IS 733 Lesson 9

Ensemble Methods

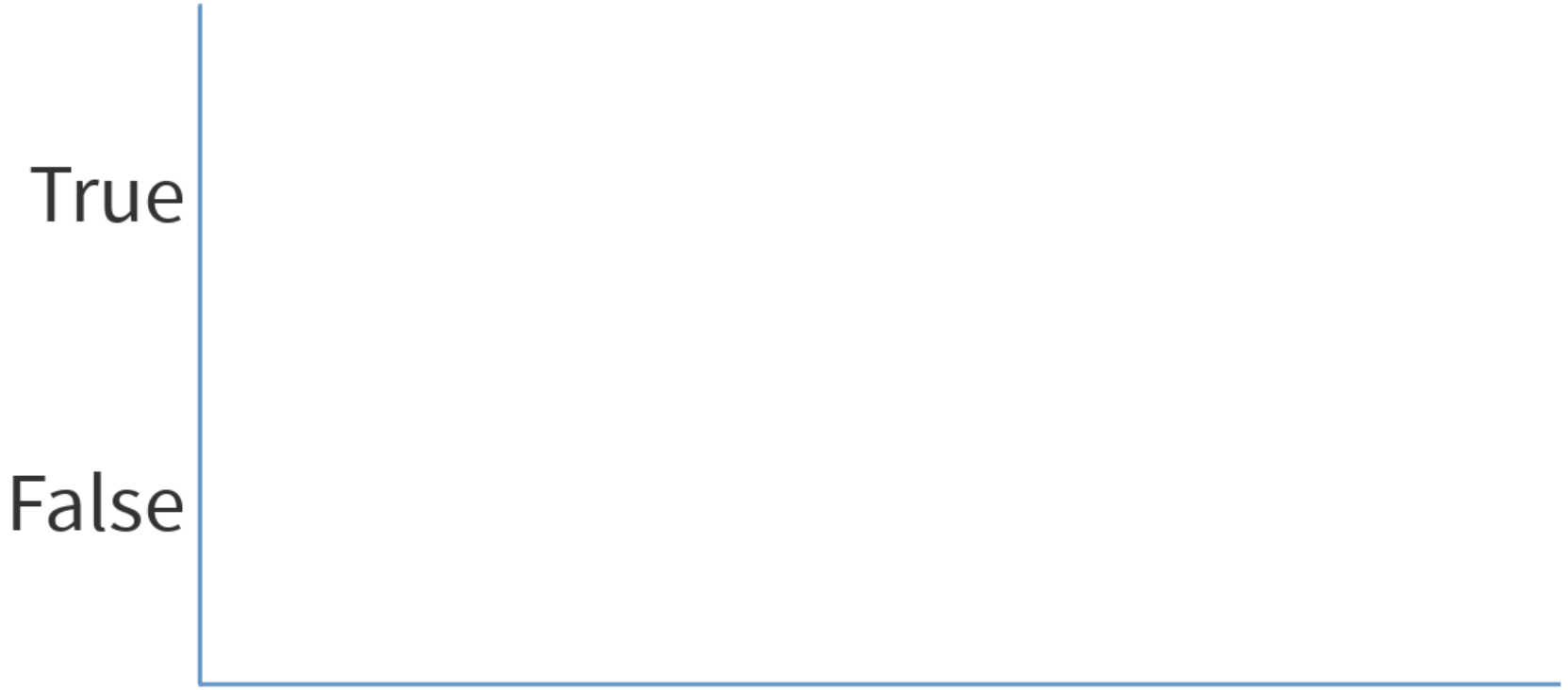
Slides based on those from Data Mining by I. H. Witten, E. Frank, M. A. Hall and C. J. Pal and James Foulds' Lecture Notes

