MA8701 Advanced methods in statistical inference and learning

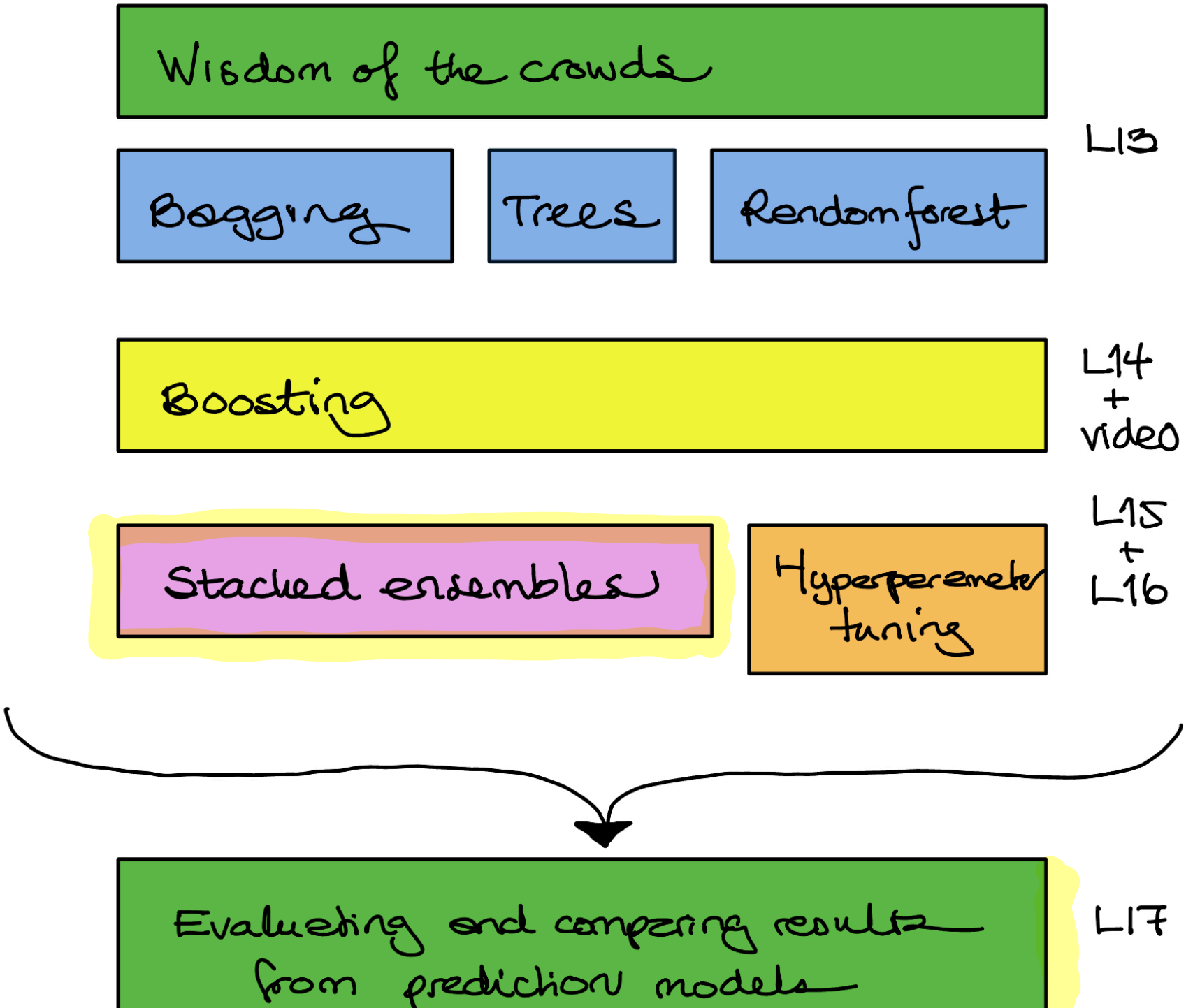
Part 3: Ensembles. L15: Stacked ensembles

Mette Langaas

~~3/5/23~~

lectured 06.03.2023

Before we start



Stacked ensembles

The idea is to combine predictors is well-known in statistics — but how should this be done in practice ?

aka super learner or generalized stacking

What is it?

The Stacked Esembles is an algorithm that combines

- multiple, (typically) diverse prediction methods (learning algorithms) called *base learners* (first-level) into a
- a second-level *metalearner* - which can be seen as a *single* method.

0) Training data (level-zero data) $Z_i = (X_i, Y_i)$, $i = 1, \dots, N$ — $X_i \in \mathbb{R}^p$, $Y_i \in \mathbb{R}$ or $\{0,1\}$ or $\{1, \dots, K\}$

1) Base learners $\psi^{\ell}(x)$ $\ell = 1, \dots, L$: from possibly different methods

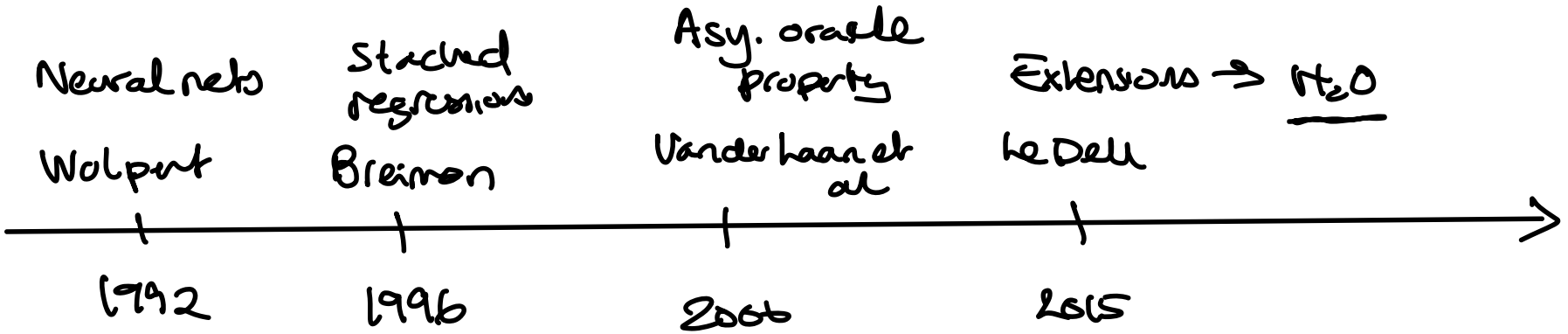
2) Meta learner Φ : an "optimal" combination of the base learners

Literature

- Erin Le Dell (2015): Scalable Ensemble Learning and Computationally Efficient Variance Estimation. PhD Thesis, University of California, Berkeley. or https://github.com/ledell/phd-thesis. Section 2.

Supporting literature

- Breiman (1996)
- Laan, Polley, and Hubbard (2007)
- Polley, Rose, and Laan (2011)



SUGGESTED "NEW" ENSEMBLE METHOD

0) $O_i = (X_i, Y_i)$ $i = 1, .., N$ level-zero data

Assume $Y_i \in \mathbb{R}$ and $X_i \in \mathbb{R}^p$

1) We first fit $\hat{\psi}^1(x), \hat{\psi}^2(x), \hat{\psi}^3(x), \ldots \hat{\psi}^L(x)$

$\hat{\psi}^1$: linear model LS; $\hat{\psi}^2$: lasso; $\hat{\psi}^3$: RF; $\hat{\psi}^L$: xgboost

to be L predictors for Y.

2) We would like to use one or all $\hat{\psi}^l$ to combine into meta learner $\hat{\Phi}(\hat{\psi}^1(x), \hat{\psi}^2(x), \ldots, \hat{\psi}^L(x))$ where

$\hat{\Phi}$: meta; $\hat{\psi}^1$: base

$\hat{\Phi}$ might be a linear model fit by LS.

