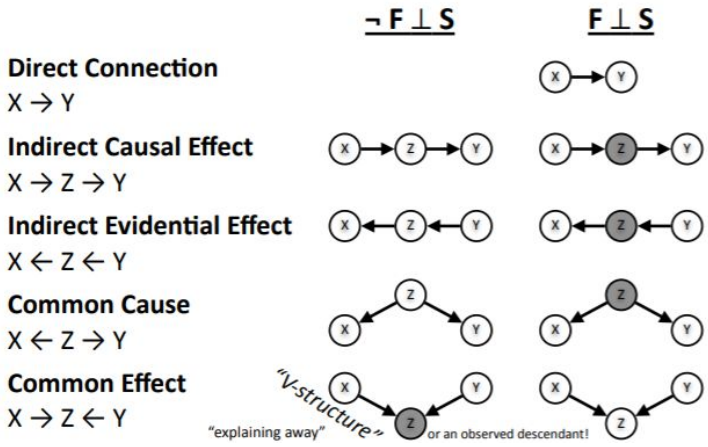


Figure 1.1: Interpreting a Graphical Model of Independencies, (*cs.umass.edu*)

Evaluation: Posterior Probability

- Posterior probability is considered to be an evaluation metric in a naive bayes model. It is the event probability that is updated after an event occurs, taking new information into consideration with each update of the prior probability. ($P(A|B)$, as explained in Bayes Theorem above).

Optimization: Probabilistic Inference

- With probabilistic inference (or bayesian inference), we seek to minimize the expected loss of *X*. This works for environments that are episodic, partially observable, and stochastic (i.e. abiding by a random probability distribution).

(*Expanded:*) Let α be the value predicted by the model and β be the actual value of *X.* The model's loss function is 0 if $\alpha = \beta$, and 1 otherwise.
Expected loss for predicting *X…*

$\sum_{\beta} L(\alpha, \beta)\, P(\beta|e) \ldots$ where *e* is Evidence.

We want to find the value for α that has the greatest posterior probability P(α|e), also known as the Maximum a Posteriori (MAP). To obtain this, we use a map decision to calculate the α of *X* possessing the highest posterior probability given evidence E = *e.* In a model where we are only concerned with one type of observation, our MAP decision is found to be:

$$\hat{x} = \arg\max_x P(X = x \mid E = e) = \frac{P(E = e \mid X = x)P(X = x)}{P(E = e)}$$

$$\propto \arg\max_x P(E = e \mid X = x)P(X = x)$$

$$\underbrace{P(x \mid e)}_{\text{posterior}} \propto \underbrace{P(e \mid x)}_{\text{likelihood}} \underbrace{P(x)}_{\text{prior}}$$

Equation 1.1: MAP Decision for Single Observation, (*cs.illinois.edu*)

And our maximum likelihood is:

$$\hat{x} = \arg\max_x P(e \mid x)$$

Equation 1.2: Maximum Likelihood, (*cs.illinois.edu*)

If we have many observations (E_1 , ..., E_n) that we want to make an inference about for hypothesis *X,* we can use a more inclusive MAP decision:

$$P(X = x \mid E_1 = e_1, \ \ldots, E_n = e_n)$$

$$\propto P(X = x)P(E_1 = e_1, \ \ldots, E_n = e_n \mid X = x)$$

$$\hat{x} = \operatorname{argmax}_x \underbrace{P(x \mid e)}_{\text{posterior}} \propto \underbrace{P(x)}_{\text{prior}} \underbrace{\prod_{i=1}^{n} P(e_i \mid x)}_{\text{likelihood}}$$

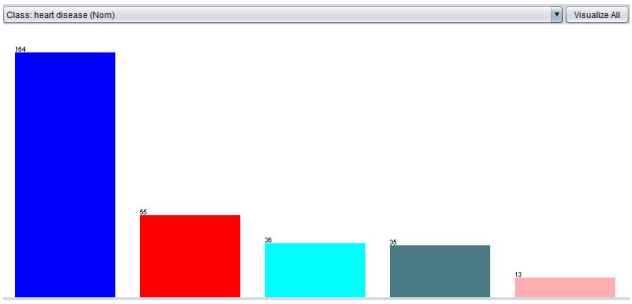
Equations 1.3, 1.4: MAP Decision for Multiple Observations, (*cs.illinois.edu*)

Naive Bayes: Unsplit results

=== Summary === *BEFORE FILTER*

Correctly Classified Instances	196	64.6865 %
Incorrectly Classified Instances	107	35.3135 %
Kappa statistic	0.4461	
Mean absolute error	0.1614	
Root mean squared error	0.3028	

Relative absolute error	62.3404 %
Root relative squared error	84.3229 %
Total Number of Instances	303