Announcements

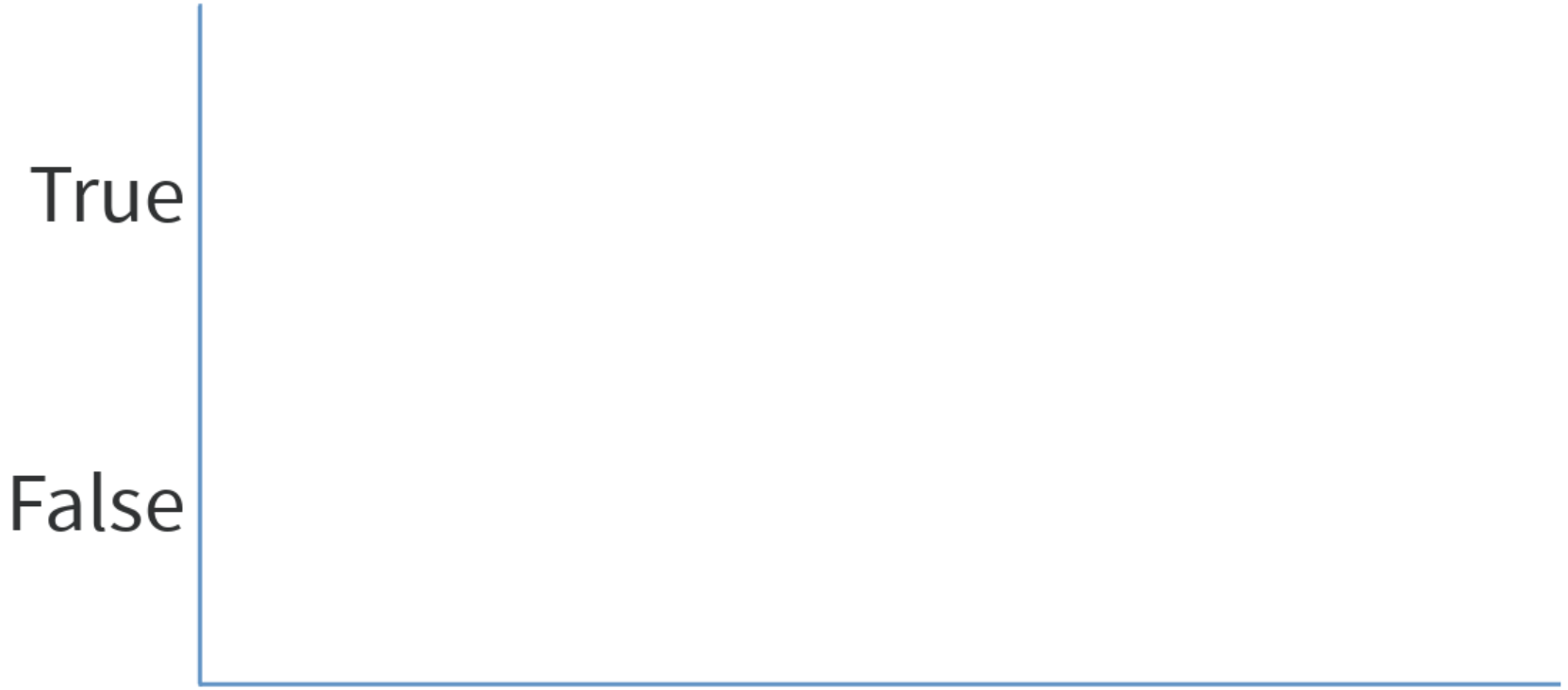
- **Project mid-term progress report** is due today on Blackboard. One of the group member can upload this.

