# "Hello... That's Odd", or Why we Should Teach AI in First Year

Tim Menzies
CSEE, WVU
tim@menzies.us*

February 22, 2006

In the view of this paper, students should learn that humans should not be passive observers of computer output. Rather, computers should are be viewed as active assistants to everyday human activities that help us; i.e. watch what we watch and point out what we might have missed. So instead of teaching

$$hello\ world$$

the canonical example first presented to students should be

$$hello....\ that's\ odd$$

*Hello... that's odd* was inspired by Lucy Stein's interaction perspective [11, 12]. She claims that *hello world* reflects a von Neumann view of serial computing where "computation is the process of executing those steps – the algorithm – to deduce the answer to a particular question".

Stein argues convincingly that this von Neumann view is an incomplete model of modern computing where "computation as a living, breathing thing that exists and coexists in a dynamic continuous parallel world". For example, even building a simple spreadsheet with standard business software is an exploratory task where users explore alternate scenarios and different layouts to discover the best way to analyze and present business knowledge. Her preferred canonical introductory example is

$$while\ true\ \{echo()\}$$

i.e. an ongoing process which samples its environment and responds to it.

The intent of *Hello... that's odd* is to extend Stein's perspective with data mining concepts and Boehm's concepts of values-based engineering [1, 2]. In order for a computer program to understand *odd*, it needs to understand what is *usual*; what is *expected*; what is *preferred* and what is *undesired*. Notions of *preferred* and *undesired* can only be defined with respect to the values of some human observer. That is, central to *hello...that's odd*, is a user model of preferences and goals.

## 1 Details

Consider what would be required to realize *hello, that's odd*:

1. A world model must be built summarizing what has been seen so far;
2. A user model must be built summarizing user beliefs and desires;
3. An anomaly alert must trigger if new inputs fall outside what has been seen so far;

There are complex ways and simple ways to satisfy these three requirements. Manual construction of world models is complex and should be deferred to upper-year data modeling subjects. On other other hand, for a first year subject, a simpler method would be to take a data set containing instances that some oracle has already classified into classes (a.k.a. modes). Each mode should be scored with an integer indicating the utility of that class: positive numbers for preferred outcomes and negative numbers for undesired outcomes.

This paper takes the simpler approach. Given a Bayesian classifier, and integer class utilities, the above three requirements can be meet very easily using a Bayes classifier. Bayes classifiers are based on Bayes' Theorem. Informally, the theorem says $next = old * new$ i.e. what we'll believe *next* comes from how *new* evidence effects *old* beliefs. More formally:

$$P(H \mid E) = \frac{P(H)}{P(E)} \prod_i P(E_i \mid H)$$

That is, given fragments of evidence $E_i$ and a prior probability for a class $P(H)$, the theorem lets us calculate a posteriori

*Submitted to 19th Conference on Software Engineering Education and Training Turtle Bay, North Shore Oahu, Hawaii April 19-21, 2006 http://db-itm.cba.hawaii.edu/cseet2006.

†Earlier drafts of this paper are available from http://menzies.us/pdf/06odd101.pdf.

```
# GLOBALS:
# ''F'': frequeny tables; ''I'' : number of instances;
# ''C'': how many classes?; ''N'': instances per class

function update(class,train)
   # OUTPUT: changes to the globals.
   # INPUT:  a ''train''ing example of attr/values
  #         plus that case's ''class''
  I++;
   if (++N[class]==1) then C++ fi
   for <attr,value> in train
      if (value != "?") then
         F[class,attr,value]++ fi

function classify(test)
   # OUTPUT: ''what'' is the most likely hypothesis for the test case.
   # INPUT:  a ''test'' case containing attribute/value pairs.
   k=1; m=2              # Control for Laplace and M-estimates.
   like = -100000        # Initial, impossibly small likelihood.
   for H in N            # Check all  hypotheses.
   { prior = (N[H]+k)/(I+(k*C))
     temp  = log(prior)
     for <attr,value> in attributes
     { if (value != "?") then
       inc= F[H,attr,value]+(m*prior))/(N[H]+m)
       temp += log(inc) fi
     }
     if (temp >= like) then
       like = temp; what=class fi
   }
   return what
```

Figure 1: A Bayes Classifier. "?" denotes "missing values". Probabilities are multiplied together using logarithms to stop numeric errors when handling very small numbers. The $m$ and $k$ variables handle low frequencies counts in the manner recommended by Yang and Webb [14, §3.1].

probability $P(H \mid E)$.

Figure 1 shows the pseudo-code for such a learner. The function `update` illustrates the simplicity of re-learning for a Bayes classifier: just increment a frequency table $F$ holding counts of the attribute values seen in the new training examples. The function `classify` returns the most likely class; i.e. the one with the largest product of the individual frequencies times the prior probability of that class.

Since Figure 1 is a classifier, it supports several more interesting functions:

4. It can classify new inputs using the world model;
5. It can raise a problem alert if new inputs fall into some undesired class of the world model.