ESTIMATION STRATEGY

1) Fit $\psi^1(x), \ldots, \psi^L(x)$ using the level-zero training data

$\Rightarrow \hat{\psi}^1(x), \ldots, \hat{\psi}^L(x)$

2) $\hat{\Phi}(\hat{\psi}^1(x), \ldots, \hat{\psi}^L(x))$ found of the form

$$\phi(x) = \alpha_1 \cdot \hat{\psi}^1(x) + \alpha_2 \hat{\psi}^2(x) + \cdots + \alpha_L \hat{\psi}^L(x)$$

by minimizing the squared loss

$$\sum_{i=1}^{N} (y_i - \alpha_1 \hat{\psi}^1(x_i) - \alpha_2 \hat{\psi}^2(x_i) - \cdots - \alpha_L \hat{\psi}^L(x_i))^2$$

↖ not so good when $\hat{\psi}$'s correlated

Is this a good strategy?

There are two issues — what can that be?

overfitting — same data in step 1+2

STACKED ENSEMBLE FITTING STRATEGY

1) Divide the training data into V folds (V=5 or 10 popular)

$X_{(1)}, \ldots, X_{(V)}$

For each of the L base learners use the same fold →

fit on (V-1) and predict on 1 to produce predictions Z

$\hat{\psi}^1(x) \quad \hat{\psi}^2(x) \quad \hat{\psi}^L(x)$

$$Z = \begin{bmatrix} z_{11} & z_{21} & z_{L1} \\ \vdots & & \\ \{z_{li}\} & & \\ z_{1N} & z_{2N} & z_{LN} \end{bmatrix}$$

$l = 1, \ldots, L$
$i = 1, \ldots, N$

level 1 data ($N \times L$)

What can the base learners be?

LS, elastic net, neural net, RF, CART, Bagging
lasso, ridge, boosting, PCR, k-NN
GBM

"Any" method that produces a prediction - "all" types of problems.

- linear regression
- lasso
- cart
- random forest with mtry=value 1
- random forest with mtry=value 2
- xgboost with hyperparameter set 1
- xgboost with hyperparameter set 2
- neural net with hyperparameter set 1

2) Fit the meta learner

Input: (Y_i, Z_i)

Y_i: $\mathbb{R}$, $\{0,1\}$, $\{1,\dots,K\}$

$Z_{1i},\dots,Z_{Li}$ ← the L base learners predictions for obs i

NB Y_i was not used to produce the Z_i's

Simplest case: regression

$$\hat{\phi}(x) = \hat{\alpha}_1 \hat{\psi}^1(x) + \dots + \hat{\alpha}_L \hat{\psi}^L(x)$$

$$\hat{\phi}(x_i) = \hat{\alpha}_1 Z_{1i} + \quad + \hat{\alpha}_L Z_{Li}$$

fit using elastic net

$\Rightarrow$ get $\hat{\alpha}_1,\dots,\hat{\alpha}_L$ and $\hat{\alpha}_S \geq 0$

For classification $\{0,1\}$

probability predictor

$$\text{let } \hat{\eta}(x) = \hat{\alpha}_1 \hat{\psi}^1(x) + \cdots + \hat{\alpha}_k \hat{\psi}^k(x) + \hat{\alpha}_0$$

or maybe

$$\hat{\eta}(x) = \hat{\alpha}_1 \text{logit}(\hat{\psi}^1(x)) + \cdots + \hat{\alpha}_k \text{logit}(\hat{\psi}^k(x)) + \hat{\alpha}_0$$

and fit logistic elastic net

$$\hat{\phi}(x) = \frac{e^{\hat{\eta}(x)}}{1+e^{\hat{\eta}(x)}} = \text{expit}(\hat{\eta}(x)) \quad \text{to get } \hat{\alpha}_0, \hat{\alpha}_1, \ldots, \hat{\alpha}_k$$

$\hat{\phi}$: prediction

BUT, also use other methods, for example:

S. ntrw.ac/MA8701V2023examsignup

- the mean (bagging)
- constructed by minimizing the
  - squared loss (ordinary least squares) or
  - non-negative least squares (most popular)
- ridge or lasso regression
- logistic regression (for binary classification)
- constructed by minimizing 1-ROC-AUC

not recommended

← next page

or elastic net

← default elastic net

LeDell PhD ← "suited for class imbalance"

2. Why Non-negativity Constraints Work

$\hat{\psi}(x)$

Breiman 1996

Only partial answers are available. Suppose that the $\{v_k(\boldsymbol{x})\}$ are strongly correlated and the $\{\alpha_k\}$ are chosen using least squares or ridge regression. Then there is no guarantee that the resulting predictor $\sum_k \alpha_k v_k(\boldsymbol{x})$ will stay near the range $[\min_k v_k(\boldsymbol{x}), \max_k v_k(\boldsymbol{x})]$ and generalization may be poor.

Now consider imposing the non-negativity constraints on the $\{\alpha_k\}$ together with the additional constraint $\sum \alpha_k = 1$. For any $\{\alpha_k\}$ satisfying the constraints $\alpha_k \geq 0$, $\sum_k \alpha_k = 1$,

$$v(\boldsymbol{x}) = \sum_k \alpha_k v_k(\boldsymbol{x})$$

is an "interpolating" predictor. That is, for every value of $\boldsymbol{x}$,

$$\min_k v_k(\boldsymbol{x}) \leq v(\boldsymbol{x}) \leq \max_k v_k(\boldsymbol{x}).$$

So, what our procedure does is to find the best "interpolating" predictor.

3) Re-estimate $\hat{\psi}^L$ using all train data, but keep the meta learner (λ's or more complex if e.g. xgboost)

$\hat{\Phi}(x)$ simplest case of linear function meta

$$= \sum_{l=1}^{L} \lambda_L \; \underbrace{\hat{\psi}^l(x)}_{\text{new}}$$

λ_L: from step 2

new, all train data

4) Use for prediction on new data x^*

- use the $\hat{\psi}^l_{\text{new}}(x^*)$, $l = 1, \dots, L$ to produce $\underset{\text{L vect}}{z^*}$ predictors
- feed z^* to the meta learner from 2nd 3

EXPLAIN?