- **Homework 4** is posted at the course webpage and due 4/20/2021 on Blackboard

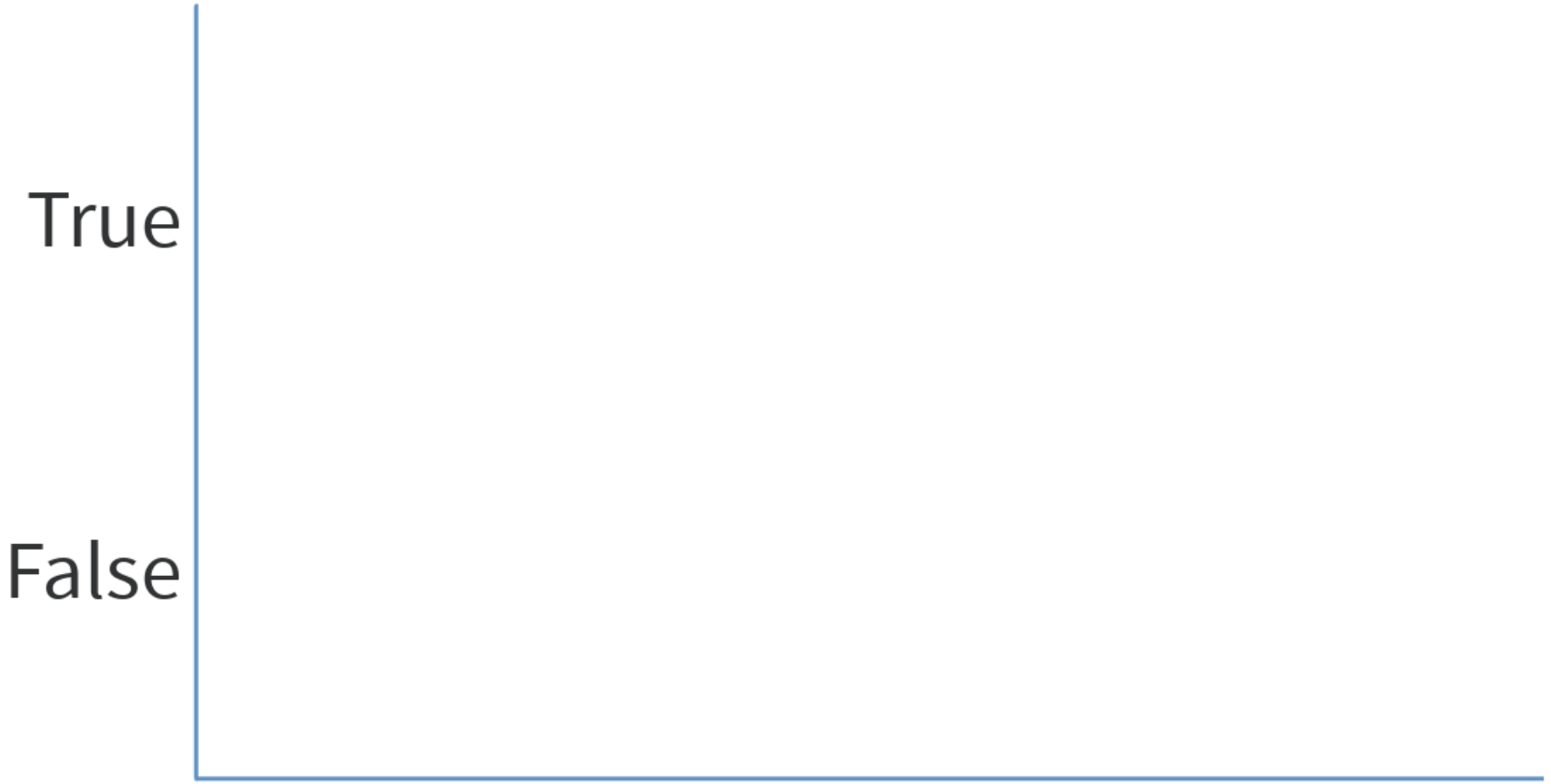
Ensemble methods rarely perform better than the best of the base classifiers that are combined into a single model.



To get good performance with ensemble methods, all of the base classifiers must individually have excellent performance



Randomized algorithms can be useful for building ensembles



Learning outcomes

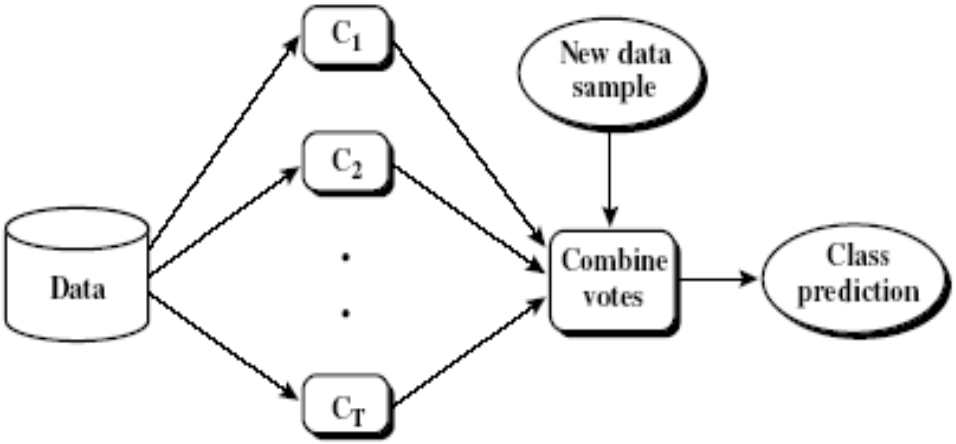
By the end of the lesson, you should be able to:

- **Describe,** and **compare and contrast** the main ensemble methods:
  bagging, **boosting**, **random forests**, **stacking**