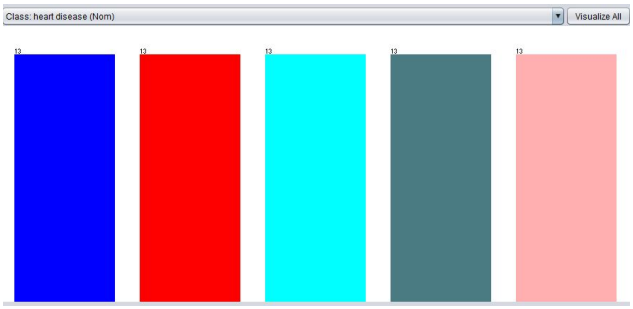
=== Confusion Matrix ===

a	b	c	d	e	<-- classified as
144	14	3	0	3	a = 0
22	18	9	5	1	b = 1
1	11	15	8	1	c = 2
1	9	7	17	1	d = 3
0	5	2	4	2	e = 4

Figure 1.2: Before Class Balancing

=== Summary === *SPREAD SUB-SAMPLE FILTER*

Correctly Classified Instances	44	67.6923 %
Incorrectly Classified Instances	21	32.3077 %
Kappa statistic	0.5962	
Mean absolute error	0.1588	
Root mean squared error	0.2825	
Relative absolute error	49.6135 %	
Root relative squared error	70.6244 %	
Total Number of Instances	65	



=== Confusion Matrix ===

a	b	c	d	e	<-- classified as
10	2	1	0	0	a = 0
4	8	0	0	1	b = 1
0	1	8	3	1	c = 2
1	0	0	9	3	d = 3
1	0	1	2	9	e = 4

Figure 1.3: After Undersampling with *SpreadSubSample* Filter

Model Assessment: This model has a low accuracy even after attempting to balance the classes by undersampling. It's accuracy did improve (~65% → ~68%) after balancing, but by synthesizing results through undersampling, a significant number of instances were lost (303 → 65). That loss, in combination with any missing attributes from the fourth class, is likely to be the primary source of misclassification within this model.

4.) Kernel-Based (Analogist)

Approach: Sequential Minimal Optimization (SMO)

An SMO is a special algorithm for support vector machines (SVM) that solves the quadratic programming problem (QP). The SMO classifier in Weka is an SVM, inclusive of this solution to the QP. Support vectors are vectors that pass between data points lying closest to a hyperplane dividing two classes. The training of an SVM involves optimizing the placement of this decision boundary to attain the highest possible level of accuracy. To do this, the algorithm essentially gets rid of all information that doesn't help to define the frontier. Analogists reason by similarity, therefore SVM's (and by extension SMO's) are one of the primary approaches to classification problems for this tribe. The areas between vectors are areas where data points share a likeness of features and as such, it's intended that they also share the same class.

<u>Representation:</u> Support Vectors

- The decision boundary is comprised of a small subset of training points, the support vectors. *In brief,* support vectors are a subset of the training set that, if removed or altered, would change the position of the dividing hyperplane.

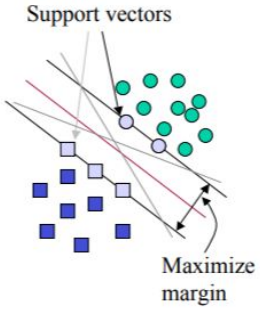


Figure 2.1: Support Vectors, (*web.mit.edu*)

Input: Labelled training data

Output: Set of weights for each feature whose linear combination predicts value *y*

<u>Evaluation:</u> Margin

- The margin is an evaluation metric because the efficacy of an SVM as a classifier is in most cases determined by the accuracy of the margins. To maximize the margin, we minimize the number of nonzero weights that correspond to the important features for the model. These important features define the hyperplane. Nonzero weights correspond to the support vectors by 'supporting' the separating hyperplane. Therefore, the accuracy of the model's support vectors directly correspond to how well defined the margins supporting them are.

Mathematically, “In order to maximize the margin, we thus need to minimize ||w||. With the condition that there are no data points between H_1 and H_2” (*web.mit.edu)*.

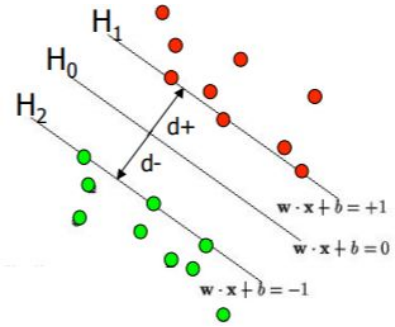


Figure 2.2: Maximizing the Margin, (*web.mit.edu*)

Optimization: Constrained Optimization

- The quadratic programming problem is a constrained optimization problem. A more explicit definition of the problem is: in the process of minimizing ||w||, we are attempting to optimizing a quadratic function of several variables and subject them to linear constraints. While there are a small handful of modern methods for solving the QP (ellipsoid method, equality constraints, and Lagrangian dual functions), the SMO implements Lagrange multipliers - a method of finding local minima and maxima of a function subject to equality constraints (i.e. one or more equations must be satisfied by the chosen values of the variables).

“The algorithm proceeds as follows:

1.) Find a Lagrange multiplier α_1 that violates the Karush–Kuhn–Tucker (KKT) conditions for the optimization problem.
2.) Pick a second multiplier α_2 and optimize the pair (α_1 , α_2)
3.) Repeat steps 1 and 2 until convergence.