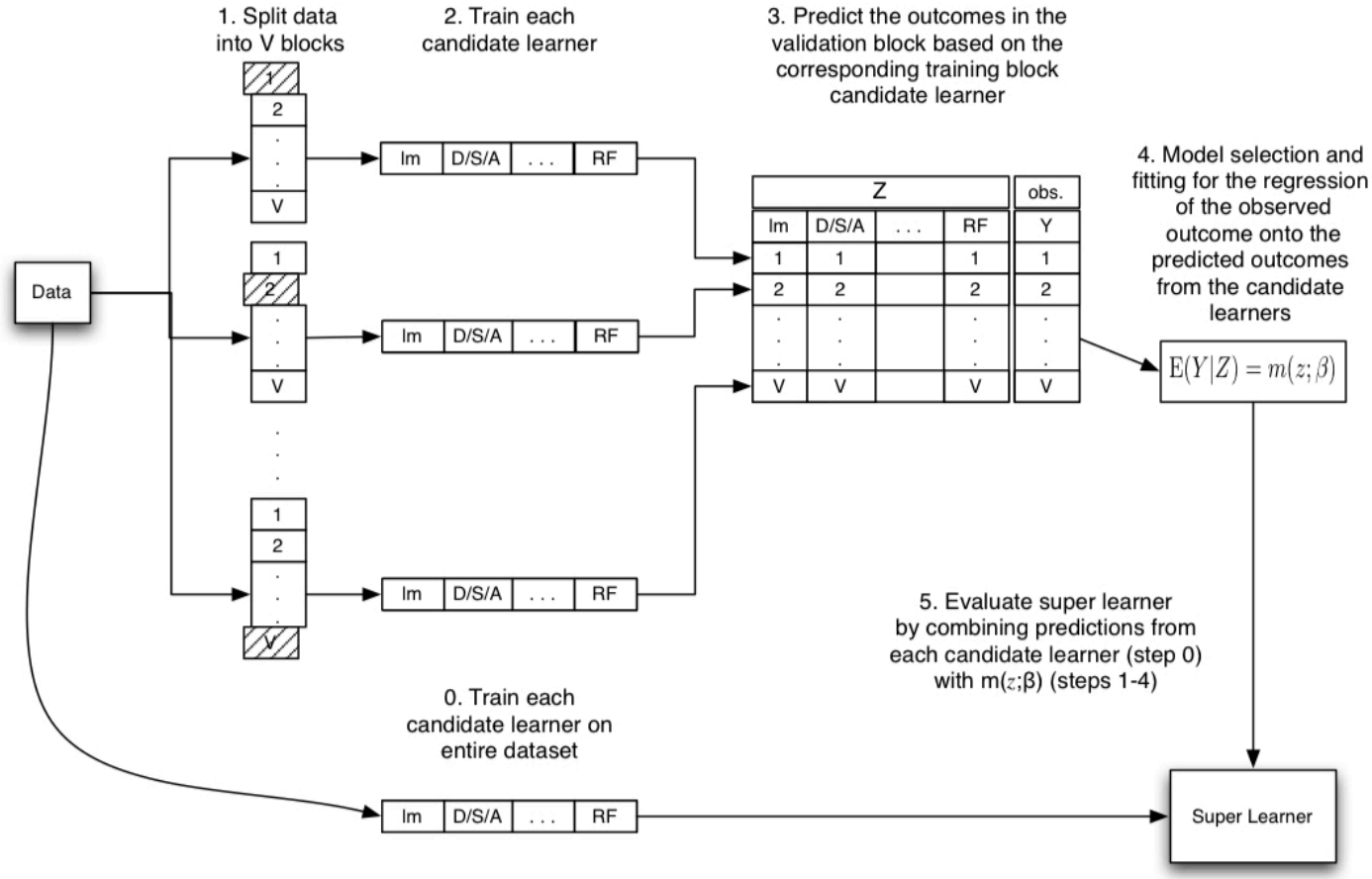


Figure 1: Flow Diagram for Super Learner

(Class notes: Study Figure 3.2 from Polley, Rose, and Laan (2011) and/or Figure 1 from Laan, Polley, and Hubbard (2007))

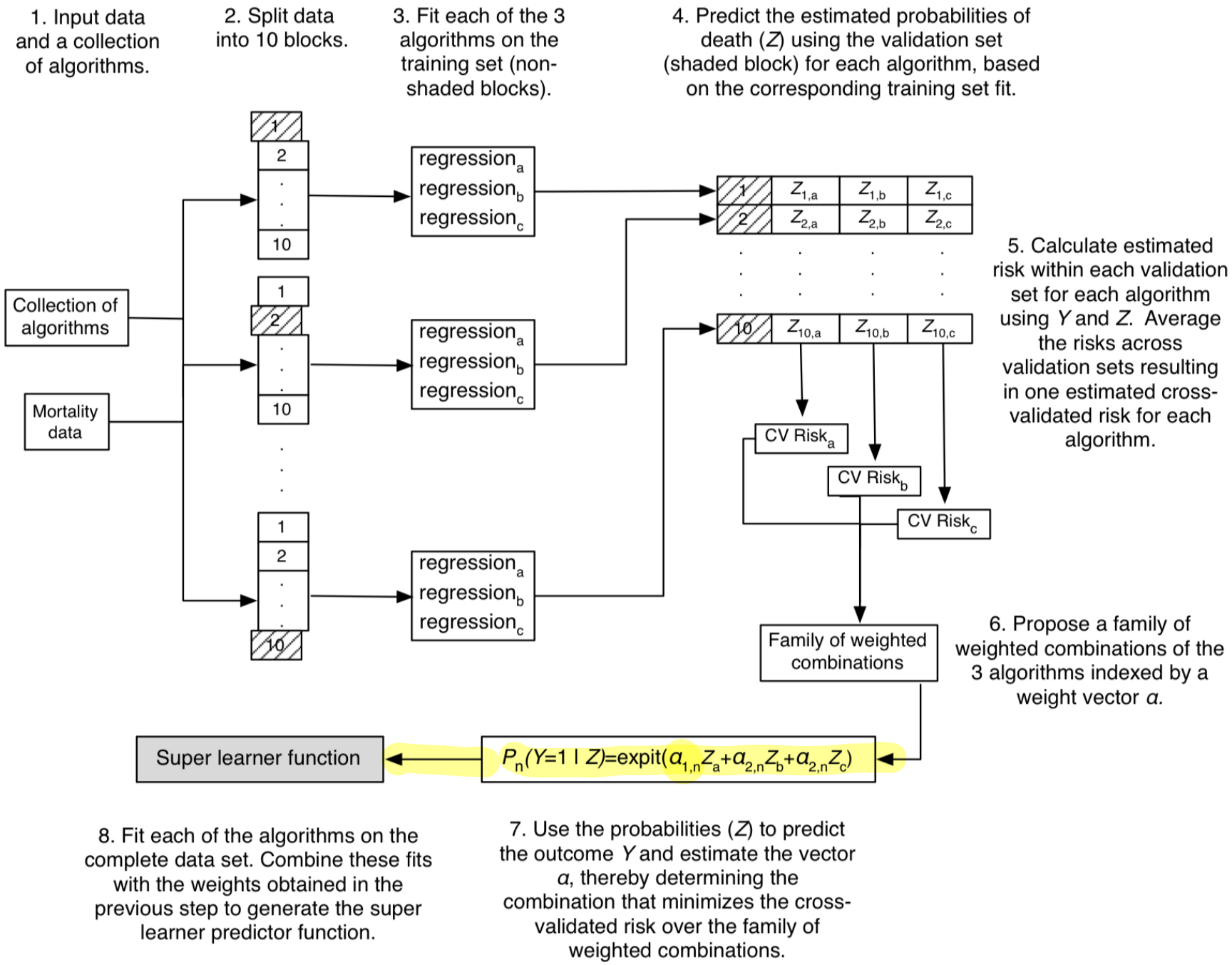


Fig. 3.2 Super learner algorithm for the mortality study example

The metalearning

Some observations

only one $\hat{\alpha}_L \neq 0$

- The term *discrete super learner* is used if the base learner with the lowest risk (i.e. CV-error) is selected.
- Since the predictions from multiple base learners may be highly correlated - the chosen method should perform well in that case (i.e. ridge and lasso).
- When minimizing the squared loss it has been found that adding a non-negativity constraint $\alpha_l \geq 0$ works well,
- and also the additivity constraint $\sum_{l=1}^{L} \alpha_l = 1$ - the ensemble is a *convex combination* of the base learners.
- Non-linear optimization methods may be employed for the metalearner if no existing algorithm is available
- Historically a regularized linear model has "mostly" been used
- For classification the logistic response function can be used on the linear combination of base learners (Figure 3.2 Polley, Rose, and Laan (2011)).

