Principles of Machine Learning

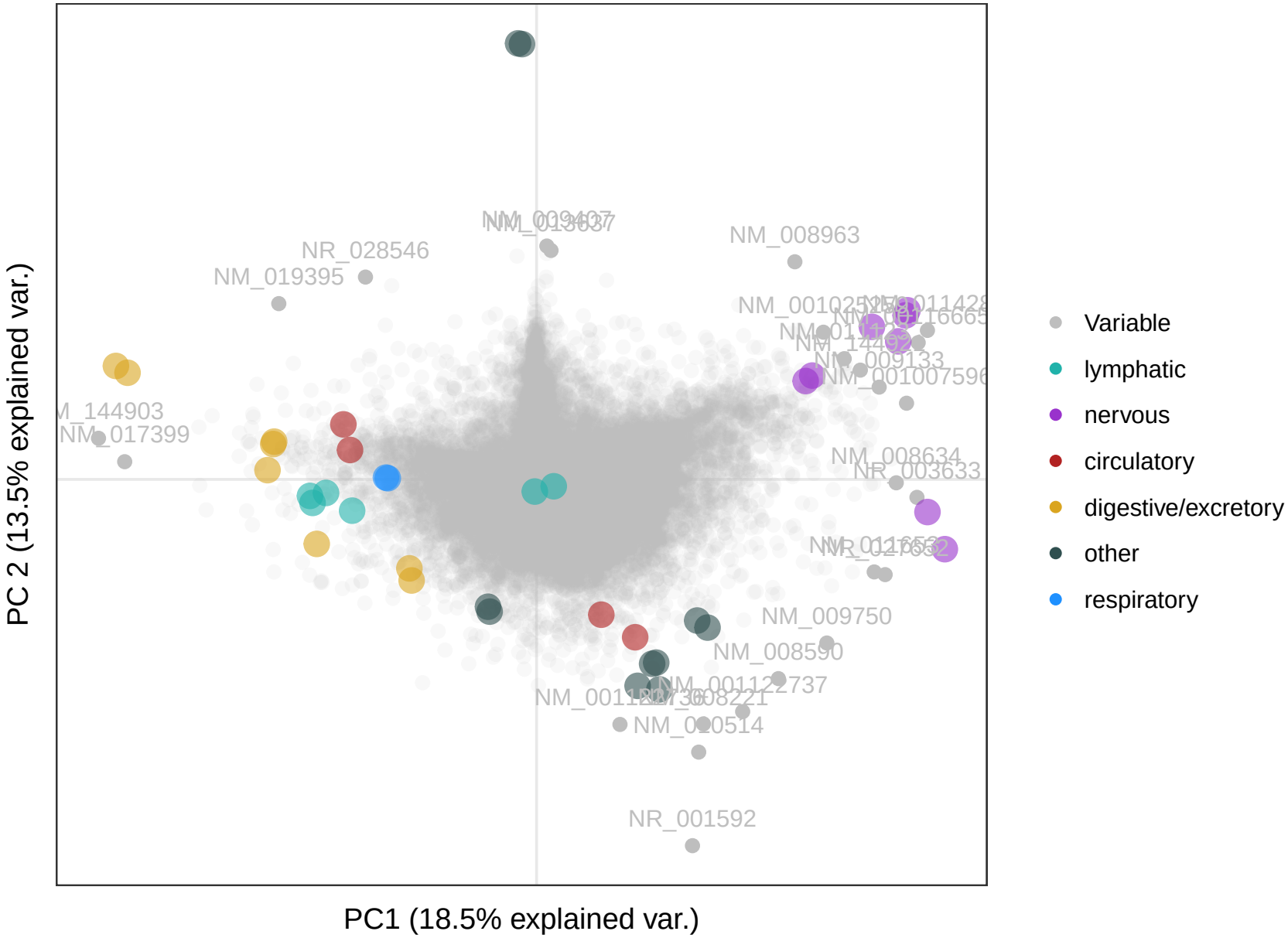
Day 2

Dennis Wylie, UT Bioinformatics Consulting Group

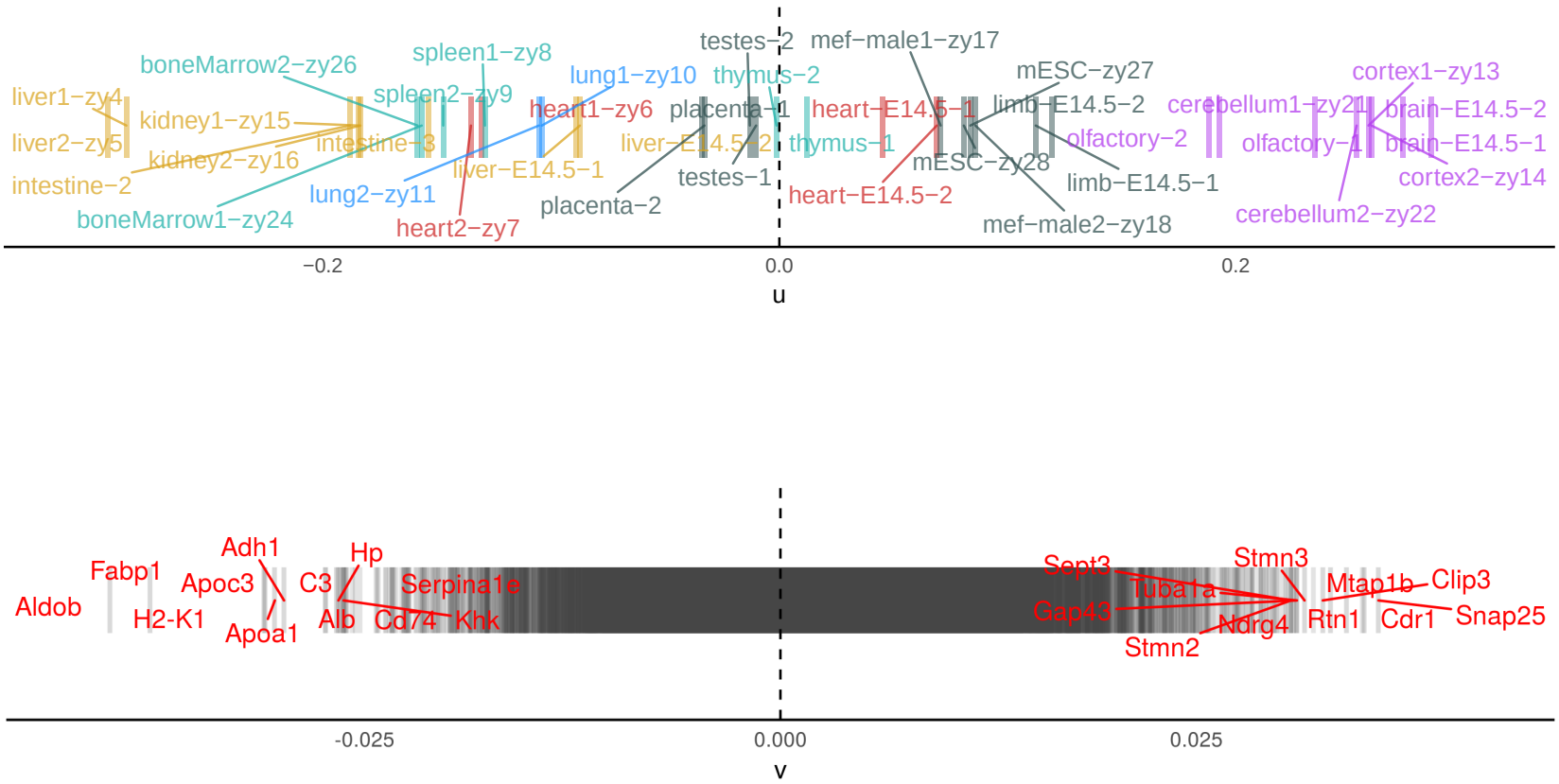
May 29, 2019

Outline

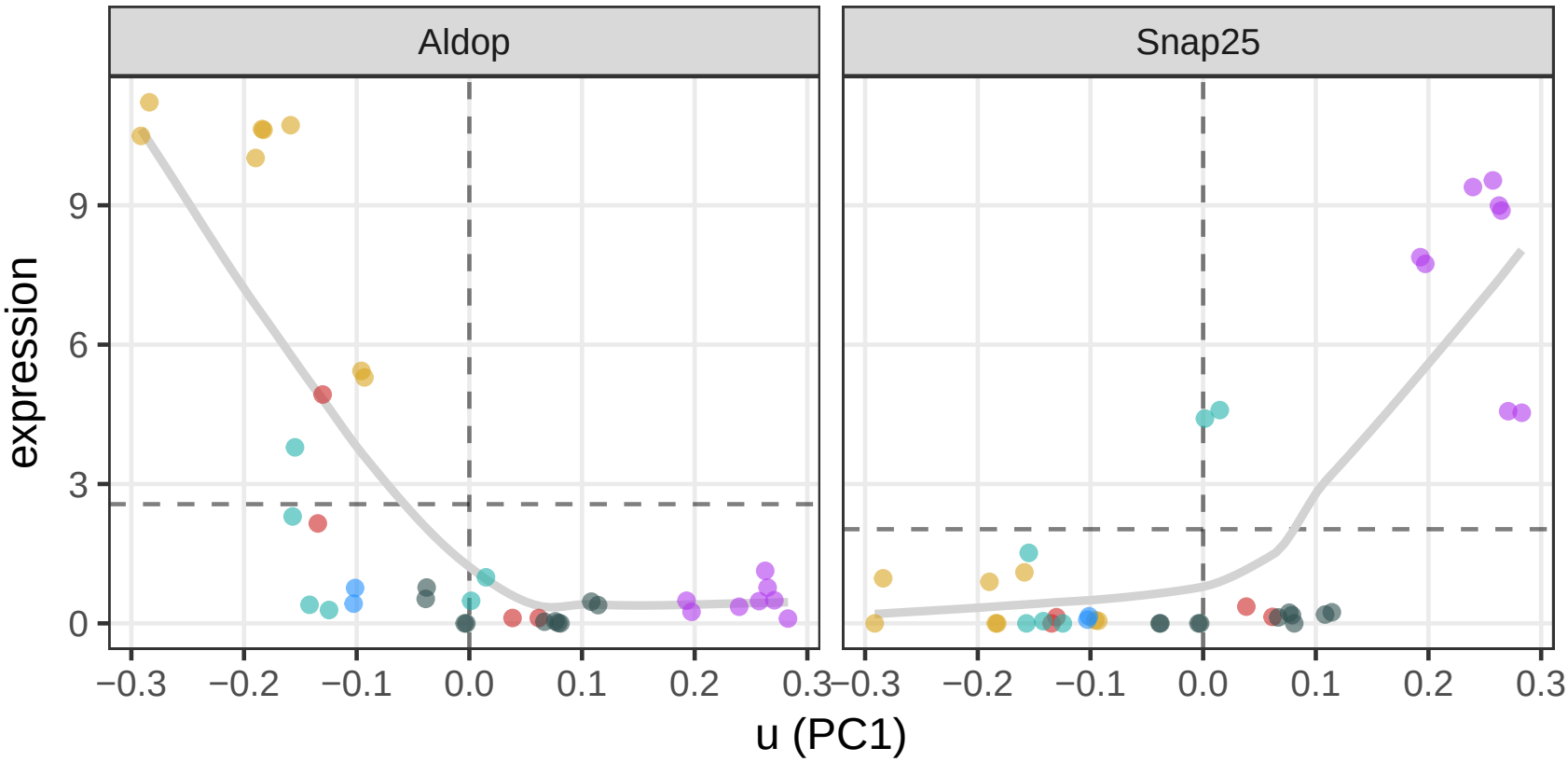
PCA



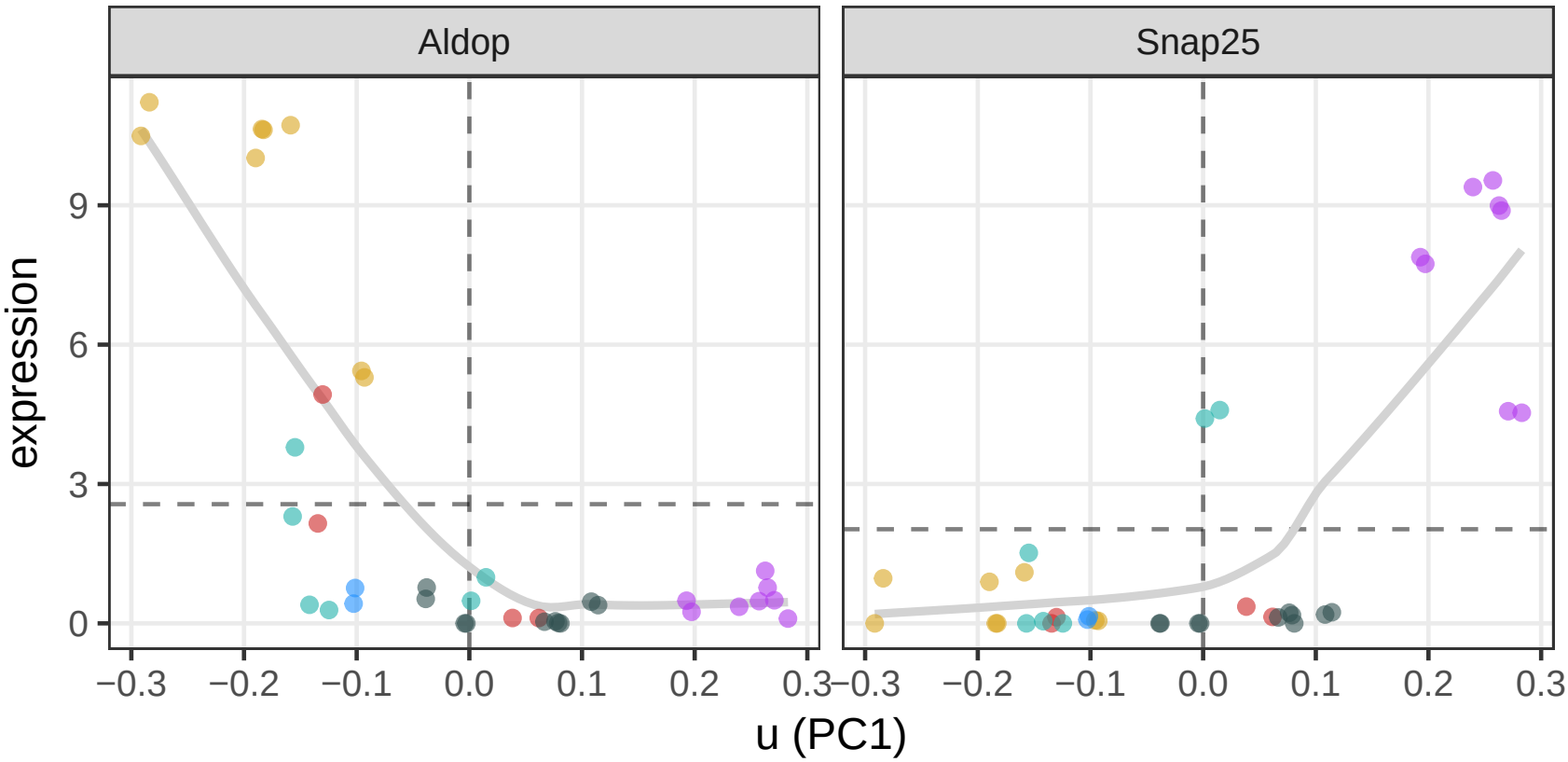
Single principal component



Top PC1 genes



Top PC1 genes



Expression values of genes with highest PC1 *loadings*
(=entries in vector **v** here)

will tend to correlate with sample PC1 loadings
(=entries in vector **u**).

Single principal component model

Given a gene expression matrix (x_{ig})

- with samples i in rows and genes g in columns

PC1 model for random variable X_{ig} is:

$$X_{ig} - \mathbb{E}[X_{\cdot g}] = u_i d\, v_g + \epsilon_{ig}$$

Single principal component model

Given a gene expression matrix (x_{ig})

- with samples i in rows and genes g in columns

PC1 model for random variable X_{ig} is:

$$X_{ig} - \mathbb{E}[X_{\cdot g}] = u_i d\, v_g + \epsilon_{ig}$$

assuming that $\epsilon_{ig} \sim \mathcal{N}(0, \sigma^2)$ and constraining

$$\sum_i u_i^2 = \sum_g v_g^2 = 1$$

Single principal component model

Given a gene expression matrix (x_{ig})

- with samples i in rows and genes g in columns

PC1 model for random variable X_{ig} is:

$$X_{ig} - \mathbb{E}[X_{\cdot g}] = u_i d\, v_g + \epsilon_{ig}$$

assuming that $\epsilon_{ig} \sim \mathcal{N}(0, \sigma^2)$ and constraining

$$\sum_i u_i^2 = \sum_g v_g^2 = 1$$

Model says that:

- gene g with $v_g > 0$ expected to have higher expression
- in sample i than in sample j if $u_i > u_j$
  - by an amount $(u_i - u_j)v_g$.

