Big Data BUS 41201

Week 5: Classification

Veronika Ročková

University of Chicago Booth School of Business

`http://faculty.chicagobooth.edu/veronika.rockova/`

[5] Classification

✓ K-nearest neighbors and group membership.

✓ Binary classification: from probabilities to decisions.

✓ Misclassification, sensitivity and specificity.

✓ Multinomial logistic regression: fit and probabilities.

✓ Distributed multinomial regression (DMR) and distributed computing.

Classification

Just as in linear regression, we have a set of training observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.

But now y_i are *qualitative* rather than *quantitative*, i.e. y_i is membership in a category $\{1, 2, ..., M\}$.

The classification problem:

given new $\mathbf{x}_i^{new}$ what is the class label $\hat{y}(\mathbf{x}_i^{new})$?

The quality of classifier can be assessed by its *misclassification risk*, i.e. probability of falsely classifying a new observation

$$\mathbb{P}\left(Y_{new} \neq \widehat{y}(\mathbf{x}_{new})\right)$$

This quantity is unknown but can be estimated by a proportion of wrong labels in a validation dataset. Good classifiers yield small risk.

Bayes Classifier

There is actually a theoretically optimal classifier, the *Bayes classifier*, which minimizes the misclassification risk.

The idea is to assign each observation to *the most likely class, given its predictor values*, i.e. choose the class $j \in \{1, \ldots, M\}$ for which

$$\mathbb{P}(Y = j \mid \mathbf{x})$$

is the largest.

☹ Unfortunately $\mathbb{P}(Y = j \mid \mathbf{x})$ is not known. Bayes classifier is unattainable gold standard. ☺ But! **We can estimate it!**

Classifiers

☺ There are many ways to estimate $\mathbb{P}(Y = j \mid \mathbf{x})$ from the training data.

We can go *parametric*:

- ⇝ Assume that $\mathbb{P}(Y = j \mid \mathbf{x}, \boldsymbol{\beta})$ is a specific function of unknown **parameters** $\boldsymbol{\beta}$ and learn those.
  Sounds familiar? Logistic regression...

We can go *non-parametric*:

- ⇝ We estimate $\mathbb{P}(Y = j \mid \mathbf{x})$ directly without estimating any parameters.
  K-nearest Neighbors (KNN)

Nearest Neighbors

The idea is to estimate $\mathbb{P}(Y = j \mid \mathbf{x}_{new})$ *locally* by looking at the labels of similar observations that we already saw.

K-NN: what is the most common class around x?

(1) Take K-nearest neighbors $\mathbf{x}_{i_1} \dots \mathbf{x}_{i_K}$ of $\mathbf{x}_{new}$ in the training data

‘Nearness’ is in euclidean distance: $\sqrt{\sum_{j=1}^{p} (x_{new\,j} - x_{i_k j})^2}$.

(2) Estimate

$$\mathbb{P}(Y = j \mid \mathbf{x}_{new}) = \frac{1}{K} \sum_{k=1}^{K} \mathbb{I}(y_{i_k} = j)$$

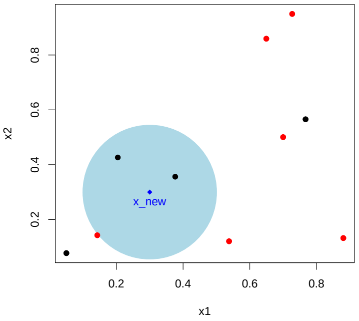
(3) Select the class with highest $\mathbb{P}(Y = j \mid \mathbf{x}_{new})$(Bayes classifier).

Since we’re calculating distances on $\mathbf{X}$, **scale Matters!**

We’ll use R’s scale function to divide each x_j by $\mathrm{sd}(x_j)$

The new units of distance are in *standard deviations*.

