Introduction

In general variable valuation is estimating the utility that a column of explanatory values `x` will have in predicting a column of dependent values `y`. It is quantifying the strength of the relation between `x` and `y`. If this were asked in isolation, this would be easy (essentially just compute something like a correlation). However, this is usually asked in the context of more than one possible explanatory variable and a variety of possible modeling strategies.

In our opinion, variable valuation and selection is an often necessary step. It is useful when your machine learning modeling is in danger of being overwhelmed by values in irrelevant columns (such as our Bad Bayes examples). These situations often arise in practice in data science projects, as data scientists are often using wide tables or data-marts with hundreds of columns that were designed for many users. Such tables often have many columns that are not in, a statistical sense, useful variables for a given task.

One might argue it is unlikely there are hundreds of causal columns for a given project. And this is likely true. However, for many data science projects we are attempting to estimate `P[y = TRUE]` from side-effects or correlated-attribute columns, and it is typical for an online record keeping system to have hundreds of these columns and attributes. Thus it is common to work with many columns, many being relevant variables and many not.

Our stance is: we need variable valuation, but we can't lean on it too hard. Roughly, variable valuation is not well founded, unless stated with respect to a given generative model, data set, model fitting procedure, and even a given model. This is why our `vtreat` variable preparation package (`vtreat` for `R` here, `vtreat` for Python here) package defers on the issue, and uses only simple linear methods for *light variable screening* (delete as few variables as possible, and have the option to not even do that).

What a variable is worth, unfortunately, depends on who is asking.

Practical Solutions

In our opinion the most successful variable valuation tools are the permutation importance tests (such as those found in random forest and `xgboost`) that compute the utility of a variable with respect to a single fit model. They do this by damaging the variable, and seeing how the fixed model predictions then degrade. Useless variables inflict little damage, critical variables inflict a lot of damage.

These solutions are very usable, efficient, and give good practical advice. Furthermore, they give, in addition to a useful variable ranking, a quantified value for each variable (how much quality the model loses if lied to about the values of the variable in question).

However, these methods *are* measuring the value of variable *with respect to a single, already fit model*. So they are possibly not measuring intrinsic variable importance or even true variable importance with respect to a single model fitting procedure.

To emphasize: estimating the practical *posterior utility* of a variable in a project is a *solved* problem. Mere practitioners, such as myself, routinely do this to great benefit. What we are arguing is we can't expect completely reliable domain agnostic hands-off *prior utility* estimates. So, we routinely use context dependent estimates.

Examples and Issues

Let’s make the issue concrete by working a few examples. I find it amazing how people are expected to spend hundreds of hours learning grand frameworks, yet not a few minutes working a few small examples.

Duplicate Variables

The first issue with a possible theory of intrinsic variable importance is: the issue of duplicate columns in a data store. If two columns are identical then it is literally impossible to determine from data which one is intrinsically more important than the other.

For example: if we delete one such column from a data store, we can still fit the equivalent of any model we could have before. It is hard to argue fully replaceable variables are uniquely essential. So from a model-fitting perspective duplicate columns are a fundamental issue.

Cooperating Variables

It has long been known that sometimes variables are not usable on their own, but can combine with or synergize with (perhaps in the form of what is called an interaction) to produce useful variables and models. The classic example is `xor` which we will represent as multiplying numbers chosen from the set `{-1, 1}`. Let’s work this example in `R`.

First we attach our packages.

```
library(rpart)
library(glmnet)
library(wrapr)
```

Now we show a data frame where each of our two variables appear to be individually useless in predicting the value of `y`.

```
d_joint <- data.frame(
  x1 = c(    1,      -1,     1,    -1),
  x2 = c(   -1,       1,     1,    -1),
  y  = c(FALSE,   FALSE, TRUE,  TRUE)
)

knitr::kable(d_joint)
```

x1	x2	y
1	-1	FALSE
-1	1	FALSE
1	1	TRUE
-1	-1	TRUE

We see each variable is individually of no value, as the conditional distribution of `y` is identical for all values of any one of these variables. We confirm this here.

```
table(
  y = d_joint$y,
  x1 = d_joint$x1)
```

```
##         x1
## y        -1 1
##   FALSE   1 1
##   TRUE   1 1
```

```
table(
  y = d_joint$y,
  x2 = d_joint$x2)
```

```
##        x2
## y       -1 1
##   FALSE  1 1
##   TRUE   1 1
```

We can see these variables are very useful jointly. Let’s build an interaction derived variable out of them.

```
d_joint$x1_x2 <- d_joint$x1 * d_joint$x2
```

Notice this derived variable completely determines `y`.

```
table(
  y = d_joint$y,
  x1_x2 = d_joint$x1_x2)
```

```
##        x1_x2
## y       -1 1
##   FALSE  2 0
##   TRUE   0 2
```

Dependence on Modeling Method

Given the difficulty in evaluating variables alone, let’s try evaluating them in groups. The obvious tool for combining sets of variables is a machine learning or statistical model.

That is in fact the case. We will now show an example where which variables are selected (or valued above zero) depends on the modeling method.