Single principal component model

Given a gene expression matrix (x_{ig})

- with samples i in rows and genes g in columns

PC1 model for random variable X_{ig} is:

$$X_{ig} - \mathbb{E}[X_{\cdot g}] = u_i d\, v_g + \epsilon_{ig}$$

assuming that $\epsilon_{ig} \sim \mathcal{N}(0, \sigma^2)$ and constraining

$$\sum_i u_i^2 = \sum_g v_g^2 = 1$$

That is: PC1 model assigns

- to each sample i a score u_i
- and to each gene g a score v_g such that
  - high scoring genes highly expressed in high scoring samples

Single principal component model

Given a gene expression matrix (x_{ig})

- with samples i in rows and genes g in columns

PC1 model for random variable X_{ig} is:

$$X_{ig} - \mathbb{E}[X_{\cdot g}] = u_i d\, v_g + \epsilon_{ig}$$

assuming that $\epsilon_{ig} \sim \mathcal{N}(0, \sigma^2)$ and constraining

$$\sum_i u_i^2 = \sum_g v_g^2 = 1$$

$\mathbb{E}[X_{\cdot g}]$ is mean expression of gene g in full population.

When doing PCA:

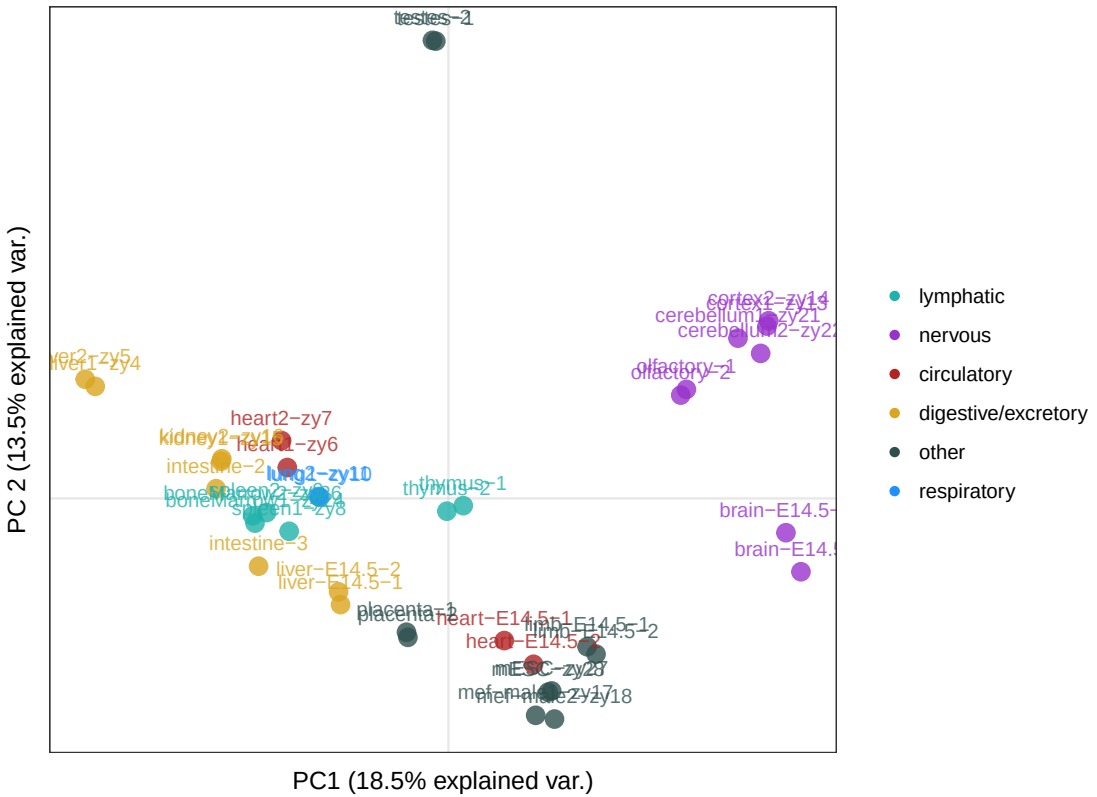
- use estimated $\tilde{x}_{ig} = x_{ig} - \frac{1}{n}\sum_i x_{ig}$ in place of $x_{ig} - \mathbb{E}[X_{\cdot g}]$
- principled approach should account for this estimation...

PCA plot

It seems unreasonable to expect a single linear pattern to explain everything in our data, though. . . let's try two:

$$\tilde{x}_{ig} = (u_{i1} d_{11} v_{g1} + u_{i2} d_{22} v_{g2}) + \epsilon_{ig}$$

and again select u_{iq}, d_{qq}, v_{gq}, and ϵ_{ig} so as to minimize $\sum_{i,g} \epsilon_{ig}^2$



PCA by SVD

Can continue this process to obtain n principal components (assuming $n \leq p$).