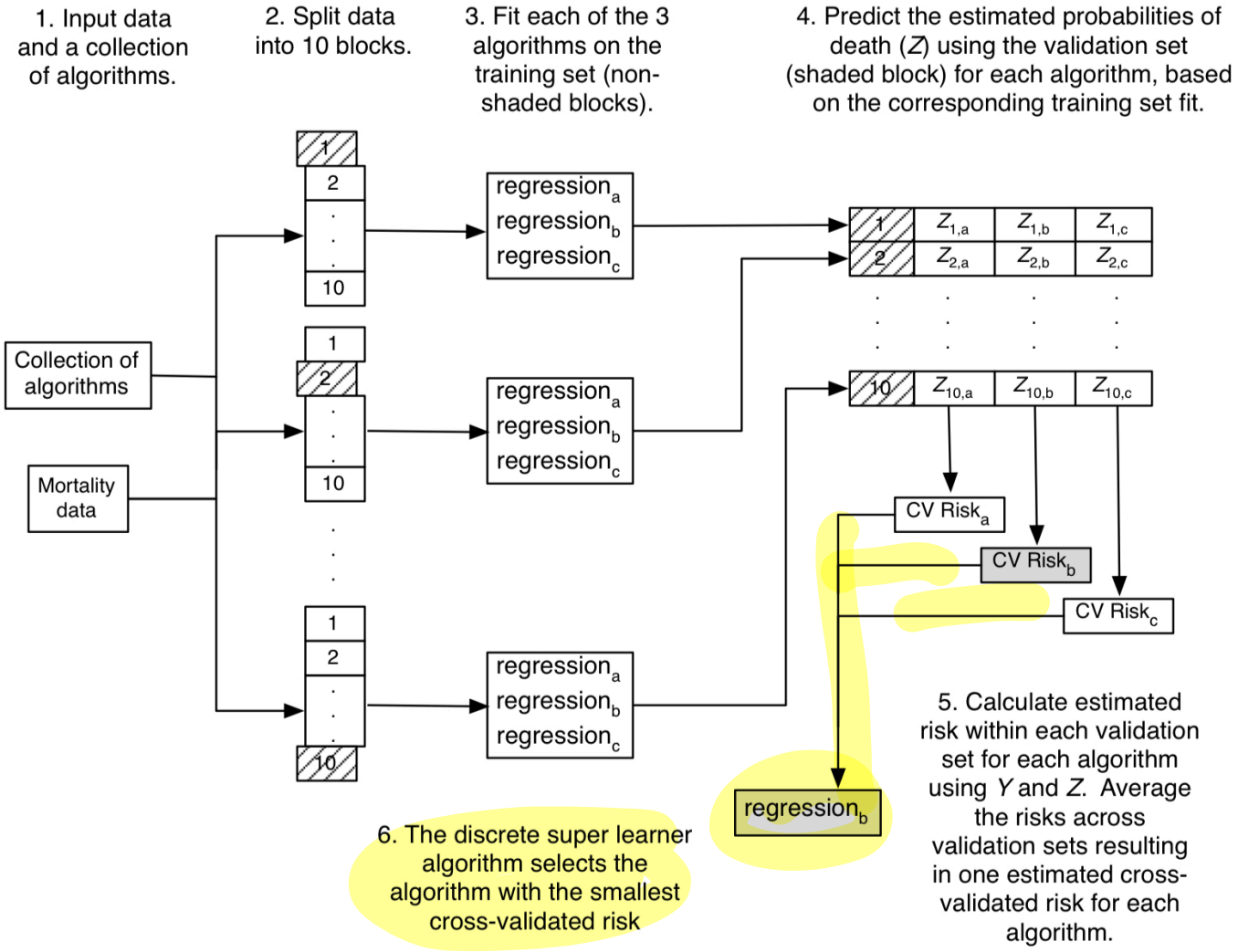


Fig. 3.1 Discrete super learner algorithm for the mortality study example where $\bar{Q}_n^b(A, W)$ is the algorithm with the smallest cross-validated risk

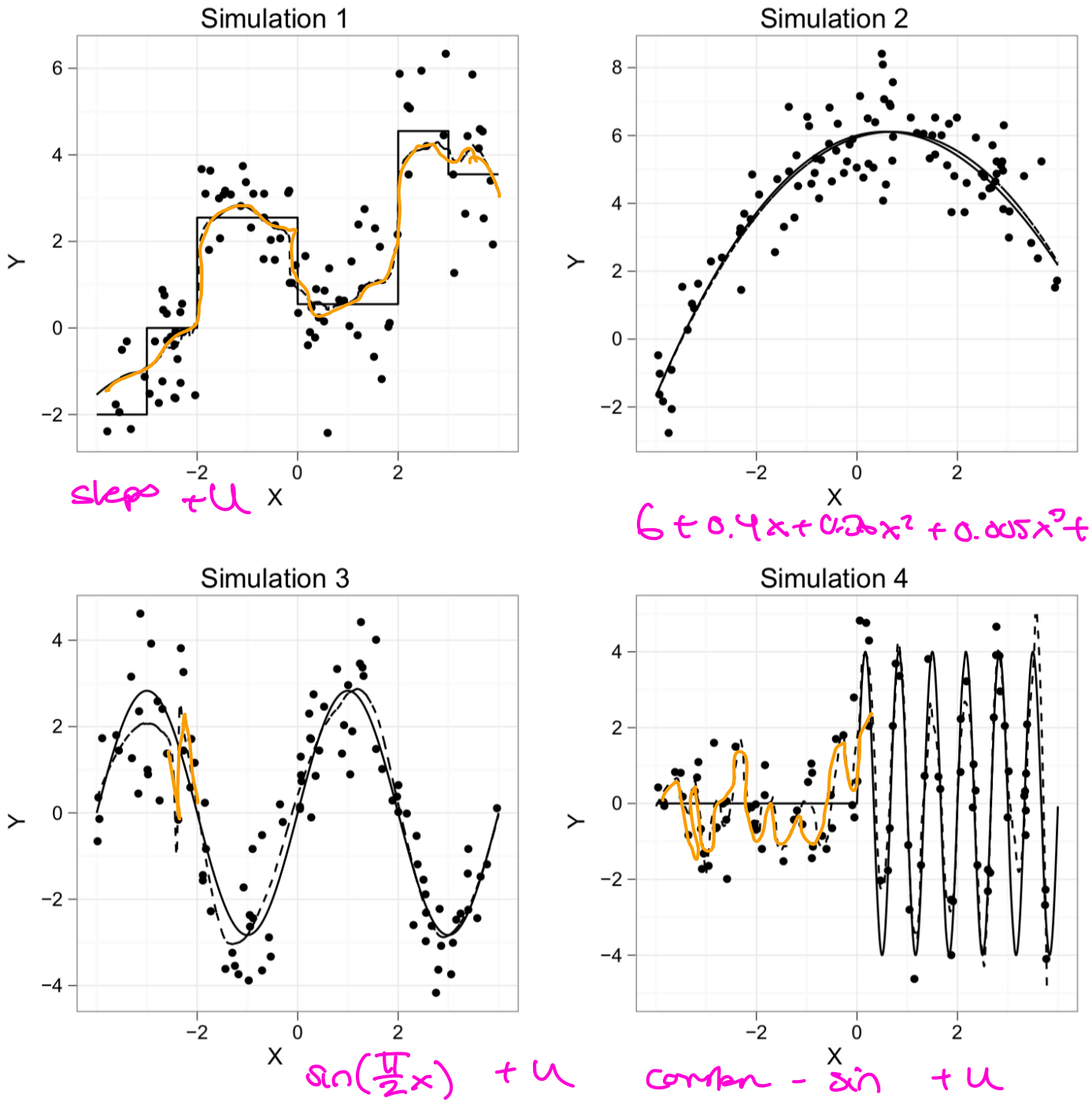


Fig. 3.3 Scatterplots of the four simulations. The *solid line* is the true relationship. The *points* represent one of the simulated data sets of size $n = 100$. The *dashed line* is the super learner fit for the shown data set