Nearest Neighbors



K-NN's *collaborative estimation*:

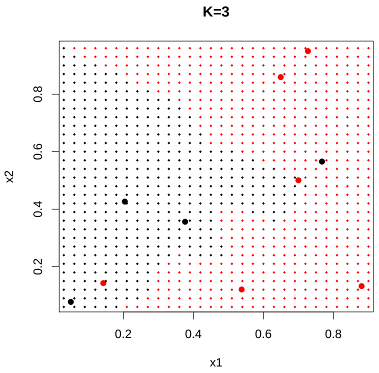
...Each neighbor votes.

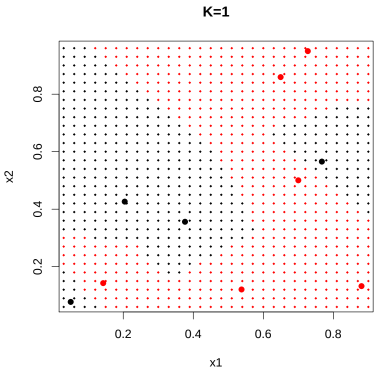
Neighborhood is by shortest distance (shown as the circle)$\rightsquigarrow$ magnifying glass

The relative vote counts provide a very crude estimate of probability.

For 3-nn, $\mathbb{P}(\text{black}) = 2/3$, but for 4-nn, it's only $1/2$.
Sensitive to neighborhood size (think about extremes: *1* or *n*).

Nearest Neighbors: Decision Boundaries





$\rightsquigarrow$ Larger K leads to higher training error (proportion of *in-sample* misclassification rate)

$\rightsquigarrow$ Smaller K leads to higher flexibility (overfitting and poor *out-of-sample* misclassification rate)

Glass Analysis

Statistics in forensic sciences

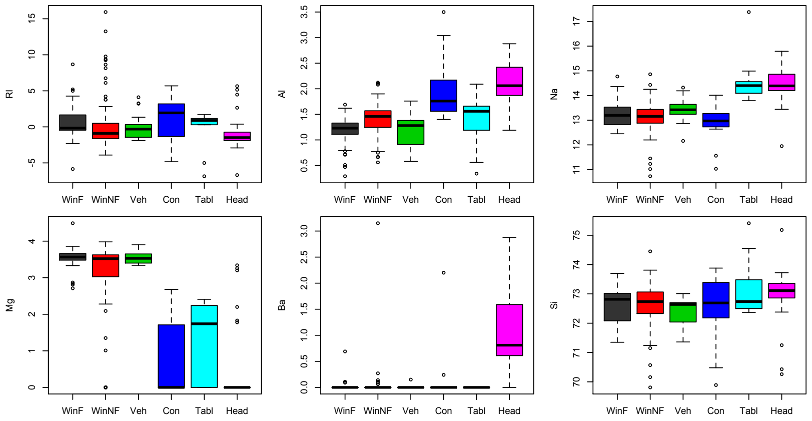
Classifying shards of glass

Refractive index, plus oxide %
Na, Mg, Al, Si, K, Ca, Ba, Fe.

6 possible glass types

`WinF:`	*float glass window*
`WinNF:`	*non-float window*
`Veh:`	*vehicle window*
`Con:`	*container (bottles)*
`Tabl:`	*tableware*
`Head:`	*vehicle headlamp*

Glass Data: characteristic by type



Some covariates are clear discriminators (Ba for headlamps, Mg for windows) while others are more subtle (Refractive Ind).

Nearest neighbors in R

Load the `class` package which includes function `knn`.

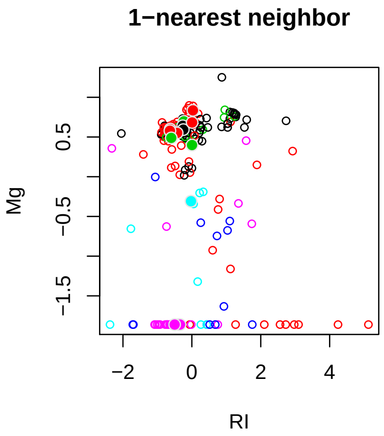
`train` and `test` are covariate matrices, `cl` holds known y's.