These can be calculated via the singular value decomposition (SVD) of $\underline{\tilde{\mathbf{X}}}$ (note $\underline{\tilde{\mathbf{X}}}$ is data matrix, not random variable)

$$\underline{\tilde{\mathbf{X}}} = \underline{\mathbf{U}}\underline{\mathbf{D}}\underline{\mathbf{V}}^T$$

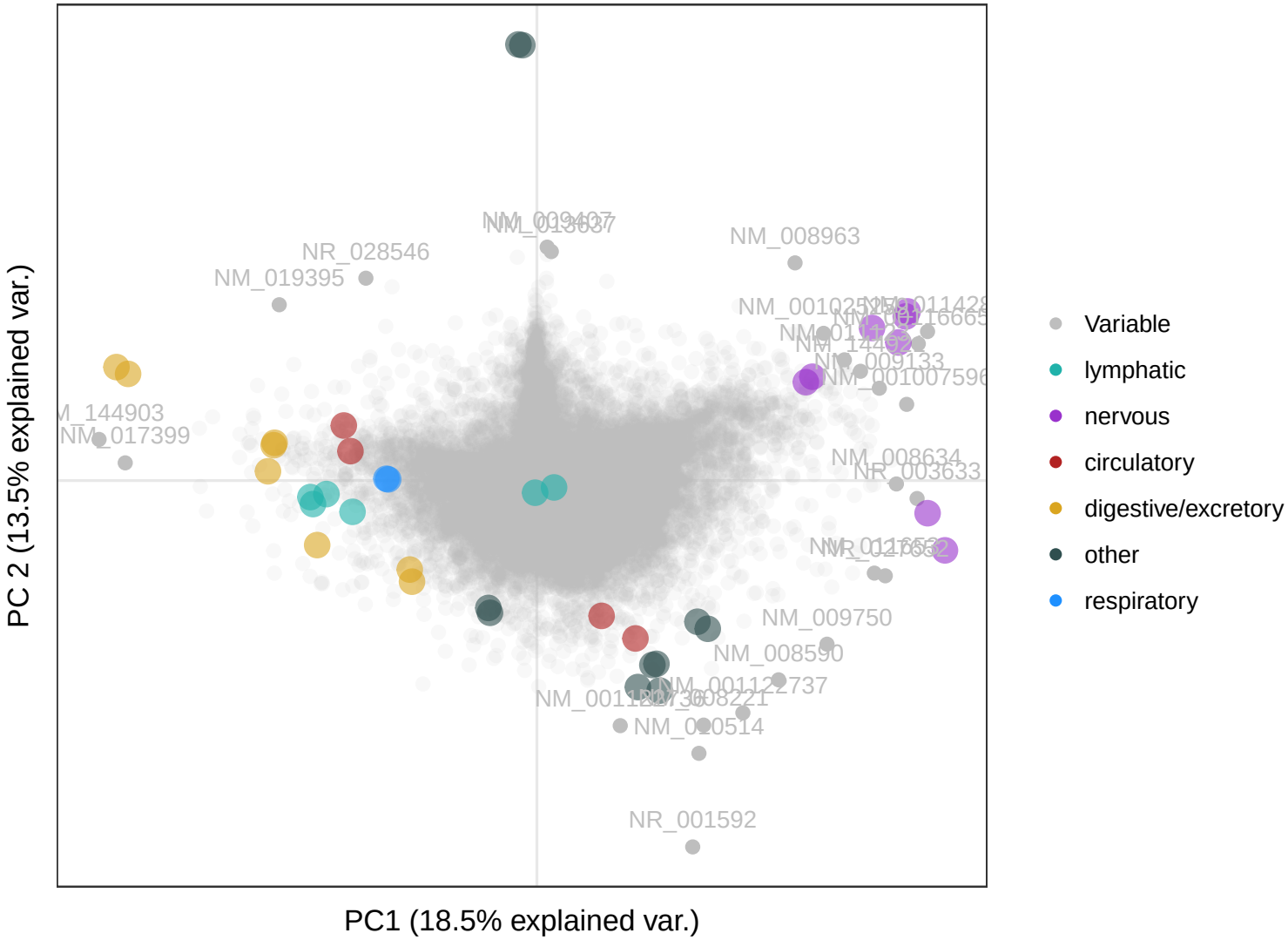
where $\underline{\mathbf{U}}$ and $\underline{\mathbf{V}}$ are orthogonal matrices and $\underline{\mathbf{D}}$ is an $n \times n$ diagonal matrix with the diagonal sorted in descending order.

In terms of the components $\tilde{x}_{ig}$:

$$\tilde{x}_{ig} = \sum_q u_{iq} d_{qq} v_{gq}$$

PCA biplot

Useful to also plot the points (v_{g1}, v_{g2}) for those genes g with large contributions to the first two principal components:



PCA and eigendecomposition

The SVD of $\underline{\tilde{\mathbf{X}}}$ also has the following interesting properties:

- $\underline{\mathbf{D}}$ the diagonal of $\underline{\mathbf{D}}$ contains the square roots of the eigenvalues of $\underline{\tilde{\mathbf{X}}\tilde{\mathbf{X}}}^T$ (and thus also of $\underline{\tilde{\mathbf{X}}}^T\underline{\tilde{\mathbf{X}}}$).
- $\underline{\mathbf{U}}$ the columns of $\underline{\mathbf{U}}$ are the eigenvectors of the matrix $\underline{\tilde{\mathbf{X}}\tilde{\mathbf{X}}}^T$, so that $\underline{\tilde{\mathbf{X}}\tilde{\mathbf{X}}}^T\underline{\mathbf{U}} = \underline{\mathbf{U}\mathbf{D}^2}$.
- $\underline{\mathbf{V}}$ the columns of $\underline{\mathbf{V}}$ are the n eigenvectors of the matrix $\underline{\tilde{\mathbf{X}}}^T\underline{\tilde{\mathbf{X}}}$ with non-zero eigenvalues, so that $\underline{\tilde{\mathbf{X}}}^T\underline{\tilde{\mathbf{X}}\mathbf{V}} = \underline{\mathbf{V}\mathbf{D}^2}$.

Note that:

- $\frac{1}{n-1}\underline{\tilde{\mathbf{X}}}^T\underline{\tilde{\mathbf{X}}}$ is the estimated gene-gene covariance matrix, and
- $\frac{1}{p-1}\underline{\tilde{\mathbf{X}}\tilde{\mathbf{X}}}^T$ is the estimated sample-sample covariance matrix.

So SVD relates the eigendecompositions of the estimated gene- and sample-covariance matrices.

Probabilistic PCA and factor analysis

PCA using k PCs can be derived from small σ limit of model:

$$\mathbb{P}\left(\tilde{\mathbf{X}} = \tilde{\mathbf{x}} \mid M, [w_{ga}], (z_a)\right) = \frac{1}{\sqrt{2\pi\sigma^2}^p} \exp\left[-\frac{1}{2\sigma^2}\sum_{g=1}^{p}\left(\tilde{x}_g - \sum_{a=1}^{k} w_{ga} z_a\right)^2\right]$$

where the k-vector $\mathbf{z} \sim \mathcal{N}(0, I)$ is a *latent* (unobserved) variable.

Probabilistic PCA and factor analysis

PCA using k PCs can be derived from small σ limit of model:

$$\mathbb{P}\left(\tilde{\mathbf{X}} = \tilde{\mathbf{x}} \mid M, [w_{ga}], (z_a)\right) = \frac{1}{\sqrt{2\pi\sigma^2}^p} \exp\left[-\frac{1}{2\sigma^2}\sum_{g=1}^{p}\left(\tilde{x}_g - \sum_{a=1}^{k} w_{ga} z_a\right)^2\right]$$

where the k-vector $\mathbf{z} \sim \mathcal{N}(0, I)$ is a *latent* (unobserved) variable.

If instead of assuming that $\widetilde{X}_g - \sum_a w_{ga} Z_a \sim \mathcal{N}(0, \sigma^2)$ for some common fixed infinitesimal σ we allow:

$$E_g = \widetilde{X}_g - \sum_{a=1}^{k} w_{ga} Z_a \sim \mathcal{N}(0, \psi_g^2)$$

- where ψ_g is no longer assumed small
- but we retain assumption of between-gene independence,

we derive the standard *factor analysis* model (Roweis & Ghahramani (1999)).

t-SNE

Instead of Euclidean distances ($\propto$ sums of squares), *stochastic neighbor embedding* attempts to preserve

- similarities scores analagous to probabilities
- while dramatically reducing dimensionality.

The similarity scores in full-dimensional space are derived from

$$p_{j|i} = \frac{\exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_k||^2}{2\sigma_i^2}\right)}$$

which is then symmetrized to a joint probability analogue

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

t-SNE

Instead of Euclidean distances ($\propto$ sums of squares), *stochastic neighbor embedding* attempts to preserve

- similarities scores analagous to probabilities
- while dramatically reducing dimensionality.

For t-SNE, the (symmetric) similarity scores in low-dimensional space are

$$q_{ij} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}}$$

How best to preserve full-dimensional similarities?

Generally cannot perfectly match q_{ij} to p_{ij}.

Instead minimize Kullback-Leibler divergence between the two:

$$D_{\mathrm{KL}}(\underline{\mathbf{P}} \,||\, \underline{\mathbf{Q}}) = \sum_{i,j} p_{ij} \left(\log p_{ij} - \log q_{ij}\right)$$

D_{KL} is a key concept in information theory

- related to efficiency of encoding information about events
  - the true probabilities of which are $\underline{\mathbf{P}}$
  - using a code built on (incorrect) probabilities $\underline{\mathbf{Q}}$.

How to set σ_i

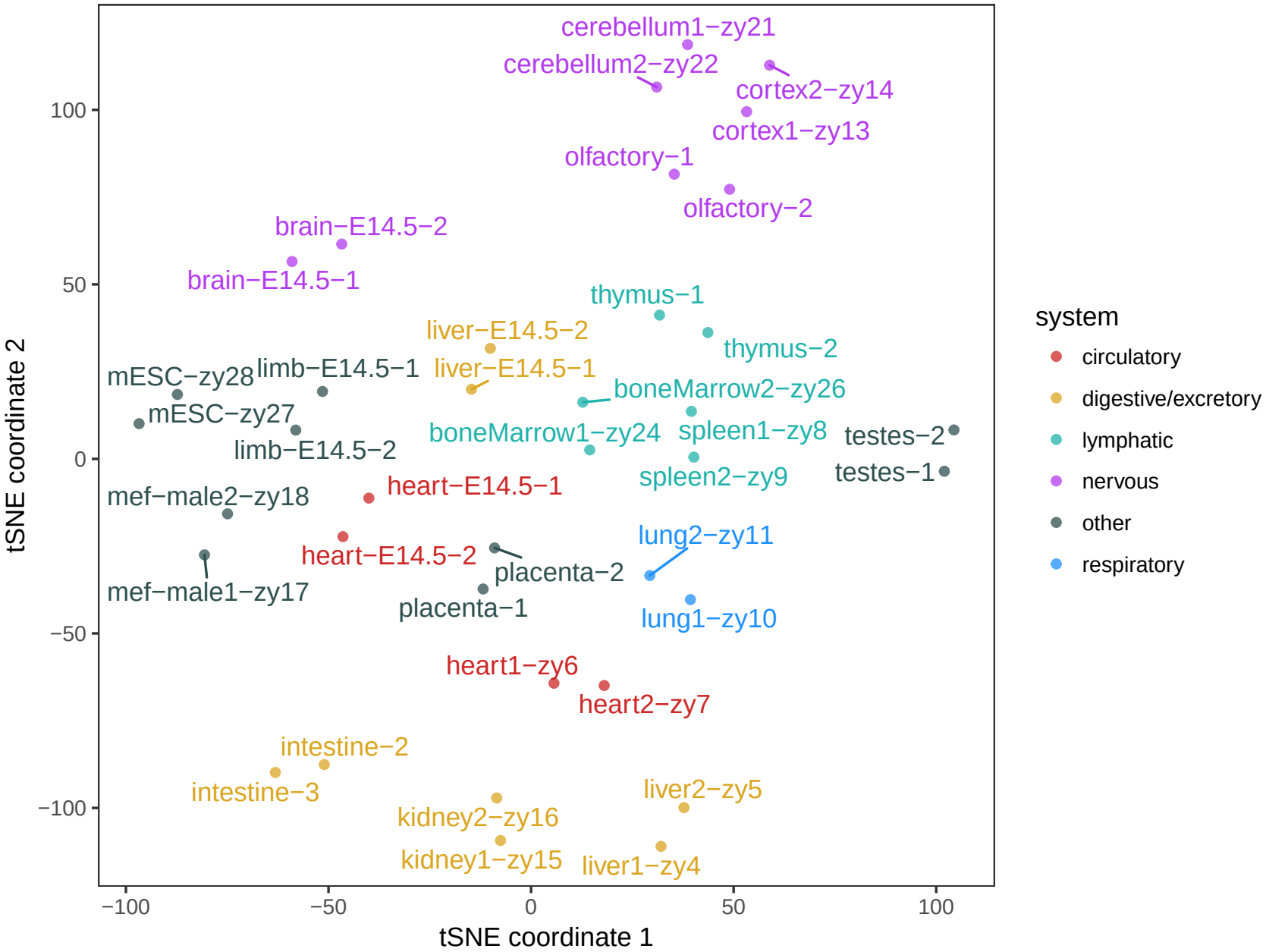
Idea here is to fix constant *perplexity* value for all samples i