When all the Lagrange multipliers satisfy the KKT conditions (within a user-defined tolerance), the problem has been solved.” (*Wikipedia, Sequential minimal optimization*)

SMO: Unsplit results

=== Summary ===

Correctly Classified Instances	54	83.0769 %
Incorrectly Classified Instances	11	16.9231 %
Kappa statistic	0.7885	
Mean absolute error	0.2486	
Root mean squared error	0.3287	
Relative absolute error	77.6923 %	
Root relative squared error	82.1818 %	
Total Number of Instances	65	

Model Assessment: This model performs decently with an ~83% accuracy. This is likely because after balancing the classes (via undersampling) the remaining number of data points is less integral to the model, since the SVM (SMO) emphasizes building a robust frontier to classify the data accurately.

Comparing Approaches Using the Heart Disease Dataset: Unsplit

The dataset appears to classify most accurately via the Random Forest approach, however 100% accuracy is falsely optimistic because it is testing on the same data used to train it, it does not offer any indication of how accurately it might classify new data points. The next highest classification accuracy was the perceptron, which classified data with a ~97% accuracy. This is likely to be the most successful unsplit model at training new data, but I maintain some level of skepticism about its optimism given the small sample population used to train it. The support vector machine with a QP solution (i.e. SMO) classified with an 83% level of accuracy. Given the SVM's focus on possessing a well-defined margin, it is likely that it may lose some accuracy as the sample size grows if the vectors are not allowed to update their positions. Lastly, the naive bayes model performed the worst with ~68% accuracy. This is likely because it was training categorically, when there were some significant continuous variables that should've been discretized for higher accuracy. This could have been done using a Kernel, but for the purpose of the assignment and relating Naive Bayes to its Bayesian roots, I have instead decided to highlight its shortcomings when it does not handle continuous variables as well as it is able to with the aid of a kernel. (For reference, with the use of a Naive Bayes Kernel, the accuracy increases to ~72.3%).

[Completion]: Pipelining & Comprehension of Dataset

Dataset: Cleveland Heart Disease

Business Aspects

The business aspects of this dataset likely involved the value that being able to readily predict the approximate level of cardiac blockage could add to the Cleveland public or private health sector. Having a number of [relatively simple] acquirable biometrics could translate to features used in training the model. Then in training, the model would learn the weights of these attributes and in turn define itself such that it may classify a patient into any of four categories which distinguish the presence (values 1,2,3,4) from absence (value 0) of a cardiac blockage. A model with a high level of accuracy would simplify the diagnosis process, or increase support for an existing diagnosis, so that medical professionals could respond accordingly. Such a model might also provide inferences about a patient by extrapolating some missing values for an instance. This would be especially helpful in the event that an individual is unable to be tested for certain biometrics (e.g. someone who is too sensitive to electrical impulse to receive an ecg).

Data Understanding

The database contains 76 attributes, but published experiments adhere to a subset of 14 of them: age, sex, chest pain, resting blood pressure, cholesterol, fasting blood sugar, resting ecg, maximum heart rate, exercise-induced angine, ST depression, ST slope, major vessels, defect in thallium production, and the label for the presence (or absence) of cardiac blockage. It was interesting that for certain attributes that would normally be continuous, certain nominal categories were established. This is likely to simplify the data points (especially given that there were 76 attributes to one instance in the full dataset), but what made it fascinating was the manner in which numeric data was collected into categories that still expressed values significant to the potential presence (or level) of cardiac blockage. For example, fasting blood sugar was a binary attribute, which expressed whether a patient had a fasting blood sugar level exceeding 120 mg/dl (1 = true, 0 = false). Despite being a binary value, it was still able to retain the information that a value exceeding specifically 120mg/dl was significant in the feature's label, whereas opposed one might expect treatment of a patient's fasting blood sugar level as a numeric and continuous data type ranging from 80-126+. As a final point, it should be noted that the WARNING.txt file in the data folder contained the information that the full Cleveland dataset had somehow been corrupted and that missing values were replaced with the value '-9'. This was interesting, but not entirely noteworthy given that the subset of 14 attributes was used for training.