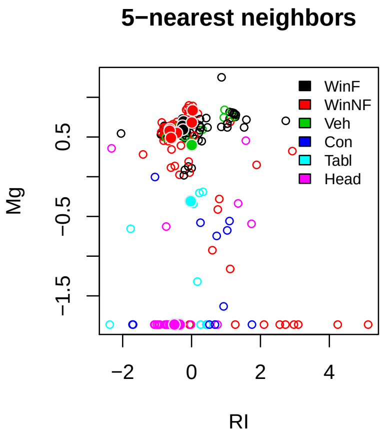
You set `k` to specify how many neighbors get to vote.

Specify `prob=TRUE` to get neighbor vote proportions.

```
knn(train=xobserved, test=xnew, cl=y, k=3)
nn1 <- knn(train=x[ti,], test=x[-ti,], cl=y[ti], k=1)
nn5 <- knn(train=x[ti,], test=x[-ti,], cl=y[ti], k=5)
data.frame(ynew,nn1,nn5)
          ynew      nn1      nn5
          WinF     WinF     WinF
           Con      Con     Head
          Tabl    WinNF    WinNF
```

KNN classification in the RI×Mg plane.





Open circles are observations and closed are predictions.

The number of neighbors matters!

KNN: Pros and Cons

☺ KNN's are simple

☺ KNN's naturally handle multiple categories ($M > 2$)

☺ KNN's will outperform linear classifiers when the decision boundary is non-linear

---

☹ Computing neighbors can be costly for large n and p.

☹ KNN's do not perform variable selection.

☹ Choosing K can be tricky. Cross-validation works, but is unstable: new data $\Rightarrow$ new K.

☹ And the classification is *very* sensitive to K.

☹ All you get is a classification, with only rough local probabilities. Without good probabilities we cannot assess uncertainty.

Binary Classification

Many decisions can be reduced to binary classification: $y_i \in \{0, 1\}$.

KNN's were an example of a *non-parametric* classification method.

A useful parametric alternative for two categories is the *logistic regression*.

Compared to KNN's

☺ *Logistic regression* yields parametric decision boundaries (linear, quadratic depending on our regression equation) $\rightsquigarrow$ it is principled but it can be flexible

☺ *Logistic regression* is a 'global' method, i.e. it uses all the training data to estimate probabilities, not just neighbors $\rightsquigarrow$ the probability estimates are more stable

☺ *Logistic regression* **can** do variable selection! (yay!)

Credit Classification

Credit scoring is a classic problem of classification.

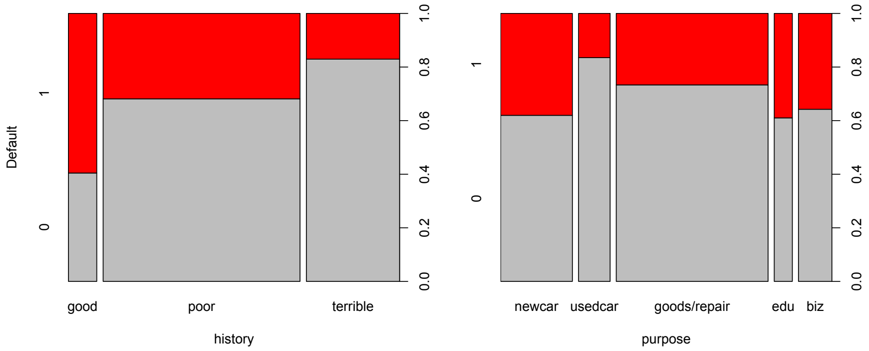
Take borrower/loan characteristics and previous defaults, use these to predict performance of potential new loans. Bond-rating is a multi-class extension of the problem.

Consider the German loan/default data in `credit.csv`.

- Borrower and loan characteristics: job, installments, etc.
- Pretty messy data, needs a bit of a clean...