- by choosing σ_i such that, for each sample i

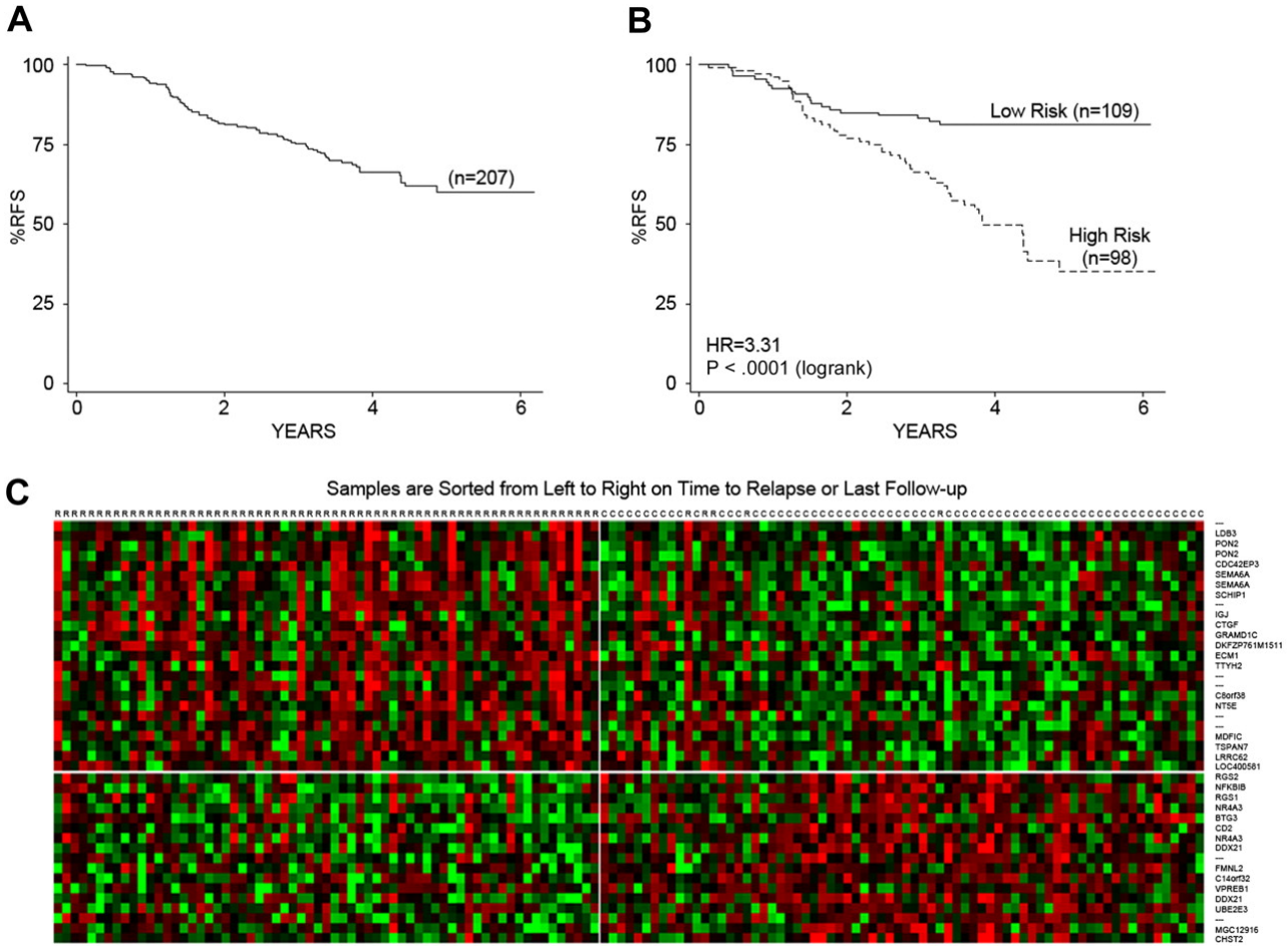
$$\prod_{j \neq i} p_{j|i}(\sigma_i)^{p_{j|i}(\sigma_i)} = \text{desired perplexity value}$$

Intuition:
Require each sample to have similar # of "stochastic neigbors"

t-SNE


cerebellum1−zy21
cerebellum2−zy22
cortex2−zy14
cortex1−zy13
olfactory−1
olfactory−2
brain−E14.5−2
brain−E14.5−1
thymus−1
thymus−2
liver−E14.5−2
liver−E14.5−1
mESC−zy28
limb−E14.5−1
boneMarrow2−zy26
mESC−zy27
boneMarrow1−zy24
spleen1−zy8
testes−2
limb−E14.5−2
spleen2−zy9
testes−1
heart−E14.5−1
mef−male2−zy18
lung2−zy11
heart−E14.5−2
placenta−2
mef−male1−zy17
placenta−1
lung1−zy10
heart1−zy6
heart2−zy7
intestine−2
intestine−3
liver2−zy5
kidney2−zy16
kidney1−zy15
liver1−zy4
100
50
0
−50
−100
tSNE coordinate 2
−100
−50
0
50
100
tSNE coordinate 1
system
circulatory
digestive/excretory
lymphatic
nervous
other
respiratory

What is a classifier?



A 38-gene expression classifier predictive of relapse-free survival (RFS) could distinguish 2 groups with differing relapse risks: low (4-year RFS, 81%, n = 109) versus high (4-year RFS, 50%, n = 98; P < .001).

Taken from Kang *et al.* (2010).

Classification by gene expression

Goal:
Given sample i, assign class label y_i

- using measured gene expression levels x_{ig}.

Vector $\mathbf{x}_i$ contains all gene measurements x_{ig} for sample i.

For simplicity, consider only two-class problems $y_i \in \{0, 1\}$.

Define random variables $\mathbf{X}$ and Y

- of which $\mathbf{x}_i$ and y_i will be regarded as particular realizations.

Probalistic models yield $\mathbb{P}(Y = y \mid \mathbf{X} = \mathbf{x})$.

Training and test sets

Select modeling strategy M

- apply algorithm to fit parameters $\boldsymbol{\theta}$
- using a set S_{train} of samples

Training and test sets

Select modeling strategy M

- apply algorithm to fit parameters $\boldsymbol{\theta}$
- using a set S_{train} of samples
- possibly by maximizing $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M,\boldsymbol{\theta}}(Y = y_i \mid \mathbf{X} = \mathbf{x}_i)$

Training and test sets

Select modeling strategy M

- apply algorithm to fit parameters $\boldsymbol{\theta}$
- using a set S_{train} of samples
- possibly by maximizing $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M,\boldsymbol{\theta}}(Y = y_i \mid \mathbf{X} = \mathbf{x}_i)$
- or maybe it should be $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M}(\boldsymbol{\theta} \mid Y = y_i, \mathbf{X} = \mathbf{x}_i)$?

Training and test sets

Select modeling strategy M

- apply algorithm to fit parameters $\boldsymbol{\theta}$
- using a set S_{train} of samples
- possibly by maximizing $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M,\boldsymbol{\theta}}(Y = y_i \mid \mathbf{X} = \mathbf{x}_i)$
- or maybe it should be $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M}(\boldsymbol{\theta} \mid Y = y_i, \mathbf{X} = \mathbf{x}_i)$?

Machine Learning point of view on classification:
fit model should accurately classify samples $j \notin S_{\text{train}}$

- whose true classifications y_j may not already be known.

Training and test sets

Select modeling strategy M

- apply algorithm to fit parameters $\boldsymbol{\theta}$
- using a set S_{train} of samples
- possibly by maximizing $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M,\boldsymbol{\theta}}(Y = y_i \mid \mathbf{X} = \mathbf{x}_i)$
- or maybe it should be $\prod_{i \in S_{\text{train}}} \mathbb{P}_{M}(\boldsymbol{\theta} \mid Y = y_i, \mathbf{X} = \mathbf{x}_i)$?

Machine Learning point of view on classification:
fit model should accurately classify samples $j \notin S_{\text{train}}$

- whose true classifications y_j may not already be known.

Generally $(M, \boldsymbol{\theta})$ less accurate for $j \notin S_{\text{train}}$ than for $i \in S_{\text{train}}$.

Thus useful to apply $(M, \boldsymbol{\theta})$ to $j \in S_{\text{test}}$ where $S_{\text{test}} \cap S_{\text{train}} = \emptyset$

- but where the $\{y_j \mid j \in S_{\text{test}}\}$ are still known.

k-nearest-neighbors (knn)

Perhaps simplest approach to classification:

k-nearest neighbors

Given feature vector $\mathbf{x}$ (e.g., expression levels x_g)

- with k nearest training vectors

$$\{\mathbf{x}_j \mid j \in \mathrm{NN}_k\},$$

with $\|\mathbf{x}_j - \mathbf{x}\| \leq \|\mathbf{x}_i - \mathbf{x}\|$ if $j \in \mathrm{NN}_k$ and $i \notin \mathrm{NN}_k$:

$$\mathbb{P}(Y = 1 \mid X = x) = \frac{1}{|\mathrm{NN}_k|} \sum_{j \in \mathrm{NN}_k} y_j$$

k-nearest-neighbors (knn)

Perhaps simplest approach to classification:

k-nearest neighbors

Given feature vector $\mathbf{x}$ (e.g., expression levels x_g)

- with k nearest training vectors

$$\{\mathbf{x}_j \mid j \in \mathrm{NN}_k\},$$

with $\|\mathbf{x}_j - \mathbf{x}\| \leq \|\mathbf{x}_i - \mathbf{x}\|$ if $j \in \mathrm{NN}_k$ and $i \notin \mathrm{NN}_k$:

$$\mathbb{P}(Y = 1 \mid X = x) = \frac{1}{|\mathrm{NN}_k|} \sum_{j \in \mathrm{NN}_k} y_j$$

Best when feature space low-dimensional with natural metric.

k-nearest-neighbors is implemented as:

R `class::knn`

Python `sklearn.neighbors.KNeighborsClassifier`

k-nearest-neighbors (knn)