Polley 2011

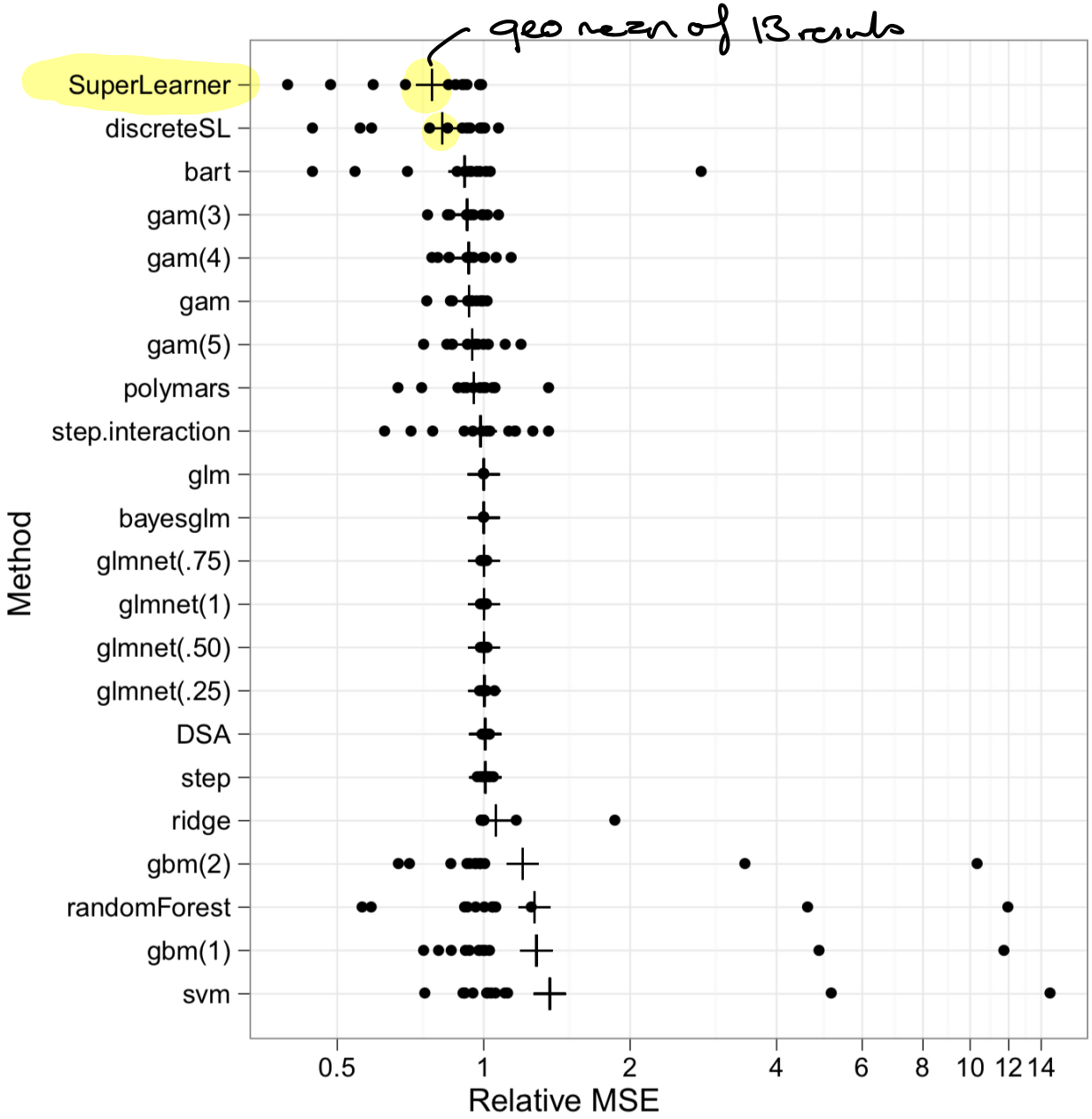
Table 3.2 Results for four simulations. Average R^2 based on 100 simulations and the corresponding standard errors

Algorithm	Sim 1		Sim 2		Sim 3		Sim 4	
	R^2	SE(R^2)	R^2	SE(R^2)	R^2	SE(R^2)	R^2	SE(R^2)
Super learner	0.741	0.032	0.754	0.025	0.760	0.025	0.496	0.122
Discrete SL	0.729	0.079	0.758	0.029	0.757	0.055	0.509	0.132
SL.glm	0.422	0.012	0.189	0.016	0.107	0.016	−0.018	0.021
SL.interaction	0.428	0.016	0.769	0.011	0.100	0.020	−0.018	0.029
SL.randomForest	0.715	0.021	0.702	0.027	0.724	0.018	0.460	0.109
SL.bagging(0.01)	0.751	0.022	0.722	0.036	0.723	0.018	0.091	0.054
SL.bagging(0.1)	0.635	0.120	0.455	0.195	0.661	0.029	0.020	0.025
SL.bagging(0.0)	0.752	0.021	0.722	0.034	0.727	0.017	0.102	0.060
SL.bagging(ms5)	0.747	0.020	0.727	0.030	0.741	0.016	0.369	0.104
SL.gam(2)	0.489	0.013	0.649	0.026	0.213	0.029	−0.014	0.023
SL.gam(3)	0.535	0.033	0.748	0.024	0.412	0.037	−0.017	0.029
SL.gam(4)	0.586	0.027	0.759	0.020	0.555	0.022	−0.020	0.034
SL.gbm	0.717	0.035	0.694	0.038	0.679	0.022	0.063	0.040
SL.nnet(2)	0.476	0.235	0.591	0.245	0.283	0.285	−0.008	0.030
SL.nnet(3)	0.700	0.096	0.700	0.136	0.652	0.218	0.009	0.035
SL.nnet(4)	0.719	0.077	0.730	0.062	0.738	0.102	0.032	0.052
SL.nnet(5)	0.705	0.079	0.716	0.070	0.731	0.077	0.042	0.060
SL.polymars	0.704	0.033	0.733	0.032	0.745	0.034	0.003	0.040
SL.bart	0.740	0.015	0.737	0.027	0.764	0.014	0.077	0.034
SL.loess(0.75)	0.599	0.023	0.761	0.019	0.487	0.028	−0.023	0.033
SL.loess(0.50)	0.695	0.018	0.754	0.022	0.744	0.029	−0.033	0.038
SL.loess(0.25)	0.729	0.016	0.738	0.025	0.772	0.015	−0.076	0.068
SL.loess(0.1)	0.690	0.044	0.680	0.064	0.699	0.039	0.544	0.118

Take home message:

- the best algo (baselearner) is not known in advance and will change = dependent on problem & data
- adding more base learners → superlearner does better and get $R^2 = 0.76$ optimaly

Fig. 3.4 Tenfold cross-validated relative mean squared error compared to glm across 13 real data sets. Sorted by geometric mean, denoted by the plus (+) sign



Name	n	p	Source
ais	202	10	Cook and Weisberg (1994)
diamond	308	17	Chu (2001)
cps78	550	18	Berndt (1991)
cps85	534	17	Berndt (1991)
cpu	209	6	Kibler et al. (1989)
FEV	654	4	Rosner (1999)
Pima	392	7	Newman et al. (1998)
laheart	200	10	Afifi and Azen (1979)
mussels	201	3	Cook (1998)
enroll	258	6	Liu and Stengos (1999)
fat	252	14	Penrose et al. (1985)
diabetes	366	15	Harrell (2001)
house	506	13	Newman et al. (1998)

Theoretical result

LeDell (2015) (page 6)

- Oracle selector: the estimator among all possible weighted combinations of the base prediction function that minimizes the risk under the *true data generating distribution*.
- The *oracle result* was established for the Super Learner by Laan, Polley, and Hubbard (2007) linear α's
- If the *true prediction function* cannot be represented by a combination of the base learners (available), then "optimal" will be the closest linear combination that would be optimal if the true data-generating function was known.
- The oracle result require an *uniformly bounded loss function*. Using the convex restriction (sum alphas =1) implies that if each based learner is bounded so is the convex combination. In practice: truncation of the predicted values to the range of the outcome in the training set is sufficient to allow for unbounded loss functions

Other issues

- Many different implementations available, and much work on parallell processing and speed and memory efficient execution.
- Super Learner implicitly can handle hyperparameter tuning by including the same base learner with different model parameter sets in the ensemble.
- Speed and memory improvements for large data sets involves subsampling, and the R `subsemble` package is one solution, the H2o package another.