Choice Sampling

A caution on retrospective sampling

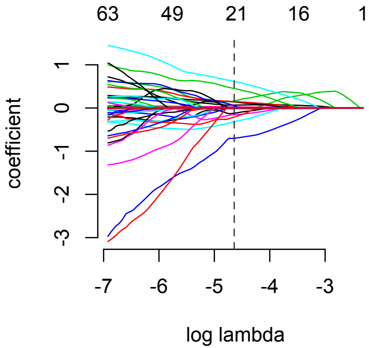


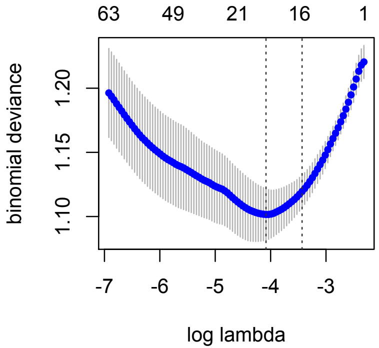
See anything strange here? Think about your data sources!

Conditioning helps here, but won't always solve everything...

German Credit Lasso

Create a numeric **x** and run lasso logistic regression.





```
> sum(coef(credscore)!=0) 13 # cv.1se
> sum(coef(credscore, s="min")!=0) 21 # cv.min
> sum(coef(credscore$gamlr)!=0) 21 # AICc
```

Decision making

There are two ways to be wrong in a binary problem.

False positive: predict $\hat{y} = 1$ when $y = 0$. (classify as defaulters when they are not)

False negative: predict $\hat{y} = 0$ when $y = 1$.(classify as non-defaulters when they in fact are)

Both mistakes are bad, but sometimes one of them can be much worse $\rightsquigarrow$ the cost *can be asymmetric*!

Logistic regression gives us an estimate $\mathbb{P}(y_{new} = 1 \,|\, \mathbf{x}_{new}, \widehat{\boldsymbol{\beta}})$.

The *Bayes decision rule* is based purely on probabilities: classify as a defaulter when $\mathbb{P}(y_{new} = 1 \,|\, \mathbf{x}_{new}, \widehat{\boldsymbol{\beta}}) > 0.5$.

However! Rather than minimizing mis-classification risk, one might like to **minimize cost***.*

Using probabilities to make decisions

To make optimal decisions, you need to take into account *probabilities* as well as *costs*.

Say that, on average, for every 1$ loaned you make 25¢ in interest if it is repayed but lose 1$ if they default.

This gives the following action-*profit* matrix

	no loan	*loan*
payer	0	0.25
defaulter	0	-1

Suppose you estimate p for the probability of default.
Expected *profit* from lending is greater than zero if

$$(1-p)\frac{1}{4} - p > 0 \quad \Leftrightarrow \quad \frac{1}{4} > \frac{5}{4}p \quad \Leftrightarrow \quad p < 1/5$$

So, from this simple matrix you should lend whenever probability of default is less than 0.2 (not 0.5!).

FP and FN Rates

Any classification cutoff (e.g., our $p = 1/5$ rule, built from an expected profit/loss analysis) has some basic properties.

False Positive Rate: # misclassified as pos / # classified pos.

False Negative Rate: # misclassified as neg / # classified neg.

In-Sample rates for our $p = 1/5$ rule:

```
## false positive rate
> sum( (pred>rule)[default==0] )/sum(pred>rule)
[1] 0.6704289
## false negative rate
> sum( (pred<rule)[default==1] )/sum(pred<rule)
[1] 0.07017544
```

For comparison, a $p = \frac{1}{2}$ cut-off gives FPR=0.27, FNR=0.28.

Sensitivity and Specificity

Two more common classification rates are

sensitivity: proportion of true $y = 1$ classified as such.

specificity: proportion of true $y = 0$ classified as such.

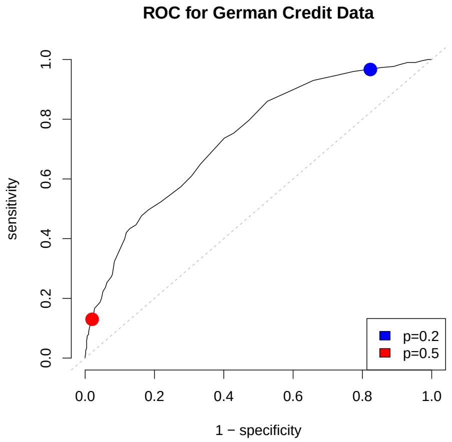
A rule is sensitive if it predicts 1 for most $y = 1$ observations, and specific if it predicts 0 for most $y = 0$ observations.

```
> mean( (pred>1/5)[default==1] )# sensitivity
[1] 0.9733333
> mean( (pred<1/5)[default==0] )# specificity
[1] 0.1514286
```

Contrast with FP + FN, where you are dividing by *total classified a certian way*. Here you divide by *true totals*.