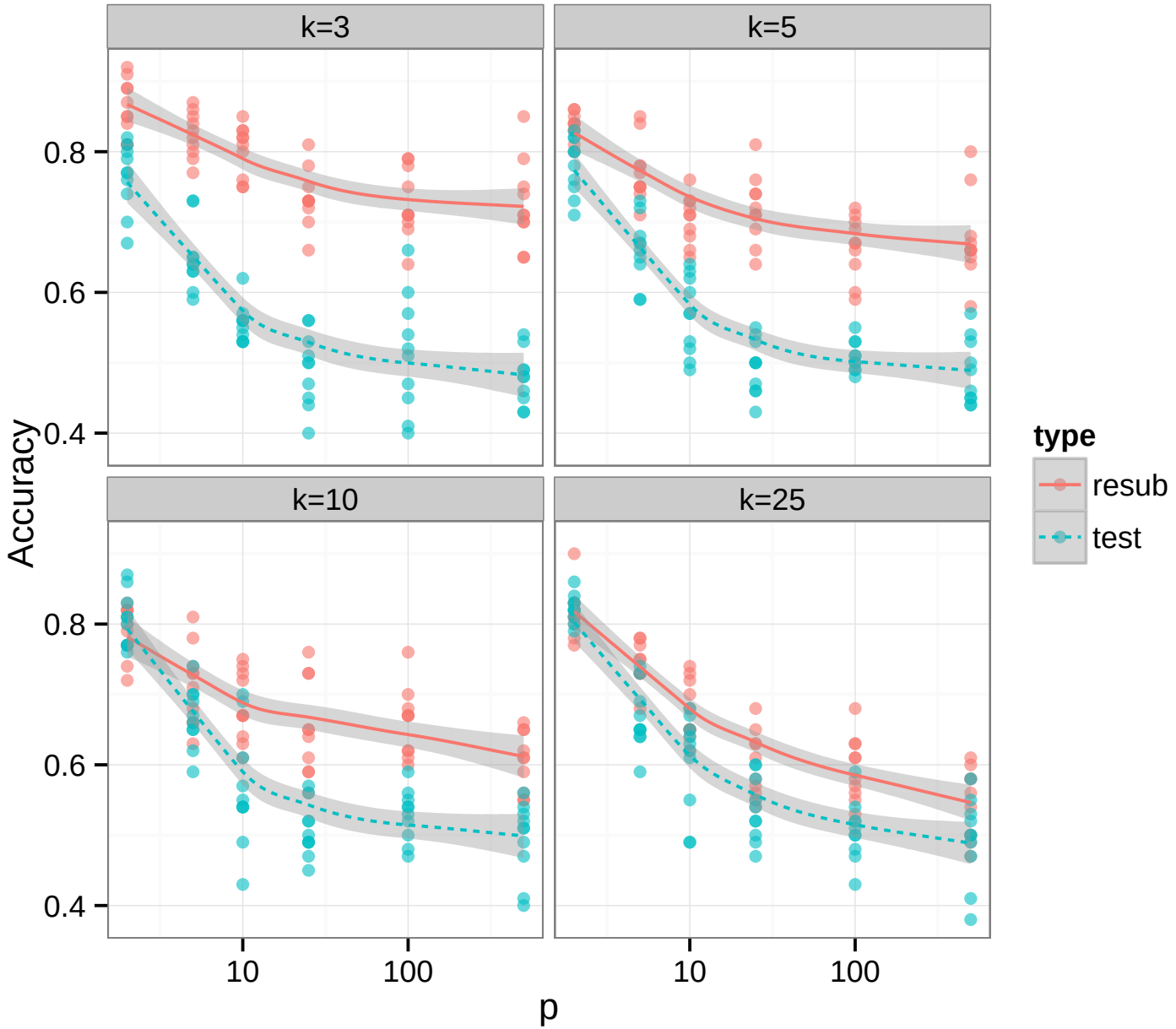
R:

```
knnTest = knn(
    train = xtrain,
    test = xtest,
    cl = ytrain,
    k = 3
)
nCorrect = sum(diag(table(knnTest, ytest)))
```

Python:

```
from sklearn.neighbors import KNeighborsClassifier
knnFit = KNeighborsClassifier(n_neighbors=3)
knnFit.fit(array(xtrain), array(ytrain))
knnTest = Series(knnFit.predict(xtest),
                 index = ytest.index)
nCorrect = sum(diag(pandas.crosstab(knnTest, ytest)))
```

k-nearest-neighbors (knn)



knn and the curse of dimensionality

Volume of p-dimensional hypersphere of radius r is

$$V_p(r) \propto r^p$$

For $\mathbf{x}$ to have many neighbors nearer than r

- must be many $\mathbf{x}_i \in S_{\text{train}}$ in volume $V_p(r)$ centered at $\mathbf{x}$.

knn and the curse of dimensionality

Volume of p-dimensional hypersphere of radius r is

$$V_p(r) \propto r^p$$

For $\mathbf{x}$ to have many neighbors nearer than r

- must be many $\mathbf{x}_i \in S_{\text{train}}$ in volume $V_p(r)$ centered at $\mathbf{x}$.

If the dimensionality p is large and r is small, this is very unlikely.

So must use points far away to guess what's going on at $\mathbf{x}$.

Not surprisingly this doesn't always work ...

knn and the curse of dimensionality

Volume of p-dimensional hypersphere of radius r is

$$V_p(r) \propto r^p$$

For $\mathbf{x}$ to have many neighbors nearer than r

- must be many $\mathbf{x}_i \in S_{\text{train}}$ in volume $V_p(r)$ centered at $\mathbf{x}$.

If the dimensionality p is large and r is small, this is very unlikely.

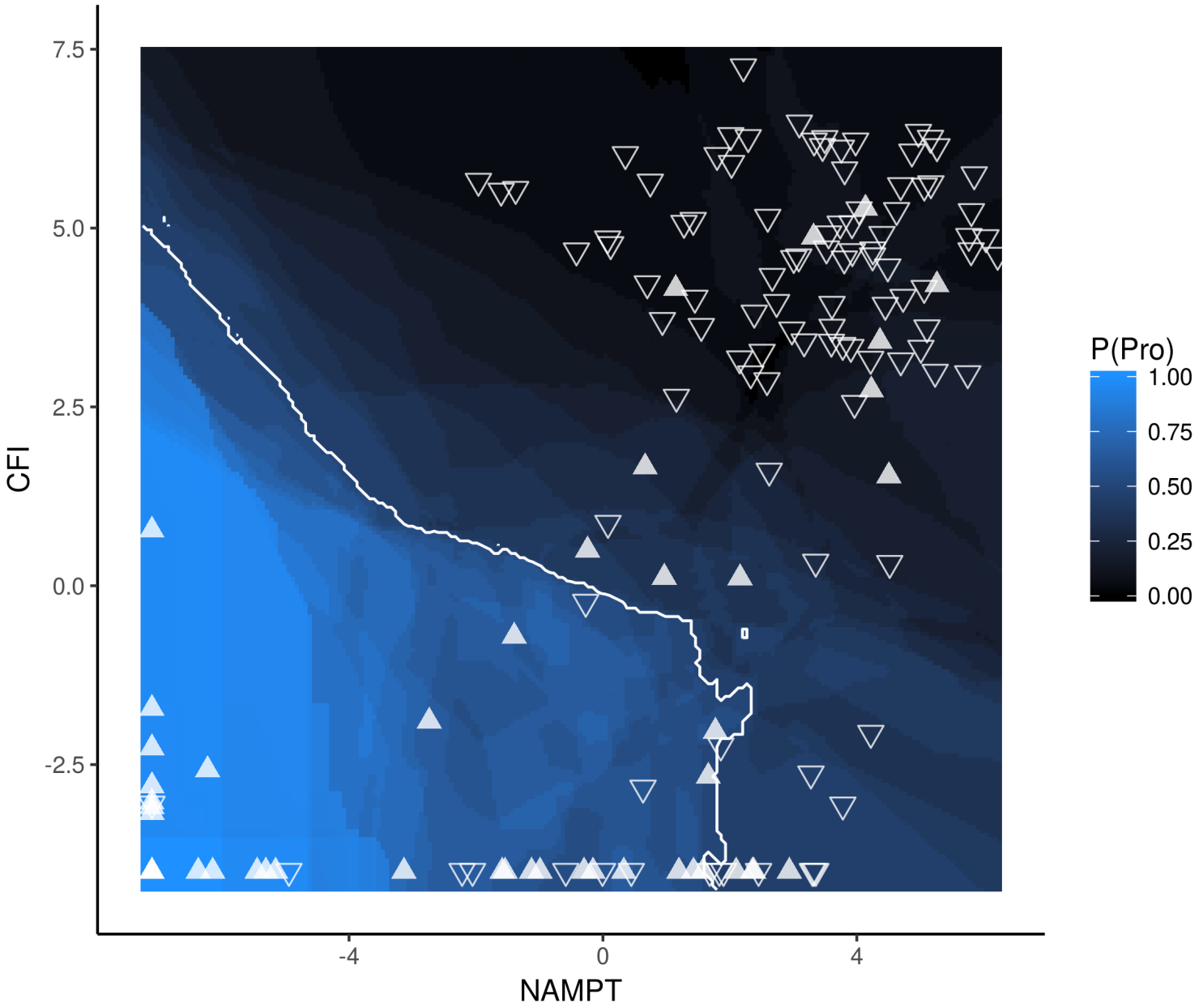
So must use points far away to guess what's going on at $\mathbf{x}$.

Not surprisingly this doesn't always work . . .

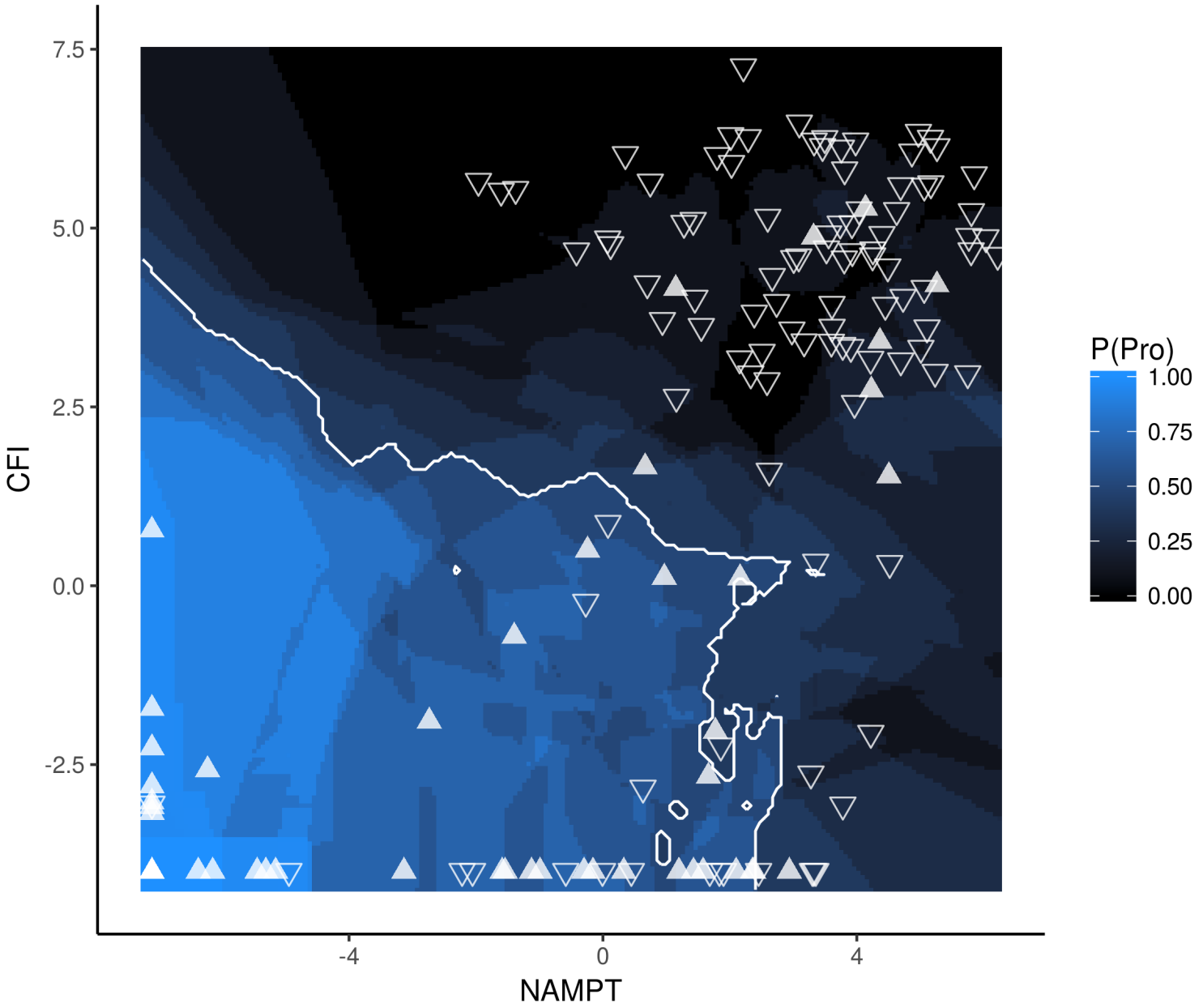
May be better to do **feature selection** or **feature extraction**

- and then fit model using reduced feature set
- will return to this idea later.