Also, based on some my own recent research [9], I assert that with minor modifications, the code can support:

6. Anomaly alerts when new input falls outside of the envelope of experience since too date.

# 2 Details

The rest of this paper makes the case that such Bayes learners are suitable for first-year teaching.

## 2.1 Simplicity

As witnessed by the proceedings of the *Uncertainty in AI* conference, the general field of Bayesian learning is on the most active research frontier of AI. Such an actives, dynamic field should not be taught to first years. Instead, this proposal is based around the more mature and more simpler concept of a Bayes classifier [5].

Bayes classifiers like Figure 1 are called *näive* since they assume that the frequencies of different attributes are independent. However, many studies [4, 6] have reported that, in many domains, this simple Bayes classification scheme exhibits excellent performance A recent theoretical analysis reveals why: Bayes classifiers gets confused by attribute dependencies in a vanishingly small percent of cases [3].

The code of Figure 1 is so simple that it could be taught to first-years:

- The core data structure is a single array $F$ storing the frequency counts.
- Unlike other learners (e.g. decision tree learners), there is no need to explore more complex issues such a recursion in order to build a running version.

## 2.2 Using Bayes Methods

The six requirements of § 1 could then be fulfilled using Figure 1 or small extensions to that code:

*1. A world model must be built summarizing what has been seen so far:* Bayes classifiers can store prior experience as distributions of the frequency counts.

*2. A user model must be built summarizing user beliefs and desires:* In the general case, this is complex issue. However, much can be done with a simple goal model where each class gets assigned an integer (positive for preferred goals, and negative for undesired goals). Further, using the anomaly detection method described below, it is possible to determine when a new class has entered the data set. That is, this approach can generalize to not only prior data that has been classified, but also novel data sets with unknown classes.

*3. An anomaly alert must trigger if new inputs fall outside what has been seen so far:* Anomalies mean "we have not seen this before". The frequency counts of a Bayes classifier can be

used to implement such an anomaly detector. Internal to the classifiers is the calculation of the likelihood; i.e. the product of the frequencies of the new example in distribution counts. A new example would be declared anomalous if this likelihood calculation dips significantly lower than prior likelihood calculations. Experiments with this approach suggest that the dip can be quite dramatic (an order or two of magnitude).

*4. A classifier that can classify new inputs the world model:* Such a classifier comes built-in to Bayes learners like Figure 1.

*5. Raise a problem alert if new input fall into some undesired class of the world model:* Given classes with positive and negative utilities (where positive/negative utilities indicate preferred/undesired classes respectively) then such a trigger could be generated if the Bayes learner classifies a new example as an undesired class.

*6. Some action that responds to the alerts (be they be problem alerts or anomaly alerts:* Such actions can be easily computed using the frequency counts of a Bayes classifier: just look for some value for some controllable attribute that is far more common in preferred than undesired classes. This kind learning of control actions is a simple way of instantiating Boehm's value-based SE proposal [1]. Rather than treat all goals as value-neutral and of equal value, this methods focus selectively on issues that most select for classes that users have scored with highest utility.

Note that once #3 is implemented, students would have a device that can say *hello... that's odd*. Also, once #6 is implemented, students would have an advisor that can suggest remedies to undesirable or anomalous situations.

## 2.3 Other Topics

Prior to teaching Figure 1 , I would introduce students to the idea of interacting with data with some lectures from Tufte's work on information design [13]. Tufte, like Stein, seeks to empower users to take and watch and improve their interaction with the Tufte explains how to visually show cause and effect, how to insure that the proper comparisons are made, and how to achieve the (valid) goals that are desired. His text reviews how information has been presented graphically through-out history and his examples range from how to present baseball scores [1] to project management graphics [2] to how Power Point slides caused a fatal crash of the Space Shuttle[3].

[1] `http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0001OR&topic_id=1`

[2] `http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=000076&topic_id=1`

[3] `http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=0001yB&topic_id=1`

Figure 1 could be used to introduce the topics shown below; i.e. databases, statistics, ethics, and systems-level programming in general and software testing in particular:

**Databases:** Data miners learn from data and data is stored in databases. A natural parallel subject would be an introduction to relational databases and a natural overlapping project would be to extract data from a database (via a join over a few tables), some data mining, then writing back the learned classifications to the database.

**Statistics:** A Bayes classifier needs Gaussians for handling numerics[4]. Comparing the performance of different data miners needs t-tests. Introducing students to Gaussians and their operations could happen in a first-year statistics subject.

**Ethics:** The conclusions reached by a learner are biased in numerous ways (by the data selected for learning, by the search strategy of the learner, by the choice of target language for the learned theory, etc). That is, the results are not some universal truth but some context-dependent summary. Hence, responsible data miners present their conclusions along with their biases. Case studies could easily be developed where data miners mislead or confused users. Students could be challenged to prepare presentations that responsibly present their conclusions to users.

**Systems-level Programming:** Leveson remarks that in modern complex systems, unsafe operations often result from an unstudied interaction *between* components [7][5]. Hence, it is vital that we teach students a *systems-level* perspective; i.e. understanding a component means not only understanding the internals of their program but also the way that program interacts with the environment.