- **Discuss** why these methods may improve classification performance

- **Explain**, at an intuitive level, the **bias-variance trade-off**

Ensemble Methods



- **Ensemble methods**
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*

- Popular ensemble methods
  - **Bagging**: averaging the prediction over a collection of classifiers
  - **Boosting**: weighted vote with a collection of classifiers
  - **Stacking**: combining a set of heterogeneous classifiers

Combining multiple models

- Basic idea:
  build different “experts”, let them vote
- Advantage:
  - often improves predictive performance
- Disadvantage:
  - usually produces output that is very hard to analyze
  - but: there are approaches that aim to produce a single comprehensible structure

Bagging

- **Combining predictions by voting/averaging**
  - Each model receives **equal weight**
- *"Idealized" version*:
  - Sample several training sets of size *n*
    (instead of just having one training set of size *n*)
  - Build a classifier for each training set
  - Combine the classifiers' predictions
- If learning scheme is *unstable,*
  bagging almost always improves performance
  - **Unstable learner:** small change in training data can make big change in model (e.g., when learning decision trees)

Bias-variance decomposition

- The ***bias-variance decomposition*** is used to analyze how much restriction to a single training set affects performance

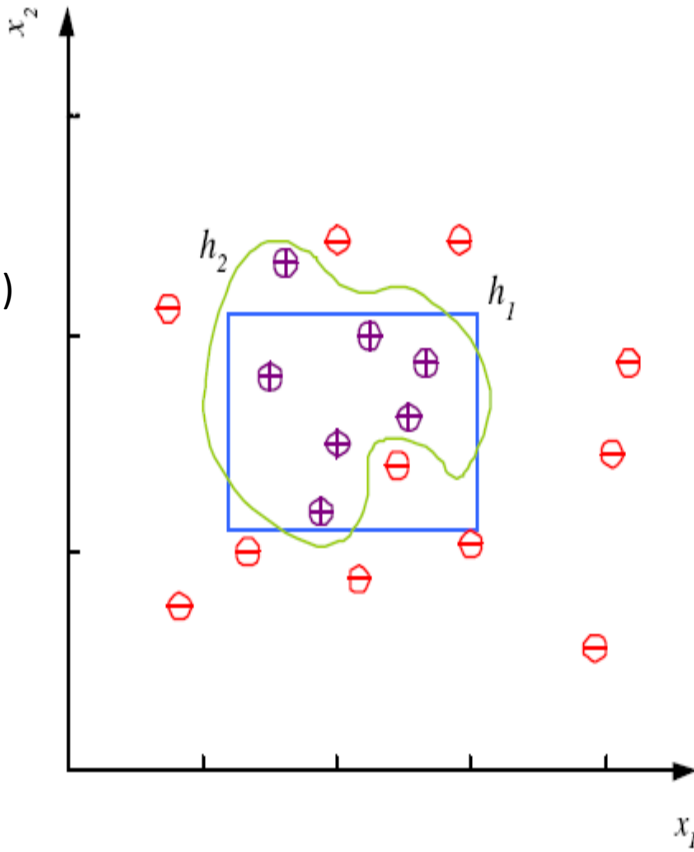
- Assume we have the idealized ensemble classifier discussed on the previous slide

- We can decompose the expected error of any individual ensemble member as follows:
  - *Bias* = expected error of the ensemble classifier on new data
  - *Variance* = component of the expected error due to the particular training set being used to build our classifier
  - Total expected error = bias + variance