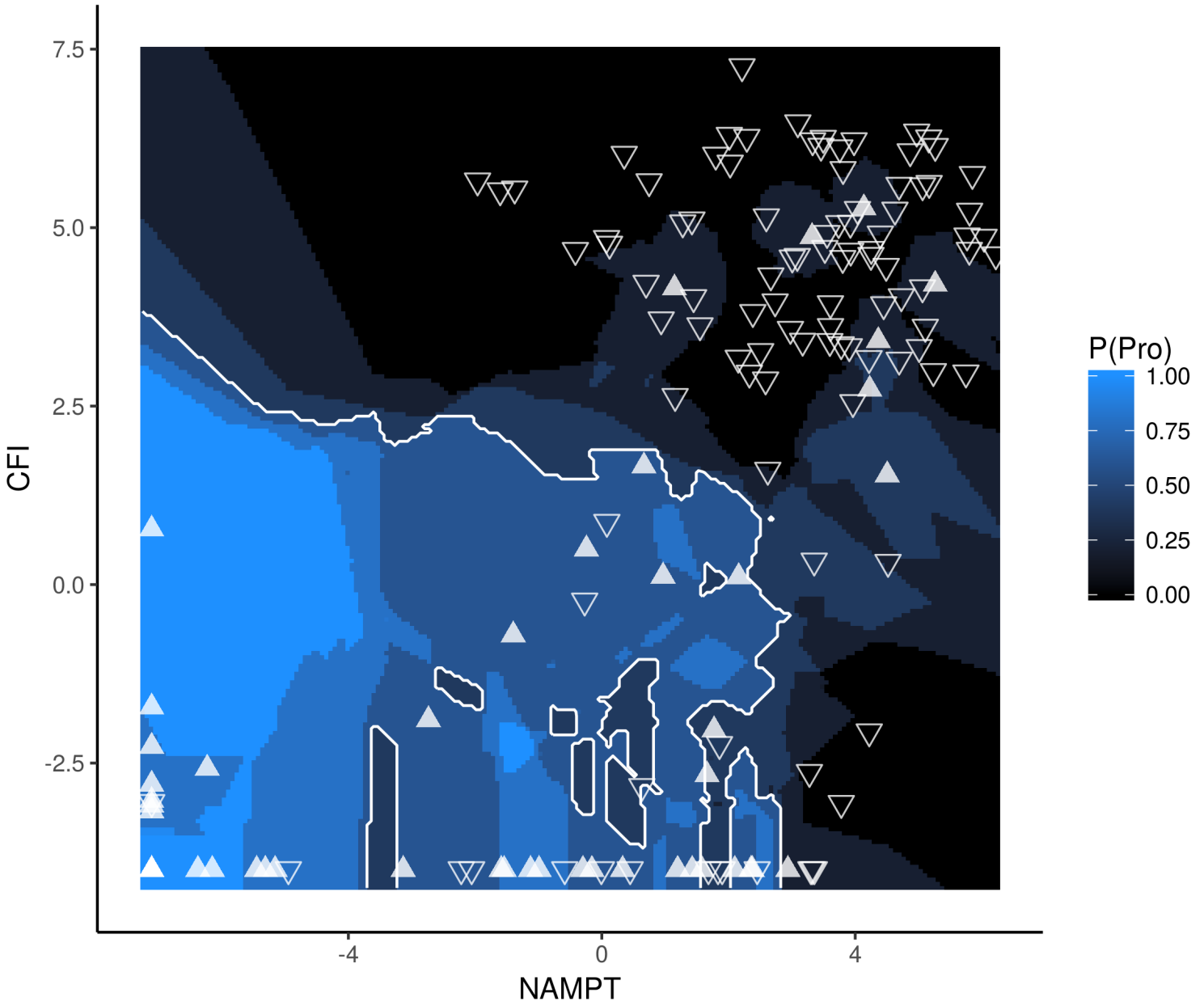
Overfitting: K=20



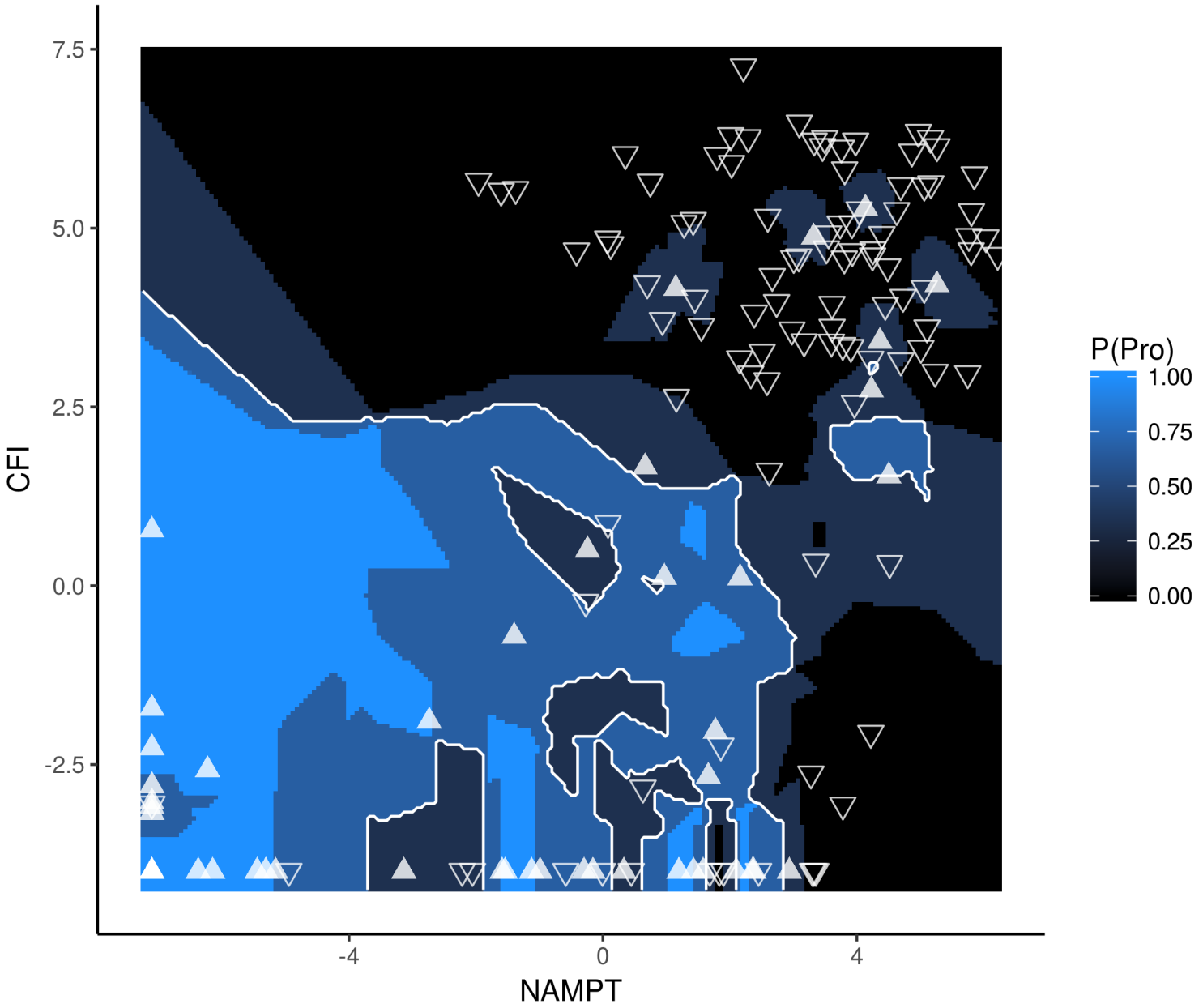
Overfitting: K=10



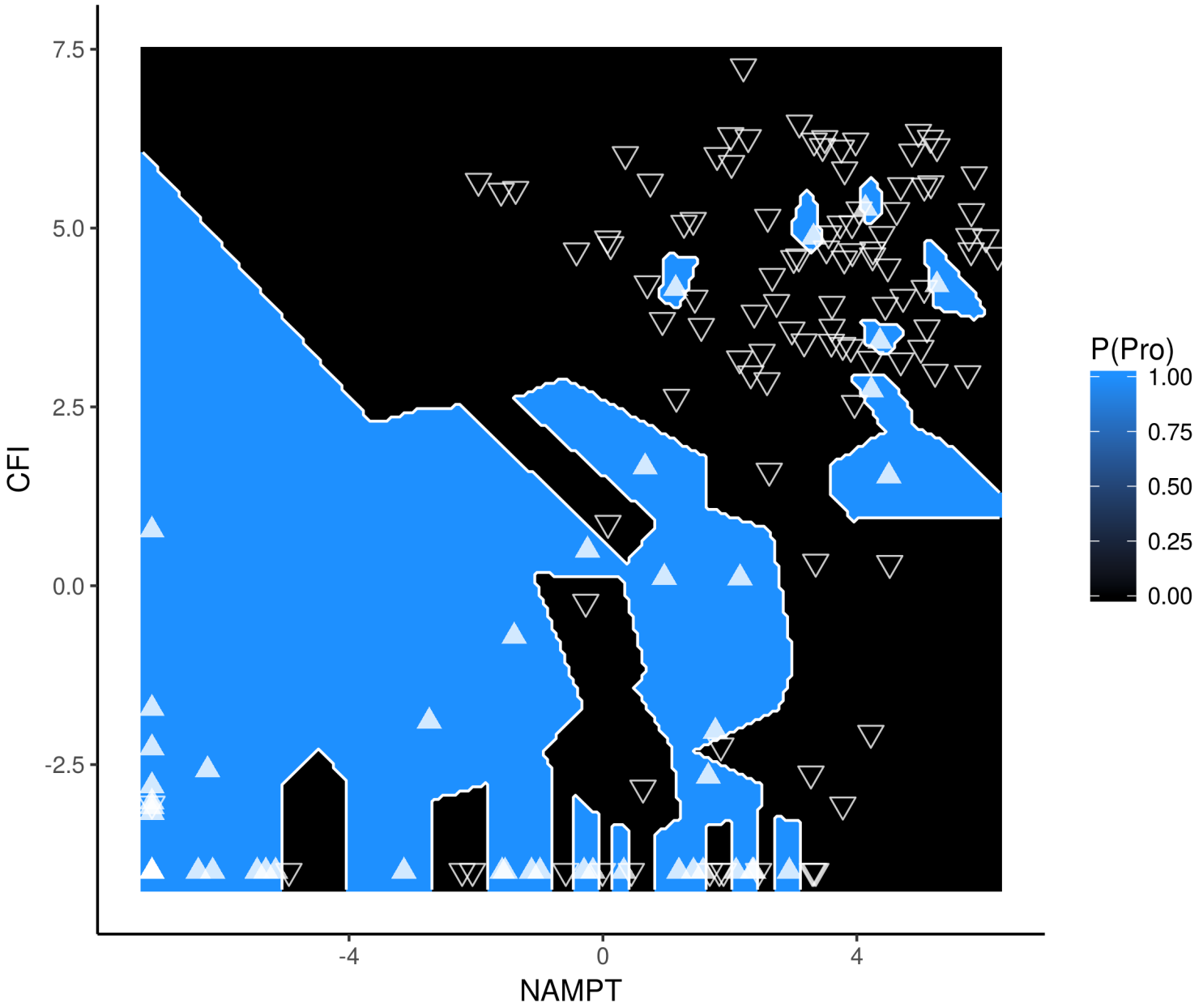
Overfitting: K=5



Overfitting: K=3



Overfitting: K=1



Cross-Validation (CV)

Evaluating performance by resubstitution suffers from bias.