Learning AI algorithms is one way to teach that systems-level perspective. One subtlety in AI algorithms like Figure 1, or the interpreter for a rule-based system, is that they are *data-driven*. When software bugs manifest, debugging means understanding both the code *and* the data feed into the code. For example, if Figure 1 fails to make accurate classifications, the reason is often in the data and not in the algorithm (e.g. noise in the data, some classes are very under-sampled in the training data seen to date, etc). Hence, debugging Figure 1 is not just a matter of staring at the code. Rather, students would need to understand how the code interacts with its surrounding environment.

**Software Testing:** In one specialized area, this notion of systems-level testing can be very useful indeed. Modern software is so complex that exhaustive testing of all inputs is im-

[4] Technically, this is not precisely true. While a standard Bayes classifier uses Gaussians, a common technique is to *discretize* numerics prior to learning [4].

[5] Lutz and Mikulski [8] found one such interaction in NASA deep-space satellites: mission critical anomalies of *flight software* can result from errors in *ground software* that fails to correctly collect data from the flight systems.

practical. Rather, software must be certified with respect to an *operational profile*[6]. Testing via operational profiles is practical since the profile constrains the space of possible tests to a manageable number.

Figure 1 has much to offer testing via operational profiles. The following operations are simple enough to teach to first year:

***Building a profile:*** The manual construction of an operational profile can be time-consuming and error-prone task. On the other hand, the frequency counts maintained by Figure 1 are automatically created from examples seen to date.

***Nominal testing:*** Operational profile testing can be *nominal*; i.e. the tests are drawn from the profile. Students could use the $F$ variable of *class* to generate nominal tests.

***Off-nominal testing:*** *Off-nominal* testing explores values outside the expected range. Students could use the $F$ variable of *class* to generate off-nominal tests by first complementing the frequency counts inside $F$.

***Learning the certification envelope:*** Operational profile testing certifies a device within some profile. If a device leaves that profile then the certificate is no longer valid. *Hello... that's odd* could be used to determine when the inputs to some software are departing from the envelope of inputs under which a device was certified.

# 3 Discussion

Consider the changing nature of the information worker in the $21^{st}$ century. Due to the Internet, an potentially overwhelmingly amount of data is available on any issue. Much of that data will be superfluous and confusing. Finding "the diamonds in the dust" (i.e. extracting succinct descriptions of the relevant sections of large data sets) will be a routine daily task faced by anyone with access to a computer. We should therefore teach "diamond mining" to first year students.

At first glance, this might appear too ambitious. However, after teaching data mining for five years, I assert that the core algorithms of certain data miners are very simple indeed and can easily and quickly be extended to all the above tasks.

# References


[1] B. Boehm. Keynote address: Automating Value-Based Software Engineering, IEEE conference on Automated Software Engineering, 2004. Available from `http://ase.cs.uni-essen.de/ase/past/ase2004/download/KeynoteBoehm.pdf`.

[2] B. Boehm, A. Egyed, D. Port, A. Shah, J. Kwan, and R. Madachy. A Stakeholder Win-Win Approach to Software Engineering Education. *Annals of Software Engineering*, 6:295–321, 1998.

[3] Pedro Domingos and Michael J. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(2-3):103–130, 1997.

[4] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *International Conference on Machine Learning*, pages 194–202, 1995. Available from `http://www.cs.pdx.edu/~timm/dm/dougherty95supervised.pdf`.

[5] R. Duda, P. Hart, and N. Nilsson. Subjective bayesian methods for rule-based inference systems. In *Technical Report 124, Artificial Intelligence Center, SRI International*, 1976.

[6] M.A. Hall and G. Holmes. Benchmarking Attribute Selection Techniques for Discrete Class Data Mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003.

[7] N. Leveson. *Safeware System Safety And Computers*. Addison-Wesley, 1995.

[8] R. Lutz and Carmen Mikulski. Operational Anomalies as a Cause of Safety-Critical Requirements Evolution. *Journal of Systems and Software (to appear)*, 2003. Available from `http://www.cs.iastate.edu/~rlutz/publications/JSS02.ps`.

[9] Tim Menzies and Andres Orrego. Incremental Discreatization and Bayes Classifiers Handles Concept Drift and Scaled Very Well, 2005. Available from `http://menzies.us/pdf/05sawtooth.pdf`.

[10] John Musa. *Software Reliability Engineered Testing*. McGraw-Hill, 1998.

[11] L. Stein. Challenging the computational metaphor: Implications for how we think, 1999.

[12] L.A. Stein. Rethinking CS101: Or, How Robots Revolutionize Introductory Computer Programming, 1996.

[13] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2nd edition, 2001.

[14] Y. Yang and G. Webb. Weighted Proportional k-Interval Discretization for Naive-Bayes Classifiers. In *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003)*, 2003.


[6] Technically, an operational profile is the probability that certain attribute ranges will appear on input [10].

Available from http://www.csse.monash.edu/~webb/Files/YangWebb03.pdf.