Super Learner Algorithm

H2O help page

The steps below describe the individual tasks involved in training and testing a Super Learner ensemble. H2O automates most of the steps below so that you can quickly and easily build ensembles of H2O models.

1. Set up the ensemble.
    a. Specify a list of L base algorithms (with a specific set of model parameters).
    b. Specify a metalearning algorithm.

2. Train the ensemble.
    a. Train each of the L base algorithms on the training set.
    b. Perform k-fold cross-validation on each of these learners and collect the cross-validated predicted values from each of the L algorithms.
    c. The N cross-validated predicted values from each of the L algorithms can be combined to form a new N x L matrix. This matrix, along with the original response vector, is called the "level-one" data. (N = number of rows in the training set.)
    d. Train the metalearning algorithm on the level-one data. The "ensemble model" consists of the L base learning models and the metalearning model, which can then be used to generate predictions on a test set.

3. Predict on new data.
    a. To generate ensemble predictions, first generate predictions from the base learners.
    b. Feed those predictions into the metalearner to generate the ensemble prediction.

- metalearner_algorithm (Optional) Specify the metalearner algorithm type. Options include:

  - `"AUTO"` (GLM with non negative weights & standardization turned off, and if `validation_frame` is present, then `lambda_search` is set to True; may change over time). This is the default.
  - `"glm"` (GLM with default parameters)
  - `"gbm"` (GBM with default parameters)
  - `"drf"` (Random Forest with default parameters)
  - `"deeplearning"` (Deep Learning with default parameters)
  - `"naivebayes"` (NaiveBayes with default parameters)
  - `"xgboost"` (if available, XGBoost with default parameters)

- metalearner_params: (Optional) If a `metalearner_algorithm` is specified, then you can also specify a list of customized parameters for that algorithm (for example, a GBM with `ntrees=100`, `max_depth=10`, etc.)
- metalearner_nfolds: (Optional) Specify the number of folds for cross-validation of the metalearning algorithm. Defaults to 0 (no cross-validation). If you want to compare the cross-validated performance of the ensemble model to the cross-validated performance of the base learners or other algorithms, you should make use of this option.
- metalearner_fold_assignment: (Optional; Applicable only if a value for `metalearner_nfolds` is specified) Specify the cross-validation fold assignment scheme for the metalearner. The available options are AUTO (which is Random), Random, Modulo, or Stratified (which will stratify the folds based on the response variable for classification problems). This value defaults to AUTO.
- metalearner_fold_column: (Optional; Cannot be used at the same time as `nfolds`) Specify the name of the column that contains the cross-validation fold assignment per observation for cross-validation of the metalearner. The column can be numeric (e.g. fold index or other integer value) or it can be categorical. The number of folds is equal to the number of unique values in this column.
- metalearner_transform: (Optional) Specify the transformation used on predictions from the base models in order to make a level one frame. Options include:

  - `"NONE"` (no transform applied)
  - `"Logit"` (applicable only to classification tasks, use logit transformation on the predicted probabilities)

R example from H2o-package

https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html

Python examples available from the same page

The Higgs boson data is used - but which version is not specified, maybe this https://archive.ics.uci.edu/ml/datasets/HIGGS or a specifically made data set. The problem is binary, so maybe to detect signal vs noise.

```
h2o.init()
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:          1 days 3 hours
    H2O cluster timezone:        Europe/Oslo
    H2O data parsing timezone:   UTC
    H2O cluster version:         3.40.0.1
    H2O cluster version age:     25 days
    H2O cluster name:            H2O_started_from_R_mettela_bze126
    H2O cluster total nodes:     1
```

```r
# Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)

# For binary classification, response should be a factor
train[, y] <- as.factor(train[, y])
test[, y] <- as.factor(test[, y])

print(dim(train))
```

```
[1] 10000    29
```

```r
# Number of CV folds (to generate level-one data for stacking)
nfolds <- 5

# There are a few ways to assemble a list of models to stack toegether:
# 1. Train individual models and put them in a list


# 1. Generate a 2-model ensemble (GBM + RF)

# Train & Cross-validate a GBM
my_gbm <- h2o.gbm(x = x,
                  y = y,
                  training_frame = train,
                  distribution = "bernoulli",
                  ntrees = 10,
                  max_depth = 3,
                  min_rows = 2,
                  learn_rate = 0.2,
                  nfolds = nfolds,
                  keep_cross_validation_predictions = TRUE,
                  seed = 1)
```

```
# Train & Cross-validate a RF
my_rf <- h2o.randomForest(x = x,
                          y = y,
                          training_frame = train,
                          ntrees = 50,
                          nfolds = nfolds,
```


```
                          keep_cross_validation_predictions = TRUE,
                          seed = 1)
```

Now the default metalearner

Default metalearner: Options include 'AUTO' (GLM with non negative weights; if validation_frame is present, a lambda search is performed)

```
# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                base_models = list(my_gbm, my_rf))
```

```
  |
  |
  |
  |=======================================================================
```

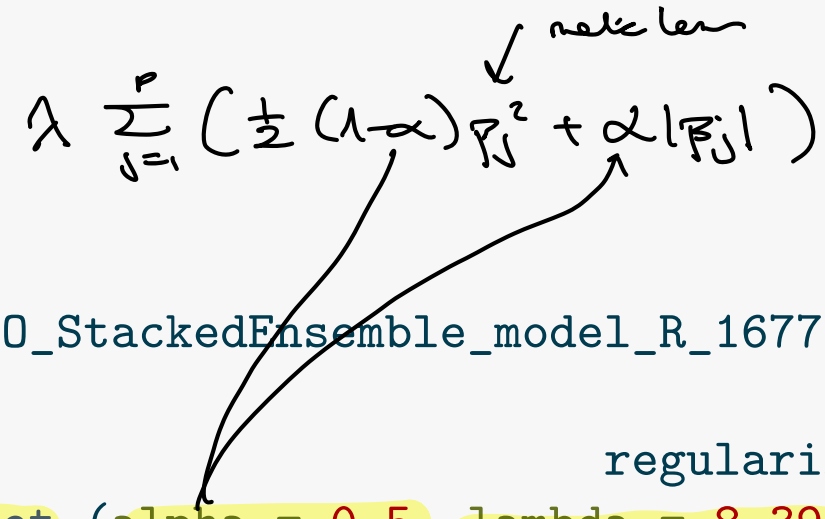
```
# default metalearner_transform should be NONE
#print(summary(ensemble))
#ensemble@model
# Eval ensemble performance on a test set
perf <- h2o.performance(ensemble, newdata = test)

# Compare to base learner performance on the test set
perf_gbm_test <- h2o.performance(my_gbm, newdata = test)
perf_rf_test <- h2o.performance(my_rf, newdata = test)
baselearner_best_auc_test <- max(h2o.auc(perf_gbm_test), h2o.auc(perf_r
```

```
metalearner_model
Model Details:
==============

H2OBinomialModel: glm
Model ID:  metalearner_AUTO_StackedEnsemble_model_R_1677945156774_1824
GLM Model: summary
    family  link                                regularization number_o
1 binomial logit Elastic Net (alpha = 0.5, lambda = 8.399E-5 )
                                                   training_frame
1 levelone_training_StackedEnsemble_model_R_1677945156774_1824

Coefficients: glm coefficients
                         names coefficients standardized_coefficients
1                    Intercept    -3.603549                  0.149102
2 GBM_model_R_1677945156774_1086     3.298011                  0.493334
3 DRF_model_R_1677945156774_1214     3.809905                  0.701246
```



$$P(Y=1) = \frac{e^{\hat{\eta}}}{1+e^{\hat{\eta}}} \quad \text{where } \hat{\eta}(x) = -3.6 + 3.25 \cdot \hat{\psi}_{GBM}(x) + 3.8 \cdot \hat{\psi}_{DRF}(x)$$