But what if we don't have a test set S_{test} lying around?

Cross-Validation (CV)

Evaluating performance by resubstitution suffers from bias.

But what if we don't have a test set S_{test} lying around?

Can split whatever sample set you have up into training+test set.

Cross-Validation (CV)

Evaluating performance by resubstitution suffers from bias.

But what if we don't have a test set S_{test} lying around?

Can split whatever sample set you have up into training+test set.

If not many samples available:
might split samples S into S_1 and S_2 and then try:

1. first train M on S_1 to obtain $(M, \boldsymbol{\theta}_1)$ for testing on S_2;
2. then train on S_2 to model $(M, \boldsymbol{\theta}_2)$ for testing on S_1.

Unbiased performance estimate could then be obtained

- using the predictions $\mathbb{P}_{M,\boldsymbol{\theta}_2}(Y \mid \mathbf{X})$ for samples in S_1 and
- predictions $\mathbb{P}_{M,\boldsymbol{\theta}_1}(Y \mid \mathbf{X})$ for samples in S_2.

K-Fold Cross-Validation

Generalization: split S up into K subsets S_k for each of which:

1. a model $(M, \boldsymbol{\theta}_{-k})$ is trained using training set $S_{-k} = \bigcup_{q \neq k} S_q$
2. predictions $\mathbb{P}_{M,\boldsymbol{\theta}_{-k}}(Y \mid \mathbf{X} = \mathbf{x}_i)$ are made for samples $i \in S_k$
3. performance estimates are made for each S_k based on $\mathbb{P}_{M,\boldsymbol{\theta}_{-k}}(Y \mid \mathbf{X} = \mathbf{x}_i)$ and then averaged over all K folds.

K-Fold Cross-Validation

Generalization: split S up into K subsets S_k for each of which:

1. a model $(M, \boldsymbol{\theta}_{-k})$ is trained using training set $S_{-k} = \bigcup_{q \neq k} S_q$
2. predictions $\mathbb{P}_{M,\boldsymbol{\theta}_{-k}}(Y \mid \mathbf{X} = \mathbf{x}_i)$ are made for samples $i \in S_k$
3. performance estimates are made for each S_k based on $\mathbb{P}_{M,\boldsymbol{\theta}_{-k}}(Y \mid \mathbf{X} = \mathbf{x}_i)$ and then averaged over all K folds.

Very important:
Cross-validation valid only if all *supervised* steps in fitting model conducted separately within each of the k folds.

5-Fold Cross-Validation

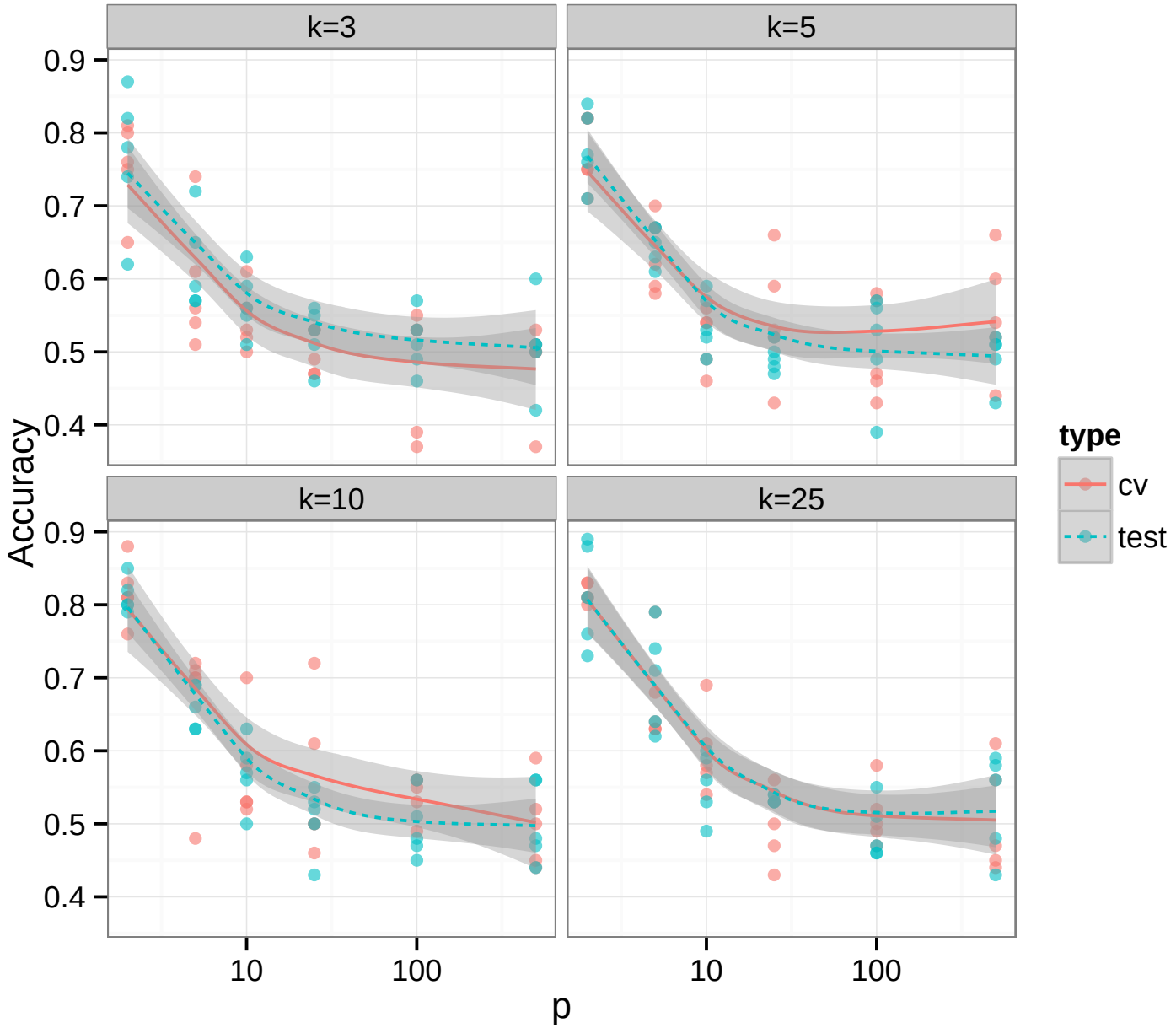
R:

```
library(caret)
knnCV = train(
    x = xtrain,
    y = ytrain,
    method = "knn",
    tuneGrid = data.frame(k=3),
    trControl = trainControl(method="cv", number=5)
)
cvAccuracyEstimate = knnCV$results[ , "Accuracy"]
```