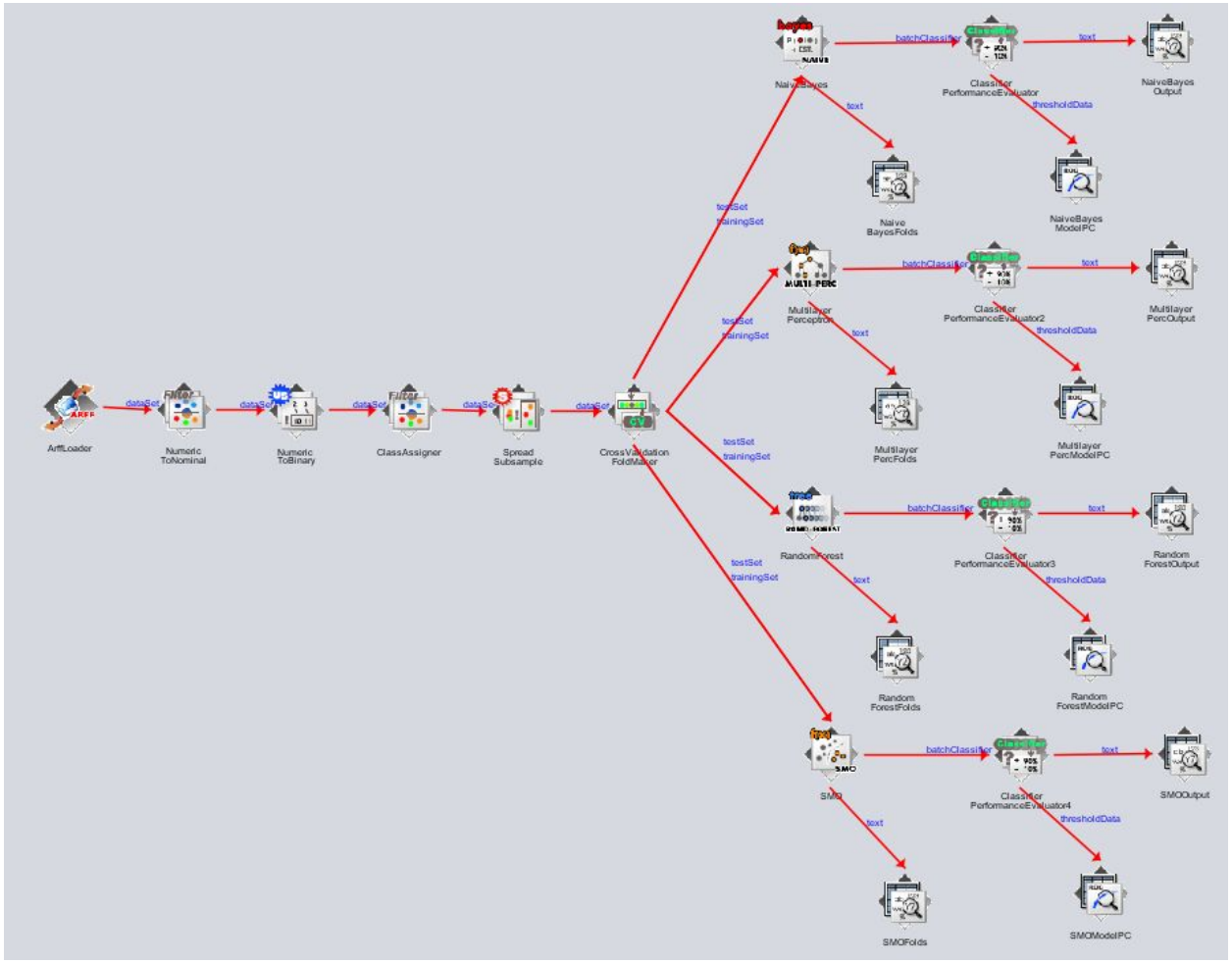
Data Preparation

The pipeline can assist in the preparation of the data prior to a technique being applied. Weka Knowledge Flow offers the same functionality as the Explorer for filtering and classifying data, and is accompanied by the visual representation for the pipeline. The pipeline is identical for the data preprocessing steps for each of the algorithms that I chose. It is possible that I would

consider not balancing the classes by undersampling the data for certain approaches so as not to put further limitations on the learner.

Modelling

This pipeline could suit one or more of the five tribes of AI, demonstrated by fitting it to each of the previously mentioned algorithms. The Naive Bayes approach could fall under the Bayesian or Analogist tribe if an assisting kernel is applied to the learner, but in this pipeline it is not given a kernel.



Evaluation

This pipeline supports only one method of evaluation, k-fold cross validation. The number of folds selected for this pipeline was 5. It is important to not test a model on its exact training data because this can generate overly optimistic and overfitted models. The two methods of evaluating models in data science are Cross-Validation and Hold-Out. In k-fold cross-validation, the data is divided into k-number of subsets of equal size. The model is built k-number of times, omitting one of k subsets upon each iteration.

Deployment

Deployment is the way that a machine learning model is integrated into a pre-existing production environment. Model deployment is often the most difficult stage of the machine learning life cycle because it requires coordination among all the involved parties (software developers, IT teams, etc.) to safely and reliably integrate it into the production environment. This is rarely a simple task, regardless of the simplicity of the model. For example, recoding may need to be done if a model is written in a different language to the environment.