Adding transform “logit”

```
# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                base_models = list(my_gbm, my_rf),
                                metalearner_transform = "Logit")
```

```
  |
  |
  |
  |=======================================================================
```

```
#print(summary(ensemble))
#print(ensemble@model)

# Eval ensemble performance on a test set
perf <- h2o.performance(ensemble, newdata = test)

# Compare to base learner performance on the test set
perf_gbm_test <- h2o.performance(my_gbm, newdata = test)
perf_rf_test <- h2o.performance(my_rf, newdata = test)
baselearner_best_auc_test <- max(h2o.auc(perf_gbm_test), h2o.auc(perf_r
ensemble_auc_test <- h2o.auc(perf)
```

```
$metalearner_model
Model Details:
==============

H2OBinomialModel: glm
Model ID:  metalearner_AUTO_StackedEnsemble_model_R_1677945156774_1830
GLM Model: summary
    family  link                                regularization number_o
1 binomial logit Elastic Net (alpha = 0.5, lambda = 3.885E-4 )
                                               training_frame
1 levelone_training_StackedEnsemble_model_R_1677945156774_1830

Coefficients: glm coefficients
                          names coefficients standardized_coefficients
1                     Intercept    -0.053725                  0.154528
2 GBM_model_R_1677945156774_1086     0.791767                  0.515081
3 DRF_model_R_1677945156774_1214     0.845217                  0.731991
```

$$\hat{\eta}(x) = -0.05 + 0.79 \cdot \text{logit}(\hat{\psi}_{GBM}(x)) + 0.85 \cdot \text{logit}(\hat{\psi}_{DRF}(x))$$

```
[1] "Best Base-learner Test AUC:  0.769204725074508"
```

```
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
```

```
[1] "Ensemble Test AUC:  0.773144298176816"
```

```
# [1] "Best Base-learner Test AUC:  0.76979821502548"
# [1] "Ensemble Test AUC:  0.773501212640419"
```

```
[1] "Best Base-learner Test AUC:  0.769204725074508"
```

```
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
```

```
[1] "Ensemble Test AUC:  0.773096033881535"
```

```
# Generate predictions on a test set (if neccessary)
pred <- h2o.predict(ensemble, newdata = test)
```

```
|
|
|
|======================================================================
```

```
print(head(pred))
```

```
  predict        p0        p1
1       0 0.6739209 0.3260791
2       1 0.5814741 0.4185259
3       1 0.5826643 0.4173357
4       1 0.1971804 0.8028196
5       1 0.4561659 0.5438341
6       1 0.3365841 0.6634159
```

Uncertainty in the ensemble

(Class notes: Study "Road map" 2 from Polley, Rose, and Laan (2011))

- Add an outer (external) cross validation loop (where the super learner loop is inside). Suggestion: use 20-fold, especially when small sample size.
- Overfitting? Check if the super learner does as well or better than any of the base learners in the ensemble.
- Results using *influence functions* for estimation of the variance for the Super Learner are based on asymptotic variances in the use of V-fold cross-validation (see Ch 5.3 of LeDell (2015))

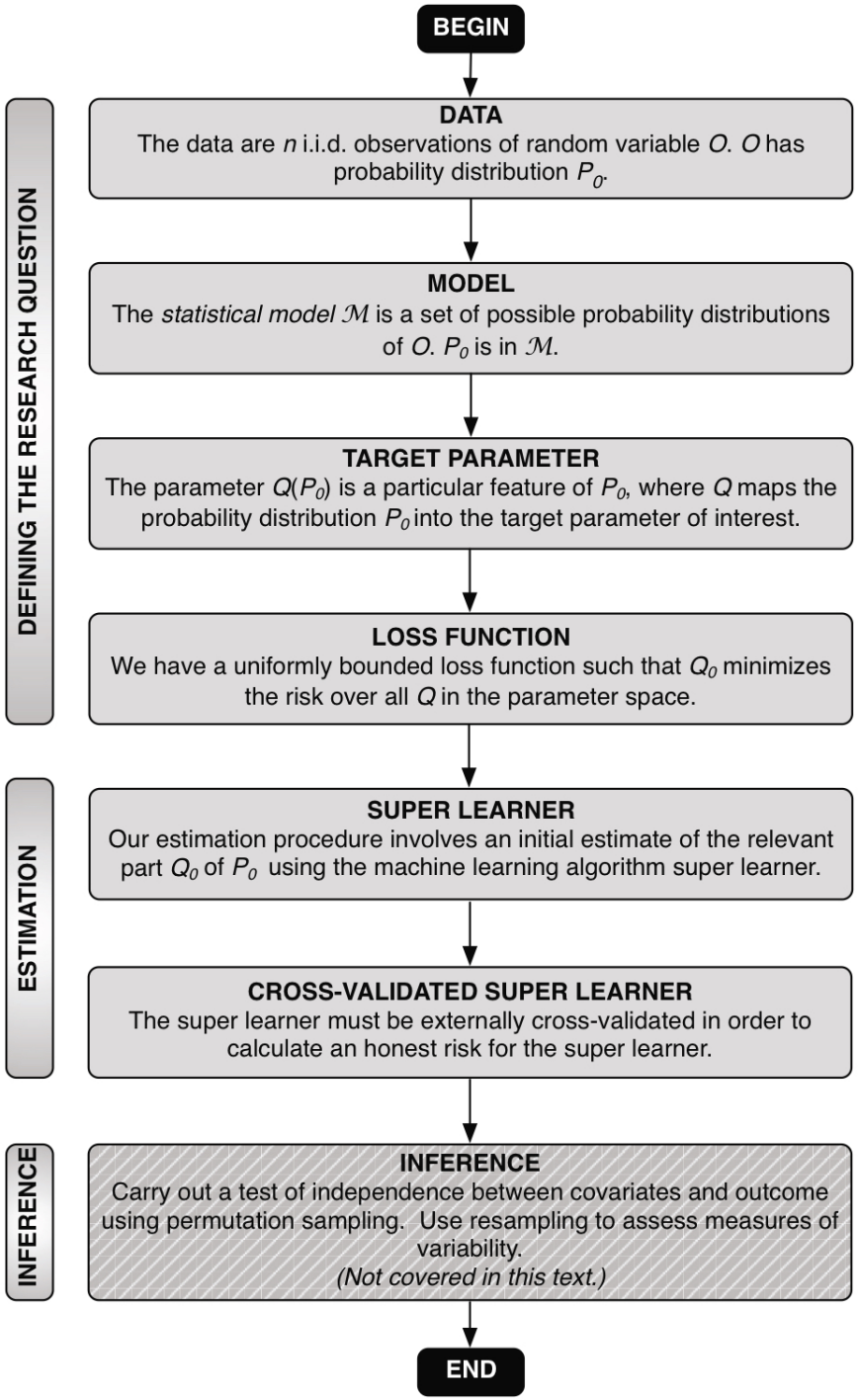


Fig. 3.6 Road map for prediction

Ensembles - overview

(ELS Ch 16.1)

With ensembles we want to build *one prediction model* which combines the strength of *a collection of models*.

These models may be simple base models - or more elaborate models.