Noise and Model Complexity

Use the simpler one because

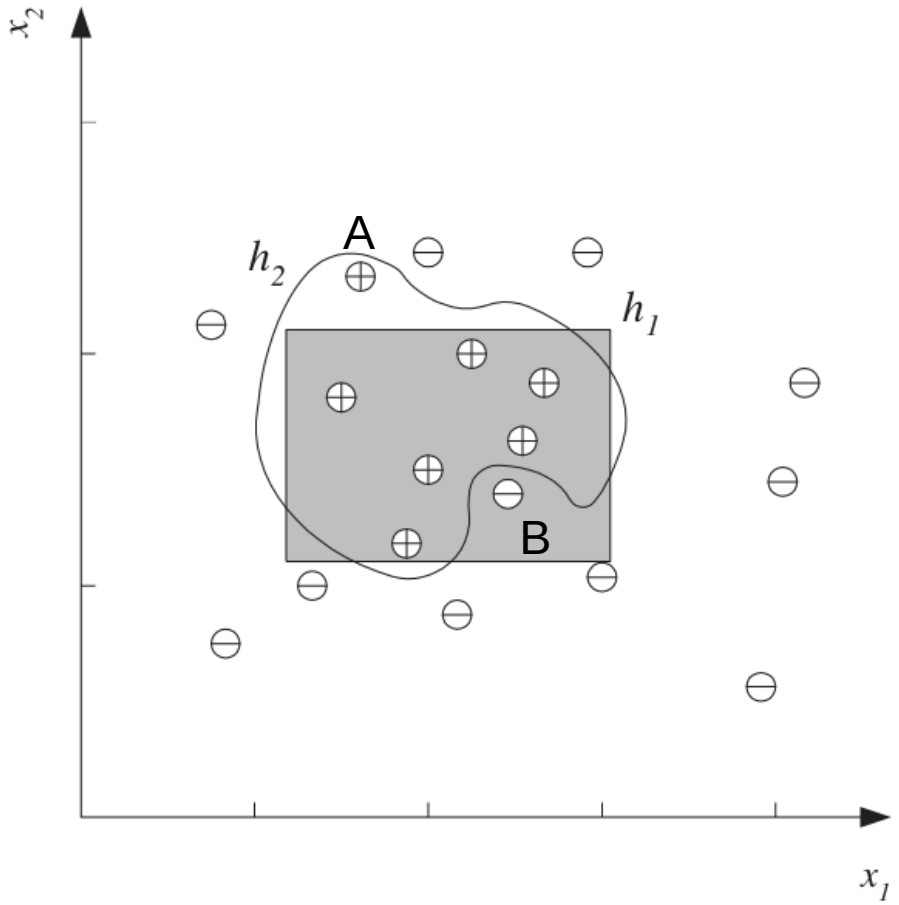
- Simpler to use (lower computational complexity)
  - Easier to check if a point is inside/outside a rectangle
- Easier to train (lower space complexity)
  - Fewer parameters
  - More bias (rigid: more likely will not change its hypothesis)
  - Less variance (less ability of learner to change its hypothesis)
  - May fail if indeed the underlying class is not that simple
- Easier to explain (more interpretable)
  - Defining intervals on the two attributes
- Generalizes better (lower variance - Occam's razor)
  - Better discriminator than wiggly shape in presence of mislabel/noise though with higher error
  - Simpler explanations are more plausible



Noise

- Sources
  - Incorrect feature values
    - Imprecision in recoding the input attributes
  - Incorrect class labels
    - teacher noise
  - Hidden or latent features
    - additional attributes
- Impact
  - Overfitting: Trying too hard to fit h to the noise

Underfitting vs. Overfitting



If A and B are noise, then h_2 overfits.

If A and B are *not* noise, then h_1 underfits.

Bias vs. Variance

- Bias: Likelihood a learner will not change its hypothesis
- Variance: Ability of learner to change its hypothesis
- Simple models have high bias, low variance
- Complex models have low bias, high variance
- Want balanced tradeoff
- Depends on hypothesis class
  - Rectangles vs. arbitrary shape
- Occam's Razor: Prefer simpler models

Inductive Bias

- Given a training set X, there are many models that are consistent with X
- Preferring one of these models over another is an "inductive bias"
- For example
  - Preferring rectangles to arbitrary shapes
  - Preferring rectangle with largest margin
  - Preferring lower-degree polynomial
  - Preferring polynomial minimizing squared error
- How do we choose the right inductive bias?

Intuition: Bias-Variance Trade-off

- **Inflexible models may not be able to fit the data well** (underfitting), if the model does not match the true concept. "**High bias**"
  - Poor performance on training and test sets

- **Overly flexible models may fit the data too well, including the noise**, and fail to generalize to new data. "**High variance**"
  - Good performance on training data, poor on test

- Altering the model's flexibility **trades** bias and variance
- Rule of thumb: *generative models (e.g. naïve Bayes) have high bias, discriminative models (e.g. logistic regression) have high variance*