Evidence of the Generation of Deployable Knowledge:

Status (with class balancing):

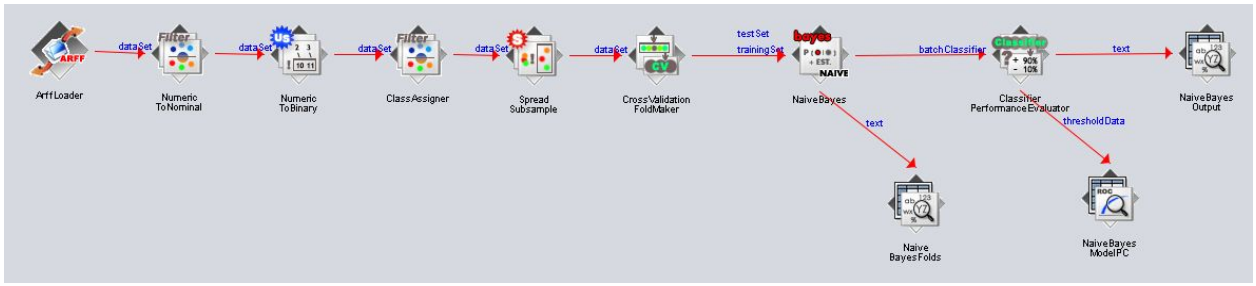
Component	Parameters	Time	Status
[KnowledgeFlow]		-	OK.
ArffLoader		-	Finished.
NumericToNominal	-R 3,7,11-14	-	Finished.
NumericToBinary	-R 2,6,9	-	Finished.
ClassAssigner	-C last	-	Finished.
SpreadSubsample	-M 0.0 -X 0.0 -S 1	-	Finished.
CrossValidationFoldMaker		-	Finished.
NaiveBayes		-	Finished.
MultilayerPerceptron	-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a	00:00:01	Finished.
RandomForest	-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1	-	Finished.
SMO	-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.fu...	-	Finished.
NaiveBayesFolds		-	Finished.
ClassifierPerformanceEvaluator		-	Finished.
SMOFolds		-	Finished.
RandomForestFolds		-	Finished.
ClassifierPerformanceEvaluator4		-	Finished.
ClassifierPerformanceEvaluator3		-	Finished.
NaiveBayesModelPC		-	Finished.
NaiveBayesOutput		-	Finished.
SMOOutput		-	Finished.
SMOModelPC		-	Finished.
RandomForestOutput		-	Finished.
RandomForestModelPC		-	Finished.
MultilayerPercFolds		-	Finished.
ClassifierPerformanceEvaluator2		-	Finished.
MultilayerPercOutput		-	Finished.
MultilayerPercModelPC		-	Finished.

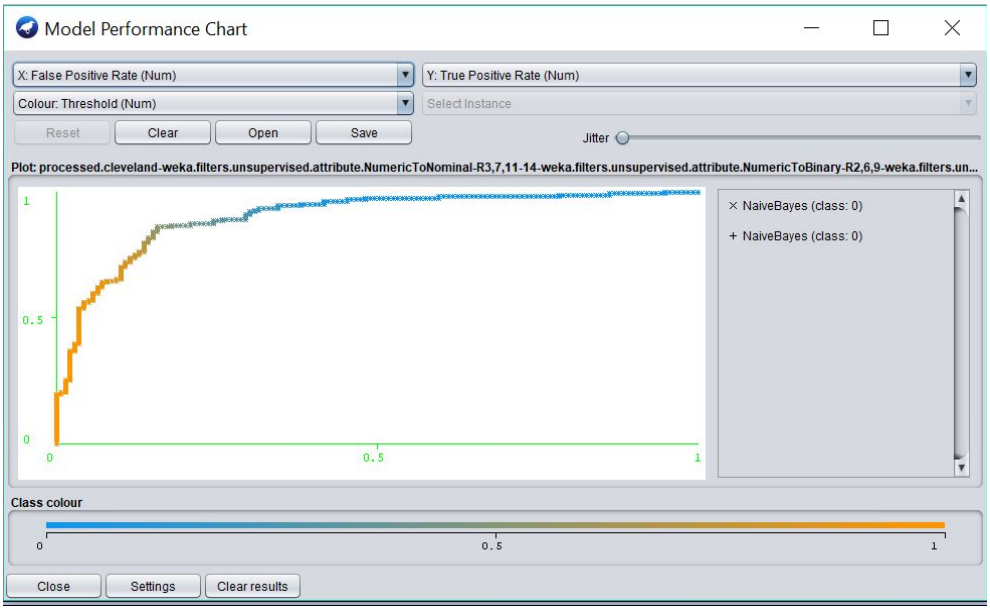
Status (without class balancing):