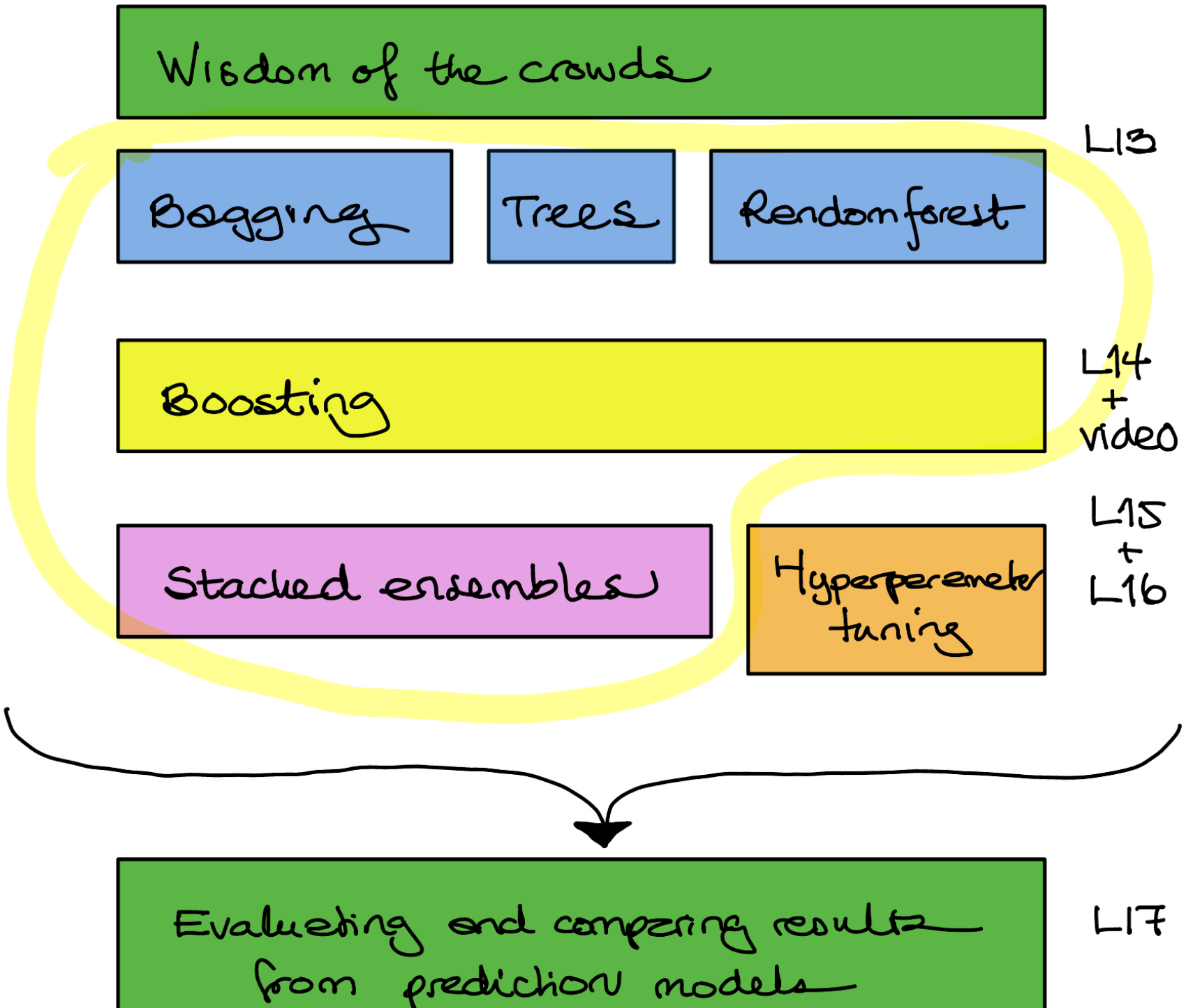
We have studied bagging - where we use the bootstrap to repeatedly fit a statistical model, and then take a simple average of the predictions (or majority vote). Here the base models can be trees - or other type of models.

Random forest is a version of bagging with trees, with trees made to be different (decorrelated).

We have studied boosting, where the models are trained on sequentially different data - from residuals or gradients of loss functions - and the ensemble members cast weighted votes (downweighted by a learning rate). We have observed that there are many hyperparameters that need to be to tuned to optimize performance.

And today - we have learned about the stacked ensemble

Before we ~~start~~ END



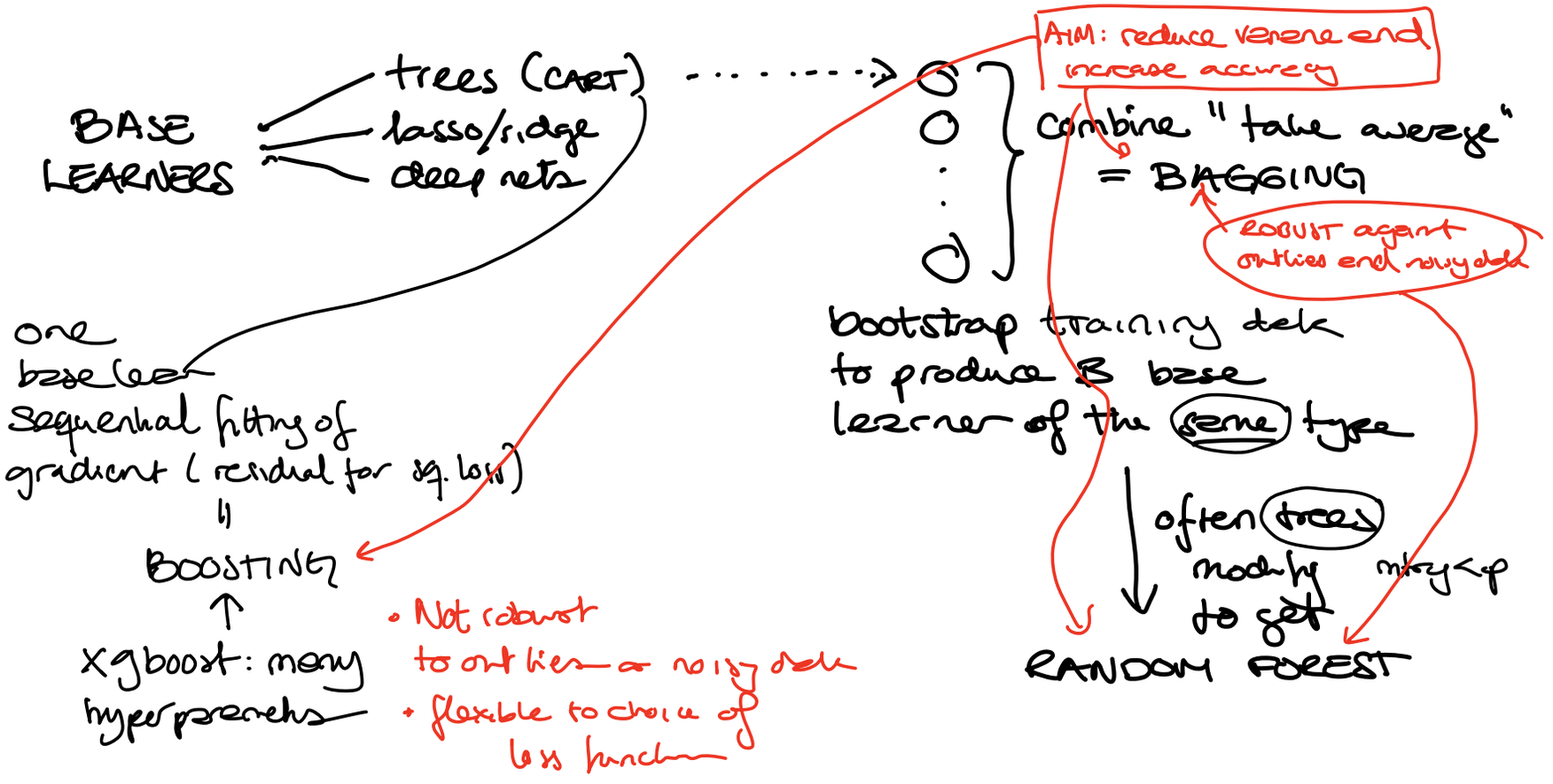
Group work:

draw a concept map / mind map to (verbs)

sum up what we know about

ensemble methods!

Wisdom of crowds ← vote ← diverse & independent "bodies"



Stacking: simple + more complex base learners → boost to have a diverse set

- asymptotic oracle properties for lincom in meta learner
- can be used to "avoid hyperparameter tuning"

- avoid model selection - no waste