Python:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import cross_val_score
knnClass = KNeighborsClassifier(n_neighbors=3)
cvAccs = cross_val_score(estimator = knnClass,
                         X = array(xtrain),
                         y = array(ytrain),
                         cv = 5)
cvAccuracyEstimate = mean(cvAccs)
```

5-Fold Cross-Validation



Metrics—Binomial

There are many ways to measure performance for classifiers; most are based only on the "discretized calls" $\hat{y}$

$$\hat{y}_{M,\boldsymbol{\theta},\psi} = \begin{cases} 1 & \text{if } \mathbb{P}_{M,\boldsymbol{\theta}}(Y = 1 \mid \mathbf{X} = \mathbf{x}) \geq \psi \\ 0 & \text{otherwise} \end{cases}$$

given some threshold ψ (e.g., $\psi = 0.5$).

Metrics—Binomial

There are many ways to measure performance for classifiers; most are based only on the "discretized calls" $\hat{y}$

$$\hat{y}_{M,\boldsymbol{\theta},\psi} = \begin{cases} 1 & \text{if } \mathbb{P}_{M,\boldsymbol{\theta}}(Y = 1 \mid \mathbf{X} = \mathbf{x}) \geq \psi \\ 0 & \text{otherwise} \end{cases}$$

given some threshold ψ (e.g., $\psi = 0.5$).
Given a sample set S of size $|S| = N$ composed of:

- **TP** true positive samples: $y = \hat{y} = 1$
- **TN** true negative samples: $y = \hat{y} = 0$
- **FP** false positive samples: $y = 0, \hat{y} = 1$
- **FN** false negative samples: $y = 1, \hat{y} = 0$,

define

- **Accuracy** fraction of calls correct $\left(\frac{TP+TN}{N}\right)$
- **Sensitivity** fraction of calls correct when $y = 1$ $\left(\frac{TP}{TP+FN}\right)$
- **Specificity** fraction of calls correct when $y = 0$ $\left(\frac{TN}{TN+FP}\right)$
- **PPV** fraction of calls correct when $\hat{y} = 1$ $\left(\frac{TP}{TP+FP}\right)$
- **NPV** fraction of calls correct when $\hat{y} = 0$ $\left(\frac{TN}{TN+FN}\right)$.

Metrics—Receiver Operating Characteristic (ROC)

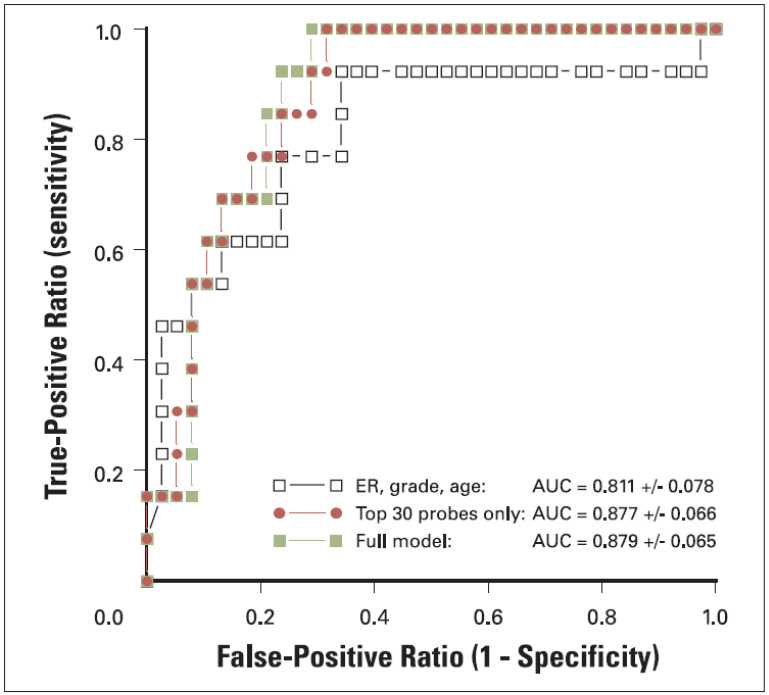


Fig 3. Receiver operating characteristic curves of three distinct pathologic complete response prediction models. The performance of the Diagonal Linear Discriminant Analysis–30 predictor and a predictor based on clinical variables and a combined clinical + pharmacogenomic prediction model are shown in the validation set (n = 51). ER, estrogen receptor; AUC, area under the curve.

Taken from Hess *et al.* (2006).

Consider binomial metrics over range of threshold values ψ.

Receiver operating characteristic (ROC) curve: sensitivity vs. specificity.

Metrics—Receiver Operating Characteristic (ROC)

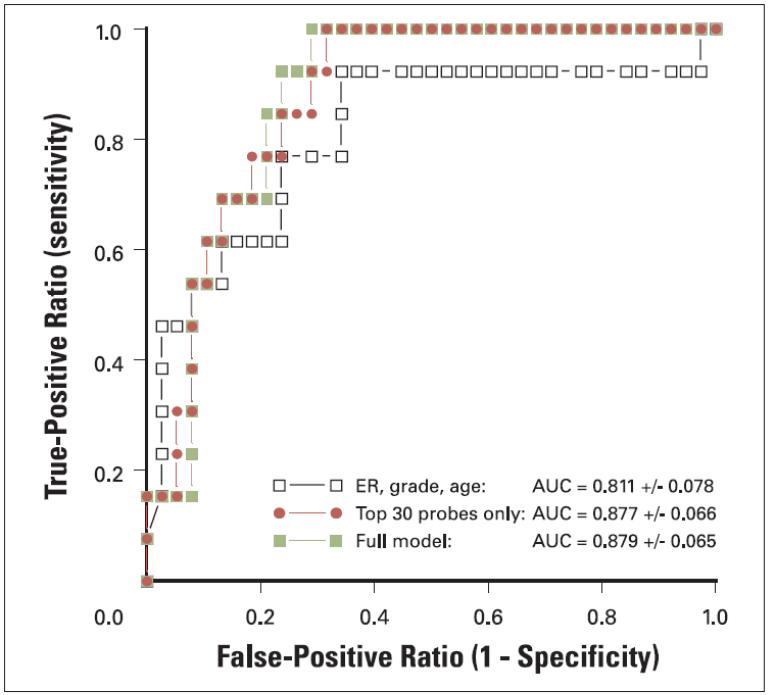


Fig 3. Receiver operating characteristic curves of three distinct pathologic complete response prediction models. The performance of the Diagonal Linear Discriminant Analysis–30 predictor and a predictor based on clinical variables and a combined clinical + pharmacogenomic prediction model are shown in the validation set (n = 51). ER, estrogen receptor; AUC, area under the curve.

Taken from Hess *et al.* (2006).

Consider binomial metrics over range of threshold values ψ.

Receiver operating characteristic (ROC) curve: sensitivity vs. specificity.

Area under ROC curve (**AUC**) equals probability that

- score $\mathbb{P}(Y = 1 \mid \mathbf{X} = \mathbf{x})$ of a randomly chosen positive case ($y = 1$)

is higher than

- score of a randomly chosen negative case ($y = 0$).

Feature selection

Often assumed that expression patterns of most genes either:

1. uninformative or
2. redundant with a few maximally useful markers

with respect to a particular classification task.

Feature selection attempts to identify optimal set of markers for inclusion in model.

Feature selection not always required but resulting simplification:

1. reduces computational workload,
2. can help to avoid overfitting (though feature selection can itself be susceptible to overfitting), and
3. facilitates model platform migration.

Taxonomy (adapted from Saeys *et al.* (2007))

Filter Selection done before and independently of classifier construction. Can be univariate or multivariate.

Wrapper Embed classifier construction within feature selection process. Heuristic search methods compare models, favor adding or removing features based on optimization of some specified metric on resulting classifiers.

Embedded Feature selection is inherently built into some classifier construction methods.

Taxonomy (adapted from Saeys *et al.* (2007))

Category	Advantages	Disadvantages	Examples
Filter	*Univariate*		
	Fast Scalable Independent of classifier	- feature dependencies - interaction w/classifier	t-test, ANOVA Wilcox test Rank Product
	Multivariate		
	+ feature dependencies Independent of classifier Intermediate complexity	Slower Less Scalable - interaction w/classifier	CFS Markov Blanket Filter
Wrapper	*Deterministic*		
	Simple + interaction w/classifier + feature dependencies	Risk of over-fitting Greedy (local optima) Classifier dependent selection	Forward Selection Backward Elimination Plus q minus r
	Randomized		
	Less prone to local optima + interaction w/classifier + feature dependencies	High risk over-fitting Computationally intensive Classifier dependent selection	Simulated Annealing Randomized Hill Climbing Genetic Algorithms
Embedded	+ interaction w/classifier + feature dependencies Intermediate complexity	No modularity Restrict algorithms	Decision trees Weighted Naive Bayes LASSO regression

Linear Models for Univariate Feature Selection

Most common univariate filter is t-test (or F-test if more than 2 groups) originating from model

$$\mathbb{P}(X_g = x \mid Y = y) = \frac{1}{\sqrt{2\pi\sigma_g^2}} \exp\left[\frac{(x - \mu_{yg})^2}{2\sigma_g^2}\right]$$

Now considering conditional probabilities of X_g given Y ...

... and considering each X_g separately!

- "Univariate" analysis; not necessarily realistic, but tractable.

Linear Models

Alternately, this model may be described by

$$X_g = \mu_{0g} + (\mu_{1g} - \mu_{0g})Y + \sigma_g \epsilon_g$$

with $\epsilon_g \sim \mathcal{N}(0, 1)$, implying

$$\mathbb{E}(X_g \mid Y = y) = \mu_{yg}$$
$$\mathbb{V}(X_g \mid Y = y) = \sigma_g^2$$

Can then rank features based on $|t_g|$ where:

$$t_g = \frac{\hat{\mu}_{0g} - \hat{\mu}_{1g}}{\hat{\sigma}_g \sqrt{\frac{1}{n_0} + \frac{1}{n_1}}}$$

Linear Models

Alternately, this model may be described by

$$X_g = \mu_{0g} + (\mu_{1g} - \mu_{0g})Y + \sigma_g \epsilon_g$$

with $\epsilon_g \sim \mathcal{N}(0, 1)$, implying

$$\mathbb{E}(X_g \mid Y = y) = \mu_{yg}$$
$$\mathbb{V}(X_g \mid Y = y) = \sigma_g^2$$

Can then rank features based on $|t_g|$ where:

$$t_g = \frac{\hat{\mu}_{0g} - \hat{\mu}_{1g}}{\hat{\sigma}_g \sqrt{\frac{1}{n_0} + \frac{1}{n_1}}}$$

While X_g is linear in Y here,

- it is linearity in μ_{yg} that makes "linear model" in statistics.

Linear Models

Alternately, this model may be described by

$$X_g = \mu_{0g} + (\mu_{1g} - \mu_{0g})Y + \sigma_g \epsilon_g$$

with $\epsilon_g \sim \mathcal{N}(0, 1)$, implying

$$\mathbb{E}(X_g \mid Y = y) = \mu_{yg}$$
$$\mathbb{V}(X_g \mid Y = y) = \sigma_g^2$$

Can then rank features based on $|t_g|$ where:

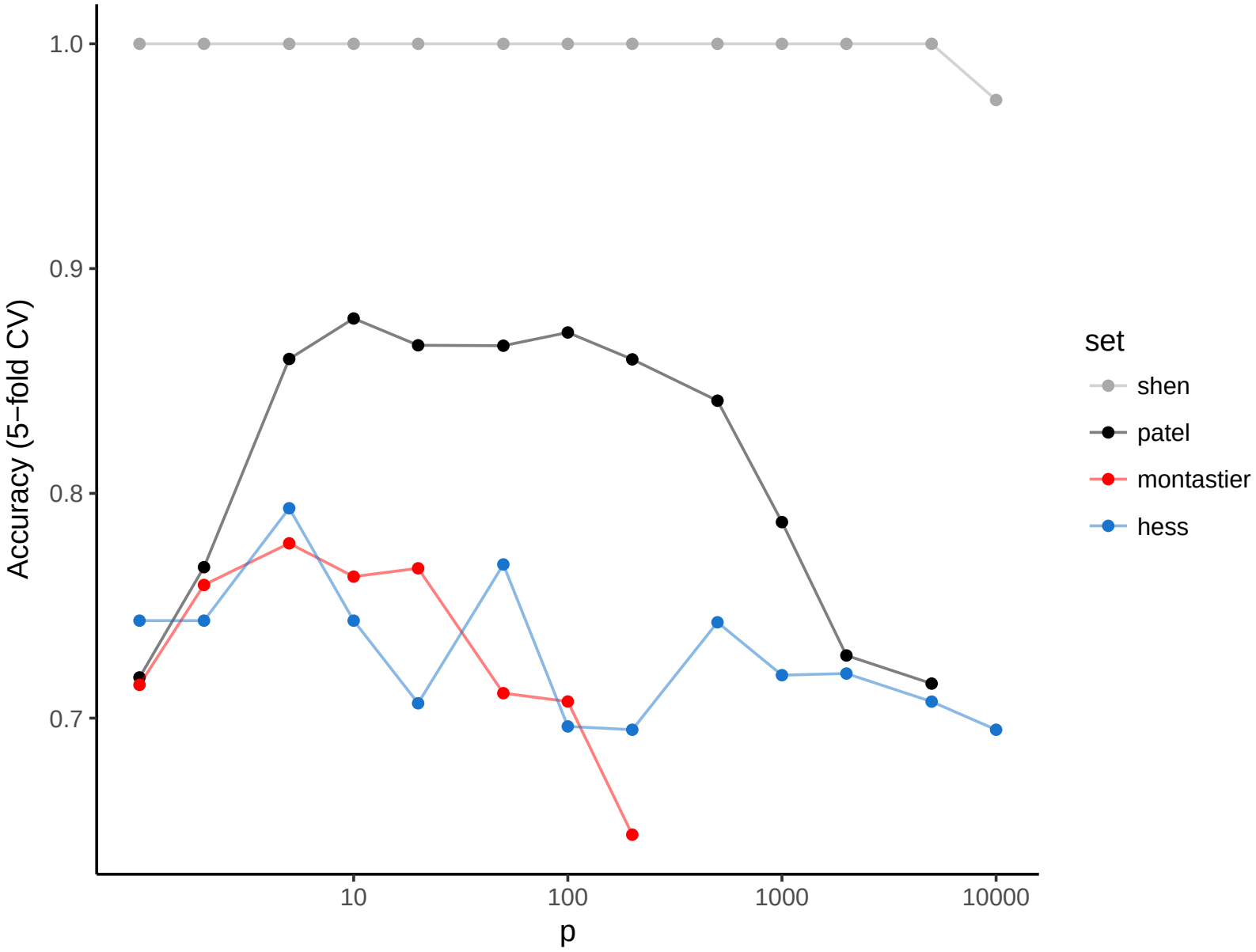
$$t_g = \frac{\hat{\mu}_{0g} - \hat{\mu}_{1g}}{\hat{\sigma}_g \sqrt{\frac{1}{n_0} + \frac{1}{n_1}}}$$

While X_g is linear in Y here,

- it is linearity in μ_{yg} that makes "linear model" in statistics.

With a few modifications linear models can model $Y \mid \mathbf{X}$ too.

knn Accuracy With *t*-Test Feature Selection



References I

Hess, Kenneth R, Anderson, Keith, Symmans, W Fraser, Valero, Vicente, Ibrahim, Nuhad, Mejia, Jaime A, Booser, Daniel, Theriault, Richard L, Buzdar, Aman U, Dempsey, Peter J, *et al.* . 2006. Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with paclitaxel and fluorouracil, doxorubicin, and cyclophosphamide in breast cancer. *Journal of Clinical Oncology*, **24**(26), 4236–4244.

Kang, Huining, Chen, I-Ming, Wilson, Carla S, Bedrick, Edward J, Harvey, Richard C, Atlas, Susan R, Devidas, Meenakshi, Mullighan, Charles G, Wang, Xuefei, Murphy, Maurice, *et al.* . 2010. Gene expression classifiers for relapse-free survival and minimal residual disease improve risk classification and outcome prediction in pediatric B-precursor acute lymphoblastic leukemia. *Blood*, **115**(7), 1394–1405.

Roweis, Sam, & Ghahramani, Zoubin. 1999. A unifying review of linear Gaussian models. *Neural Computation*, **11**(2), 305–345.

Saeys, Yvan, Inza, Iñaki, & Larrañaga, Pedro. 2007. A review of feature selection techniques in bioinformatics. *Bioinformatics*, **23**(19), 2507–2517.