Bias-Variance Trade-off

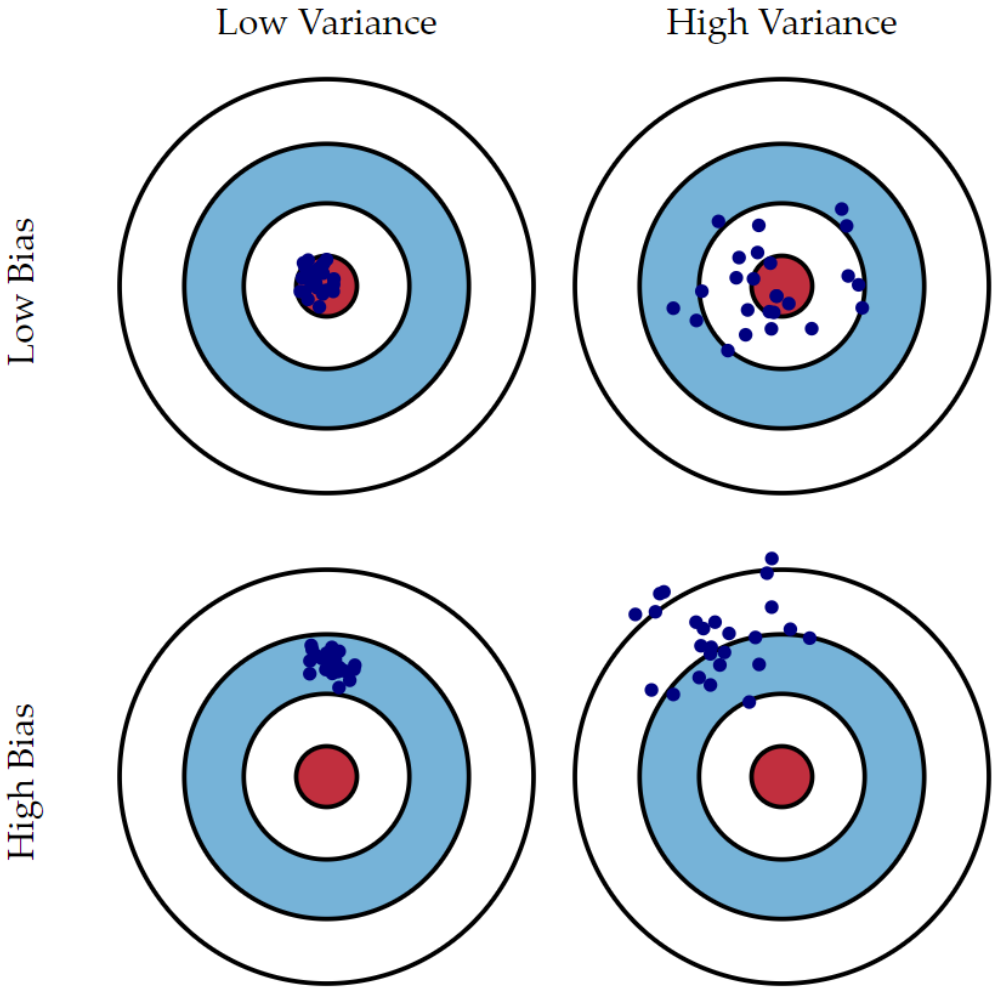


Figure due to Scott Fortmann-Roe, http://scott.fortmann-roe.com/docs/BiasVariance.html