Our rule is sensitive, not specific, because we lose more with defaults than we gain from a payer.

The ROC curve: sensitivity vs 1-specificity



From signal processing: Receiver Operating Characteristic.

A tight fit has the curve forced into the top-left corner.

◇ Discriminant Analysis

Discriminant Analysis (DA) assumes classification probabilities

$$\mathbb{P}(Y = j \mid \mathbf{x}) = \frac{p_j \pi_j(\mathbf{x})}{\sum_{k=1}^{M} p_k \, \pi_k(\mathbf{x})}$$

where $\pi_j(\mathbf{x})$ is a **model** for the j^{th} category and p_j is a prior class probability

Two useful choices: $\pi_j(\cdot)$ is Gaussian

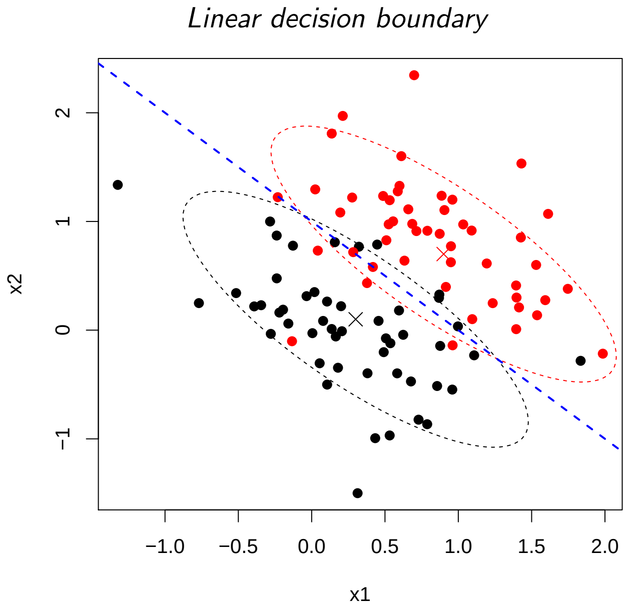
(1) LDA: *mean* μ_j and *common variance* Σ.

$\rightsquigarrow$ Linear decision boundary

(2) QDA: *mean* μ_j and *group-specific variance* Σ_j.

$\rightsquigarrow$ Quadratic decision boundary

◇ Linear Discriminant Analysis: K=2



Multinomial Logistic Regression

☺ *Probabilities are the basis for good cost-benefit classification.*

Similarly as in logistic regression (M=2), can get class probabilities $\mathbb{P}(Y = j \,|\, \mathbf{x})$ for more than two categories (M>2)?

Yes! $\rightsquigarrow$ *Multinomial Logistic Regression*

We need M models (for *each category*)

$$\begin{aligned}
&\mathbb{P}(Y = 1 \,|\, \mathbf{x}) \propto f(\mathbf{x}'\boldsymbol{\beta}_1) \\
&\mathbb{P}(Y = 2 \,|\, \mathbf{x} \propto) f(\mathbf{x}'\boldsymbol{\beta}_2) \\
&\dots \\
&\mathbb{P}(Y = M \,|\, \mathbf{x}) \propto f(\mathbf{x}'\boldsymbol{\beta}_M).
\end{aligned}$$

$\rightsquigarrow$ We need to find regression coefficients $\boldsymbol{\beta}_k$ for *each* class.
$\rightsquigarrow$ We need to make sure that $\sum_{j=1}^{M} \mathbb{P}(Y = j \,|\, \mathbf{x}) = 1$

Multinomial Logistic Regression

Extend logistic regression via the multinomial logit:

$$\mathbb{P}(Y_i = k \mid \mathbf{x}_i) = p_{ik} = \frac{e^{\mathbf{x}_i'\boldsymbol{\beta}_k}}{\sum_{j=1}^{M} e^{\mathbf{x}_i'\boldsymbol{\beta}_j}}$$

Note separate coefficients for each class: $\boldsymbol{\beta}_k$.