For this example, let’s set up some data. Again the task is to predict `y` knowing the `x`s.

```
d_test <- data.frame(
  x1 = c(3,  3, -2, -2, -2,  2,  2,  2, -3, -3),
  x2 = c(0,  0,  0,  0,  0,  1,  0,  0,  0,  0),
  x3 = c(0,  0,  0,  0,  0,  0,  1,  0,  0,  0),
  x4 = c(0,  0,  0,  0,  0,  0,  0,  1,  0,  0),
  x5 = c(0,  0,  0,  0,  0,  0,  0,  0,  1,  0),
  x6 = c(0,  0,  0,  0,  0,  0,  0,  0,  0,  1),
  y = c(rep(FALSE, 5), rep(TRUE, 5)))
```

```
knitr::kable(d_test)
```

x1	x2	x3	x4	x5	x6	y
3	0	0	0	0	0	FALSE
3	0	0	0	0	0	FALSE
-2	0	0	0	0	0	FALSE
-2	0	0	0	0	0	FALSE
-2	0	0	0	0	0	FALSE

x1	x2	x3	x4	x5	x6	y
2	1	0	0	0	0	TRUE
2	0	1	0	0	0	TRUE
2	0	0	1	0	0	TRUE
-3	0	0	0	1	0	TRUE
-3	0	0	0	0	1	TRUE

We build a larger data set that is just a number of copies of the first. This is to have enough data to run the `rpart` training procedure with default hyper parameters.

```
d_train <- do.call(
  rbind,
  rep(list(d_test), 100))
```

Regularized Generalized Linear Model

Let's use try fitting a regularized logistic regression on this data.

```
vars <- c('x1', 'x2', 'x3', 'x4', 'x5', 'x6')

rlm <- cv.glmnet(
  x = as.matrix(d_train[, vars]),
  y = d_train$y,
  family = 'binomial')
```

We get a perfect model that represents `y` as a linear function of some of the `x`s.

```
predict(
  rlm,
  newx = as.matrix(d_test[, vars]),
  type = 'response') %.>%
  knitr::kable(.)
```

1
0.0006775
0.0006775
0.0006775
0.0006775
0.0006775
0.9993217
0.9993221
0.9993225
0.9993230
0.9993235

It is easy to look at the coefficients (or "parameters") of this model.

```
rlm_coef <- coef(rlm)
rlm_coef
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                      1
## (Intercept) -7.296472
## x1            .
## x2          14.591654
## x3          14.592257
## x4          14.592898
## x5          14.593586
## x6          14.594330
```

```
rlm_vars <- rownames(rlm_coef)[abs(as.numeric(rlm_coef)) > 0]
rlm_vars <- sort(setdiff(unique(rlm_vars), "(Intercept)"))
rlm_vars
```

```
## [1] "x2" "x3" "x4" "x5" "x6"
```

Notice, the variable `x1` is not used. The liner fitting method considers it to not be of value. We will return to this in our next subsection.

Tree Based Model

We can fit the same data using a tree based model.

```
tm <- rpart(
  y ~ x1 + x2 + x3 + x4 + x5 + x6,
  data = d_train,
  method = 'class')
```

Again, we get a perfect model.

```
predict(tm, newdata = d_test, type = 'prob') %.>%
  knitr::kable(.)
```

FALSE	TRUE
1	0
1	0
1	0
1	0
1	0
0	1
0	1
0	1
0	1
0	1

Notice *only* the variable `x1` is used, even though all variables were made available.

```
tm
```

```
## n= 1000
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
```

```
##
##  1) root 1000 500 FALSE (0.5000000 0.5000000)
##    2) x1>=-2.5 800 300 FALSE (0.6250000 0.3750000)
##      4) x1< 0 300   0 FALSE (1.0000000 0.0000000) *
##      5) x1>=0 500 200 TRUE (0.4000000 0.6000000)
##       10) x1>=2.5 200   0 FALSE (1.0000000 0.0000000) *
##       11) x1< 2.5 300   0 TRUE (0.0000000 1.0000000) *
##    3) x1< -2.5 200   0 TRUE (0.0000000 1.0000000) *
```

```
tree_vars <- sort(setdiff(unique(tm$frame$var), ""))
tree_vars
```

```
## [1] "x1"
```

Comparing Modeling Methods

Note that the linear method said `x1` *is not* an important variable, yet the tree based method said `x1` *is* an important variable. This argues against `x1` having a simple intrinsic value independent of modeling method.

There *is* a intrinsic metric available via algorithmic information theory, which we will quickly discuss in the next section.

Algorithmic Information Theory

A way to eliminate context in variable evaluation: assume one can enumerate out all possible contexts. I.e. a set of variables is useful if there is *any* possible model that can use them effectively. This essentially gets you to algorithmic information theory (the essentially un-realizable Solomonoff–Kolmogorov–Chaitin complexity variation). Unfortunately this also is not actionable as you typically don’t have enough power to compute the value of variable in this way. Also even if an external system informed you of the intrinsic information content of the variables, you may not be able to use such advice, unless this oracle also supplied witnessing transformations and fit models.