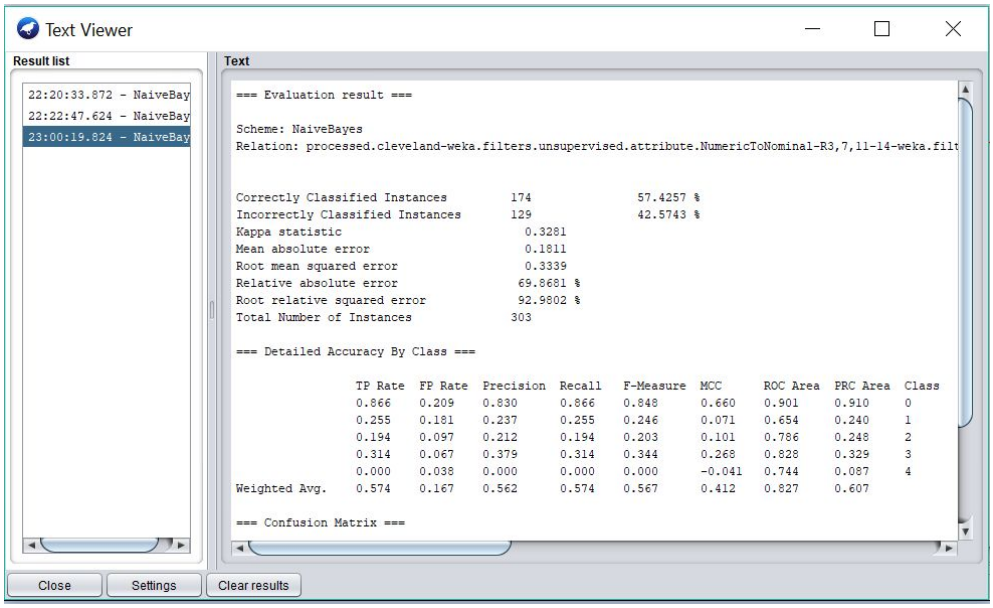
Component	Parameters	Time	Status
[KnowledgeFlow]		-	OK.
ArffLoader		-	Finished.
NumericToNominal	-R 3,7,11-14	-	Finished.
NumericToBinary	-R 2,6,9	-	Finished.
ClassAssigner	-M 1.0 -X 0.0 -S 1	-	Finished.
CrossValidationFoldMaker		-	Finished.
RandomForest	-P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1	-	Finished.
NaiveBayes		-	Finished.
MultilayerPerceptron	-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a	00:00:02	Finished.
SMO	-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.fu...	-	Finished.
ClassifierPerformanceEvaluator		-	Finished.
NaiveBayesFolds		-	Finished.
RandomForestFolds		-	Finished.
SMOFolds		-	Finished.
ClassifierPerformanceEvaluator3		-	Finished.
ClassifierPerformanceEvaluator4		-	Finished.
NaiveBayesModelPC		-	Finished.
NaiveBayesOutput		-	Finished.
SMOOutput		-	Finished.
SMOModelPC		-	Finished.
RandomForestOutput		-	Finished.
RandomForestModelPC		-	Finished.
MultilayerPercFolds		-	Finished.
ClassifierPerformanceEvaluator2		-	Finished.
MultilayerPercOutput		-	Finished.
MultilayerPercModelPC		-	Finished.

Comparing Naive Bayes Results Using Different Tools

Pipeline:







Readable Classifier Output:

=== Classifier model ===

Scheme: NaiveBayes
Relation:
processed.cleveland-weka.filters.unsupervised.attribute.NumericToNominal-R3,7,11-14-weka.filters.unsupervised.attribute.NumericToBinary-R2,6,9-weka.filters.unsupervised.attribute.ClassAssigner-Clast-weka.filters.supervised.instance.SpreadSubsample-M0.0-X0.0-S1

Naive Bayes Classifier

	Class				
Attribute	0	1	2	3	4
	(0.53)	(0.18)	(0.12)	(0.12)	(0.04)
age					
mean	52.6874	55.8549	57.7135	55.9945	57.9949
std. dev.	9.4366	7.2982	6.9387	7.1597	9.6397
weight sum	131	44	29	28	10
precision	1.1026	1.1026	1.1026	1.1026	1.1026
sex_binarized					
0	57.0	9.0	7.0	7.0	2.0
1	76.0	37.0	24.0	23.0	10.0
[total]	133.0	46.0	31.0	30.0	12.0
chest pain					
1	15.0	5.0	2.0	1.0	2.0
2	32.0	6.0	2.0	2.0	1.0
3	56.0	8.0	3.0	4.0	2.0
4	32.0	29.0	26.0	25.0	9.0
[total]	135.0	48.0	33.0	32.0	14.0
resting bp					
mean	128.8183	134.1061	134.023	136.1175	137.3289
std. dev.	16.1661	19.3555	17.6203	20.5194	18.4592
weight sum	131	44	29	28	10
precision	2.3556	2.3556	2.3556	2.3556	2.3556
cholesterol					
mean	243.7544	249.7485	259.4437	245.8825	272.5333
std. dev.	55.6399	37.311	57.7593	53.5815	58.4721
weight sum	131	44	29	28	10
precision	3.2444	3.2444	3.2444	3.2444	3.2444
fasting blood sugar_binarized					
0	112.0	42.0	23.0	22.0	10.0
1	21.0	4.0	8.0	8.0	2.0
[total]	133.0	46.0	31.0	30.0	12.0
resting ecg					
0	80.0	21.0	19.0	10.0	2.0
1	2.0	1.0	2.0	2.0	2.0
2	52.0	25.0	11.0	19.0	9.0
[total]	134.0	47.0	32.0	31.0	13.0

max hr

mean	158.0152	147.5791	134.6946	132.7831	146.1904
std. dev.	19.49	21.9716	17.7718	21.7744	18.0316
weight sum	131	44	29	28	10
precision	1.2892	1.2892	1.2892	1.2892	1.2892

exercise angina_binarized

0	115.0	27.0	10.0	11.0	7.0
1	18.0	19.0	21.0	19.0	5.0
[total]	133.0	46.0	31.0	30.0	12.0