Denote by k_i the class of i^{th} observation y_i. Then, the likelihood is

$$LHD(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_M) \propto \prod_{i=1}^{n} p_{ik_i}$$

and the deviance is

$$\mathrm{Dev}(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_M) \propto -2 \sum_{i=1}^{n} \log p_{ik_i}.$$

Multinomial Logistic Regression

☺ Once we have a model, we can do *variable selection* in each of the M regressions.

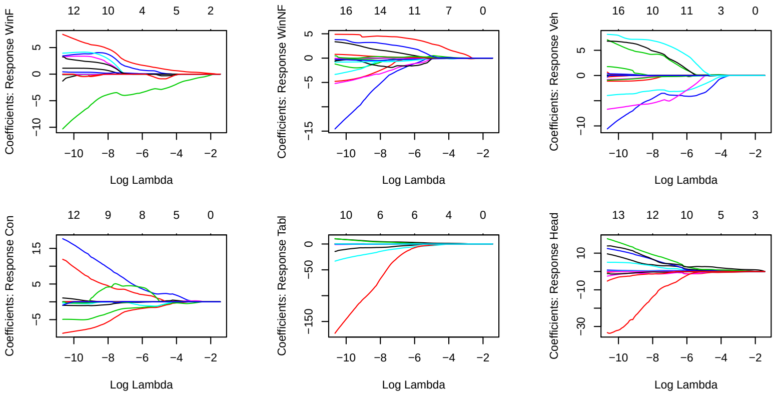
We can use the LASSO penalty: *penalized deviance minimization.*

$$\min \left\{ -\frac{2}{n} \sum_{i=1}^{n} \log p_{ik_i} + \lambda \sum_{k=1}^{M} \sum_{j=1}^{p} |\beta_{kj}| \right\}$$

We can also have λ_k: different penalty for each class.

We can find out which predictors in $\mathbf{x}_i$ are relevant discriminators of each of the M classes.

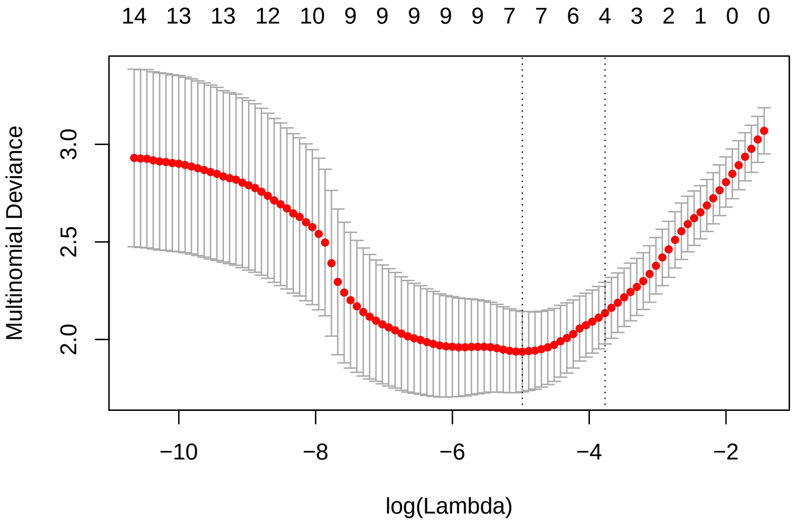
Fit the model in `glmnet` with `family="multinomial"`.



A separate path plot for every class.

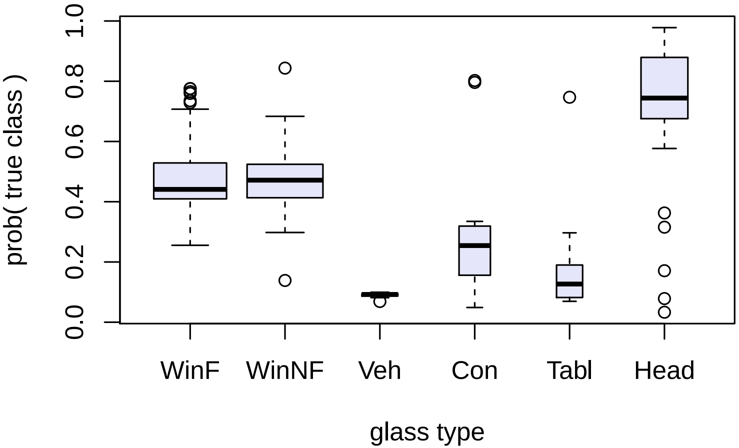
See `glass.R` for coefficients, prediction, and other details.

We can do OOS experiments on *multinomial deviance*.



And use this to choose λ (one shared for all classes here).

The 'fit plot' for multinomials: $\hat{p}_{ik_i}$, prob of true class, on k_i.



`Veh`, `Con`, `Tabls` have low fitted probabilities, but they are generally more rare in this sample (width of box is $\propto$ count).

MN classification via decision costs

Suppose a simple cost matrix has

	`WinF`	`WinNF`	`Veh`	`Con`	`Tabl`	`Head`
$\hat{k} =$`Head`	9	9	9	9	9	0
$\hat{k} \neq$`Head`	0	0	0	0	0	1

e.g. a court case where `Head` is evidence for the prosecution (innocent until proven guilty, and such).

Then expected cost of $\hat{k} \neq$ `Head` is greater than $\hat{k} =$ `Head` if

$$p_{\texttt{head}} > 9(1 - p_{\texttt{head}}) \quad \Leftrightarrow \quad p_{\texttt{head}} > 0.9$$

If you don't have asymmetric costs,
just use a maximum probability rule: $\hat{k} = \mathrm{argmax}_k \ \hat{p}_k$.
You can get this in R with `apply(probs,1,which.max)`.

Interpreting the MN logit

We're estimating a function that sums to one across classes. But now there are K categories, instead of just two.

The log-odds interpretation now compares between classes:

$$\log\left(\frac{p_a}{p_b}\right) = \log\left(\frac{e^{\mathbf{x}'\boldsymbol{\beta}_a}}{e^{\mathbf{x}'\boldsymbol{\beta}_b}}\right) = \mathbf{x}[\boldsymbol{\beta}_a - \boldsymbol{\beta}_b].$$

For example, with a one unit increase in `Mg`:

```
# odds of non-float over float drop by 33%
exp(B["Mg","WinNF"]-B["Mg","WinF"])
 0.6633846
# odds of non-float over Con increase by 67%
exp(B["Mg","WinNF"]-B["Mg","Con"])
 1.675311
```

An alternative version of MN logit

You might have noticed: *multinomial regression can be slow* ☹...

This is because everything needs to be done K times!

And each p_{ik} depends on $\boldsymbol{\beta}_k$ *as well as all* the other $\boldsymbol{\beta}_j$'s:
$p_{ik} = e^{\mathbf{x}'\boldsymbol{\beta}_k} / \sum_j e^{\mathbf{x}'\boldsymbol{\beta}_j}$.

Let y_{ik} be 0/1 random variable where $y_{ik} = 1$ when $Y_i = k$.

It turns out that multinomial logistic regression is *very similar* to

$$\mathbb{P}(Y_i = k \,|\, \mathbf{x}_i) = \mathbb{E}[y_{ik}|\mathbf{x}_i] = \exp(\mathbf{x}_i'\boldsymbol{\beta}_k).$$

That is, K *independent* log regressions for each class k.