st depression

mean	0.5798	0.959	1.8944	1.9683	2.3767
std. dev.	0.7714	0.9882	1.161	1.6113	1.373
weight sum	131	44	29	28	10
precision	0.1722	0.1722	0.1722	0.1722	0.1722

st slope

1	86.0	19.0	7.0	7.0	2.0
2	41.0	25.0	21.0	18.0	9.0
3	7.0	3.0	4.0	6.0	2.0
[total]	134.0	47.0	32.0	31.0	13.0

major vessels

0	108.0	24.0	9.0	7.0	3.0
1	16.0	13.0	12.0	6.0	3.0
2	6.0	7.0	8.0	13.0	3.0
3	3.0	3.0	4.0	6.0	5.0
[total]	133.0	47.0	33.0	32.0	14.0

thallium defect

3	100.0	19.0	6.0	6.0	2.0
6	7.0	2.0	5.0	2.0	3.0
7	26.0	26.0	20.0	23.0	8.0
[total]	133.0	47.0	31.0	31.0	13.0

Explorer:

(Using the same methods of pre-processing the data as in the pipeline, but classifying without the k-fold cross validation.)

=== Summary === (*UNSPLIT NAIVE BAYES MODEL*)

Correctly Classified Instances	196	64.6865 %
Incorrectly Classified Instances	107	35.3135 %
Kappa statistic	0.4461	
Mean absolute error	0.1614	
Root mean squared error	0.3028	
Relative absolute error	62.3404 %	
Root relative squared error	84.3229 %	
Total Number of Instances	303	

=== Summary === (*WITH 5-FOLD CROSS VALIDATION*)

Correctly Classified Instances	172	56.7657 %
Incorrectly Classified Instances	131	43.2343 %
Kappa statistic	0.3209	
Mean absolute error	0.1825	
Root mean squared error	0.3352	
Relative absolute error	70.4068 %	
Root relative squared error	93.3429 %	
Total Number of Instances	303	

Accuracy of the Models: (Pipeline v/s Unsplit), (Pipeline v/s Explorer)
The accuracy of the Naive Bayes model decreased from 64.6% (unsplit) to 56.7% after 5-fold cross validation. This is because the model that the data used to train on was also used for testing in the unsplit version. Having models test on training data leads to overly optimistic results, as was the case in the unsplit model's 64.6% accuracy. The variation in accuracy between the pipeline and the explorer is >1%, meaning it is most likely normal, insignificant variation stemming from the k-fold subsets. It would be possible to reduce this disparity by increasing the number of folds insofar as the model allows.

After using the Pipeline approach, the accuracy decreased as a result of the 5-fold cross validation that wasn't previously integrated into the classification process. Snapshots of the data pipeline(s) are provided above.