The full regression is $y_{ik} \sim \mathrm{Poisson}(\exp[\mathbf{x}_i'\boldsymbol{\beta}_k])$, which is the glm for 'count response'. Deviance is $\propto \sum_{i=1}^{n} \exp(\mathbf{x}_i'\boldsymbol{\beta}_k) - y_i(\mathbf{x}_i'\boldsymbol{\beta}_k)$.

Distributed Multinomial Regression

Since each $y_{ik} \sim \mathrm{Poisson}(e^{\mathbf{x}_i'\boldsymbol{\beta}_k})$ regression is independent, wouldn't it be faster to do these all at the same time? Yes!

`dmr` function in the `distrom` library does just this.
In particular, `dmr` minimizes

$$\sum_{i=1}^{n}[\exp(\mathbf{x}_i'\boldsymbol{\beta}_k) - y_{ik}(\mathbf{x}_i'\boldsymbol{\beta}_k)] + \lambda_k \sum_j |\beta_{jk}|$$

along a path of λ_k *in parallel for every response class* k.
We then use AICc to get a different $\hat{\lambda}_k$ for each k.

You can use $\hat{\boldsymbol{\beta}}_1 \ldots \hat{\boldsymbol{\beta}}_K$ as if they are for a multinomial logit.
The intercepts differ from `glmnet`'s, but that's a wash anyways.

DMR

`dmr` is a faster way to fit multinomial logit in parallel.
It's based on `gamlr`, so the syntax will be familiar.

```
dmr(cl, covars, counts, ...)
```

- `covars` is **x**.
- `counts` is **y**. Can be a factor variable.
- `...` are arguments to `gamlr`.
- `cl` is a parallel socket cluster.

It takes `coef` and `predict` as you're used to.

The returned `dmr` object is actually a list of K gamlr objects, and you can call plot, etc, on each of these too if you want.

“ **to compute in parallel** ”
do many calculations at the same time on different processors.

Supercomputers have long used parallelism for massive speed.
Since 2000’s, it has become standard to have many processor ‘cores’ on consumer machines. Even my phone has 4.

You can take advantage of this without even knowing.

- Your OS runs applications on different cores.
- Videos run on processing units with 1000s of tiny cores.

And numeric software can be set up to use multiple processors.
e.g., if you build R ‘from source’, you can set this up.

Parallel Computing in R

R's `parallel` library lets you take advantage of many cores.

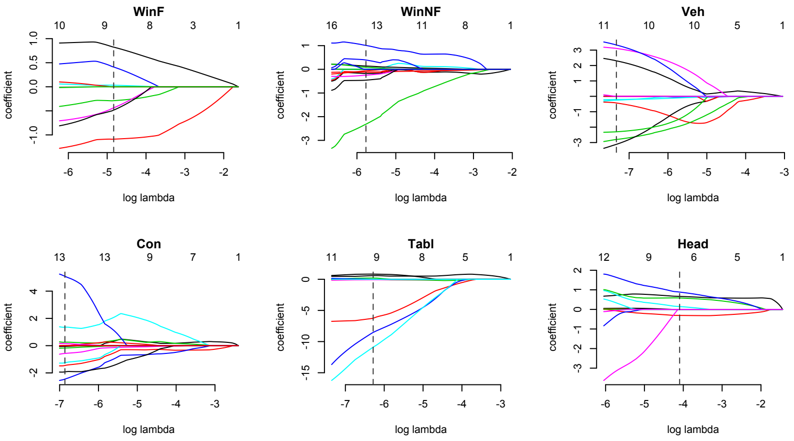
It works by organizing *clusters* of processors.

To get a cluster of cores do `cl <- makeCluster(4)`
You can do `detectCores()` to see how many you have.
If you're on a unix machine (mac/linux), you can ask for `makeCluster(4,type="FORK")` and it will often be faster.

After building `cl`, just pass it to `dmr` and you're off to the [parallel] races. Use `stopCluster(cl)` when you're done.

Note: this requires your computer is setup for parallelization. This *should* be true, but if not you can run `dmr` with `cl=NULL`.

DMR for glass data



The vertical lines show AICc selection: note it moves!

Note that `glmnet` `cv.min` rule chose $\log\hat{\lambda} \approx 5$.