References

https://archive.ics.uci.edu/ml/datasets/heart+Disease

https://medium.com/42ai/the-5-tribes-of-the-ml-world-670ebce96b4c

https://en.wikipedia.org/wiki/Lagrange_multiplier

https://en.wikipedia.org/wiki/Quadratic_programming

https://en.wikipedia.org/wiki/Sequential_minimal_optimization

https://en.wikipedia.org/wiki/ST_depression

https://www.ncbi.nlm.nih.gov/pubmed/9714106

http://slazebni.cs.illinois.edu/fall17/lec13_bayesian_inference.pdf

https://people.cs.umass.edu/~mccallum/courses/gm2011/02-bn-rep.pdf

https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

https://www.saedsayad.com/model_evaluation.htm

https://www.datarobot.com/wiki/machine-learning-model-deployment/

REFTree (Compatible Program Code)

```
// Generated with Weka 3.8.3
//
// This code is public domain and comes with no warranty.
//
// Timestamp: Mon Jul 29 23:23:50 NZST 2019

package weka.classifiers;

import weka.core.Attribute;
import weka.core.Capabilities;
import weka.core.Capabilities.Capability;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.RevisionUtils;
import weka.classifiers.Classifier;
import weka.classifiers.AbstractClassifier;

public class WekaWrapper
  extends AbstractClassifier {

  /**
   * Returns only the toString() method.
   *
   * @return a string describing the classifier
   */
  public String globalInfo() {
    return toString();
  }

  /**
   * Returns the capabilities of this classifier.
   *
   * @return the capabilities
   */
  public Capabilities getCapabilities() {
    weka.core.Capabilities result = new weka.core.Capabilities(this);

    result.enable(weka.core.Capabilities.Capability.NOMINAL_ATTRIBUTES);
    result.enable(weka.core.Capabilities.Capability.NUMERIC_ATTRIBUTES);
    result.enable(weka.core.Capabilities.Capability.DATE_ATTRIBUTES);
```

```
  result.enable(weka.core.Capabilities.Capability.MISSING_VALUES);
  result.enable(weka.core.Capabilities.Capability.NOMINAL_CLASS);
  result.enable(weka.core.Capabilities.Capability.NUMERIC_CLASS);
  result.enable(weka.core.Capabilities.Capability.DATE_CLASS);
  result.enable(weka.core.Capabilities.Capability.MISSING_CLASS_VALUES);


  result.setMinimumNumberInstances(1);

  return result;
}

/**
 * only checks the data against its capabilities.
 *
 * @param i the training data
 */
public void buildClassifier(Instances i) throws Exception {
  // can classifier handle the data?
  getCapabilities().testWithFail(i);
}

/**
 * Classifies the given instance.
 *
 * @param i the instance to classify
 * @return the classification result
 */
public double classifyInstance(Instance i) throws Exception {
  Object[] s = new Object[i.numAttributes()];

  for (int j = 0; j < s.length; j++) {
    if (!i.isMissing(j)) {
      if (i.attribute(j).isNominal())
        s[j] = new String(i.stringValue(j));
      else if (i.attribute(j).isNumeric())
        s[j] = new Double(i.value(j));
    }
  }

  // set class value to missing
  s[i.classIndex()] = null;
```

```
    return WekaClassifier.classify(s);
  }

  /**
   * Returns the revision string.
   *
   * @return        the revision
   */
  public String getRevision() {
    return RevisionUtils.extract("1.0");
  }

  /**
   * Returns only the classnames and what classifier it is based on.
   *
   * @return a short description
   */
  public String toString() {
    return "Auto-generated classifier wrapper, based on weka.classifiers.trees.REPTree
(generated with Weka 3.8.3).\n" + this.getClass().getName() + "/WekaClassifier";
  }

  /**
   * Runs the classfier from commandline.
   *
   * @param args the commandline arguments
   */
  public static void main(String args[]) {
    runClassifier(new WekaWrapper(), args);
  }
}

class WekaClassifier {

  public static double classify(Object [] i)
    throws Exception {

    double p = Double.NaN;
    p = WekaClassifier.N35547d3f0(i);
    return p;
  }
  static double N35547d3f0(Object []i) {
    double p = Double.NaN;
```

```
  /* chest pain */
  if (i[2] == null) {
    p = 0;
  } else if (i[2].equals("1")) {
    p = 0;
  } else if (i[2].equals("2")) {
    p = 0;
  } else if (i[2].equals("3")) {
    p = 0;
  } else if (i[2].equals("4")) {
  p = WekaClassifier.N13b9c4ed1(i);
  }
  return p;
}
static double N13b9c4ed1(Object []i) {
  double p = Double.NaN;
  /* major vessels */
  if (i[11] == null) {
    p = 0;
  } else if (i[11].equals("0")) {
  p = WekaClassifier.N92561832(i);
  } else if (i[11].equals("1")) {
  p = WekaClassifier.N7114a5955(i);
  } else if (i[11].equals("2")) {
    p = 3;
  } else if (i[11].equals("3")) {
    p = 4;
  }
  return p;
}
static double N92561832(Object []i) {
  double p = Double.NaN;
  /* thallium defect */
  if (i[12] == null) {
    p = 0;
  } else if (i[12].equals("3")) {
    p = 0;
  } else if (i[12].equals("6")) {
    p = 0;
  } else if (i[12].equals("7")) {
  p = WekaClassifier.N3afcadbb3(i);
  }
  return p;
```

```
}
static double N3afcadbb3(Object []i) {
  double p = Double.NaN;
  /* cholesterol */
  if (i[4] == null) {
    p = 1;
  } else if (((Double)i[4]).doubleValue() < 302.0) {
  p = WekaClassifier.N2c57825e4(i);
  } else if (true) {
    p = 2;
  }
  return p;
}
static double N2c57825e4(Object []i) {
  double p = Double.NaN;
  /* max hr */
  if (i[7] == null) {
    p = 1;
  } else if (((Double)i[7]).doubleValue() < 133.0) {
    p = 3;
  } else if (true) {
    p = 1;
  }
  return p;
}
static double N7114a5955(Object []i) {
  double p = Double.NaN;
  /* cholesterol */
  if (i[4] == null) {
    p = 1;
  } else if (((Double)i[4]).doubleValue() < 282.5) {
  p = WekaClassifier.N539ffb4d6(i);
  } else if (true) {
    p = 2;
  }
  return p;
}
static double N539ffb4d6(Object []i) {
  double p = Double.NaN;
  /* age */
  if (i[0] == null) {
    p = 1;
  } else if (((Double)i[0]).doubleValue() < 60.5) {
```

```
    p = WekaClassifier.Nc5105f07(i);
    } else if (true) {
      p = 2;
    }
    return p;
  }
  static double Nc5105f07(Object []i) {
    double p = Double.NaN;
    /* cholesterol */
    if (i[4] == null) {
      p = 1;
    } else if (((Double)i[4]).doubleValue() < 212.0) {
    p = WekaClassifier.N1c0423568(i);
    } else if (true) {
      p = 1;
    }
    return p;
  }
  static double N1c0423568(Object []i) {
    double p = Double.NaN;
    /* thallium defect */
    if (i[12] == null) {
      p = 3;
    } else if (i[12].equals("3")) {
      p = 1;
    } else if (i[12].equals("6")) {
      p = 1;
    } else if (i[12].equals("7")) {
      p = 2;
    }
    return p;
  }
}
```