Bias-Variance Trade-off

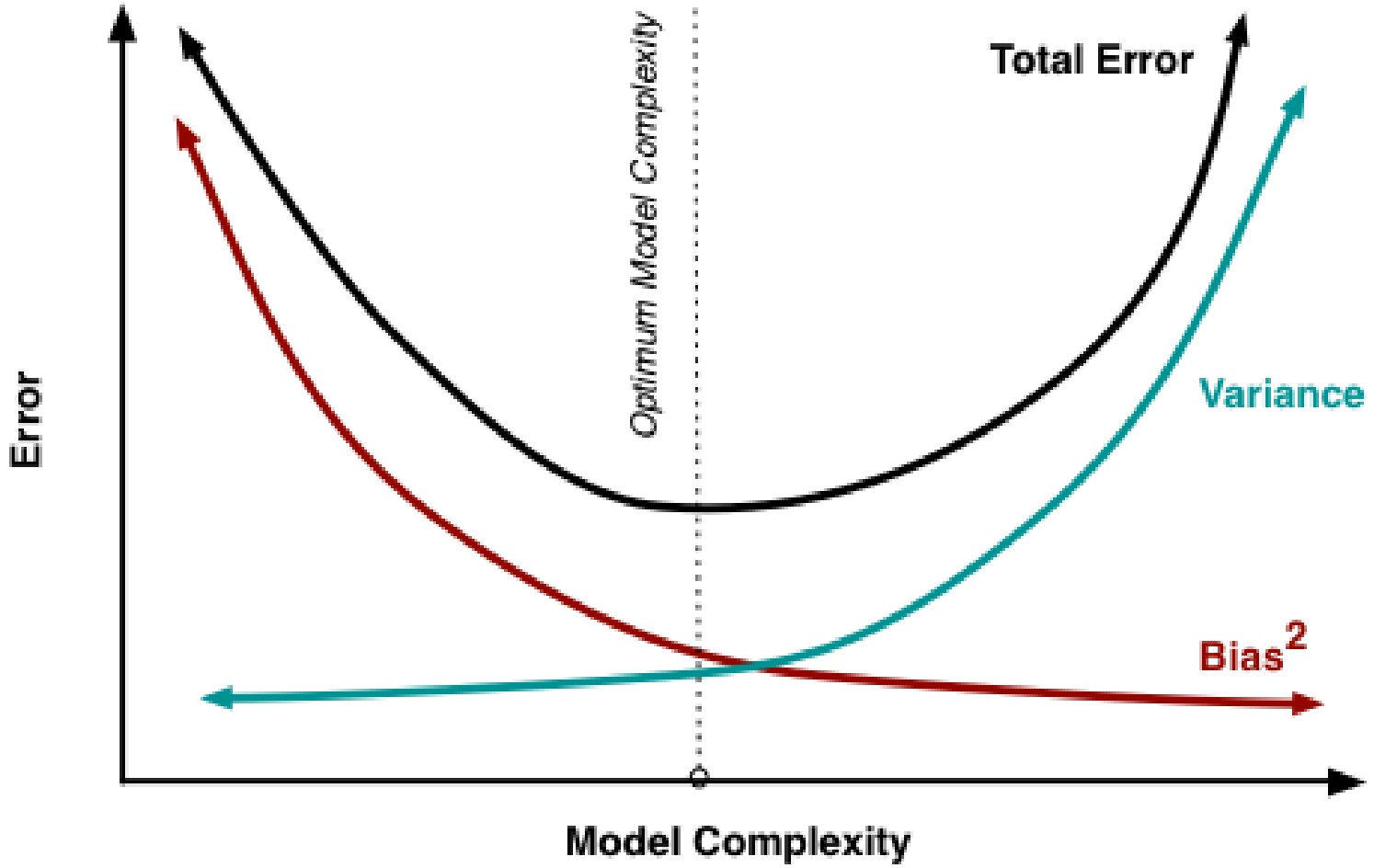


Figure due to Scott Fortmann-Roe, http://scott.fortmann-roe.com/docs/BiasVariance.html

Think-Pair-Share: Bias-Variance Trade-off

- *Which **hyper-parameters** would you alter* to modify the **model complexity**, and hence the **bias-variance trade-off**, of the following types of classifiers:
  1. Decision tree learner
  2. Decision rule learner
  3. Support vector machine
  4. Deep neural network

- Design an **experimental evaluation** to verify whether an appropriate level of bias vs variance has been achieved when varying the above hyper-parameters

Recap: “Idealized version” of Bagging

- **Combining predictions by voting/averaging**
  - Each model receives **equal weight**
- *“Idealized” version*:
  - Sample several training sets of size *n*
    (instead of just having one training set of size *n*)
  - Build a classifier for each training set
  - Combine the classifiers’ predictions

More on bagging

- The idealized version of bagging improves performance because it eliminates the *variance* component of the error

- **Problem:** we only have one dataset!
  - We can't actually implement "idealized bagging" in practice

More on bagging

- **Solution:** generate new datasets of size *n* by **sampling from the original dataset *with replacement***
- This is what ***bagging*** algorithm does
- Even though the datasets are all dependent, bagging often reduces variance, and, thus, error
  - Can be applied to numeric prediction and classification
  - Can help a lot if the data is noisy
  - Usually, the more classifiers the better, with diminishing returns

Bagging classifiers

Model generation

```
Let n be the number of instances in the training data
For each of t iterations:
Sample n instances from training set
   (with replacement)
Apply learning algorithm to the sample
Store resulting model
```

Classification

```
For each of the t models:
Predict class of instance using model
Return class that is predicted most often
```

In the datasets constructed by the bagging algorithm, some instances could get _____.

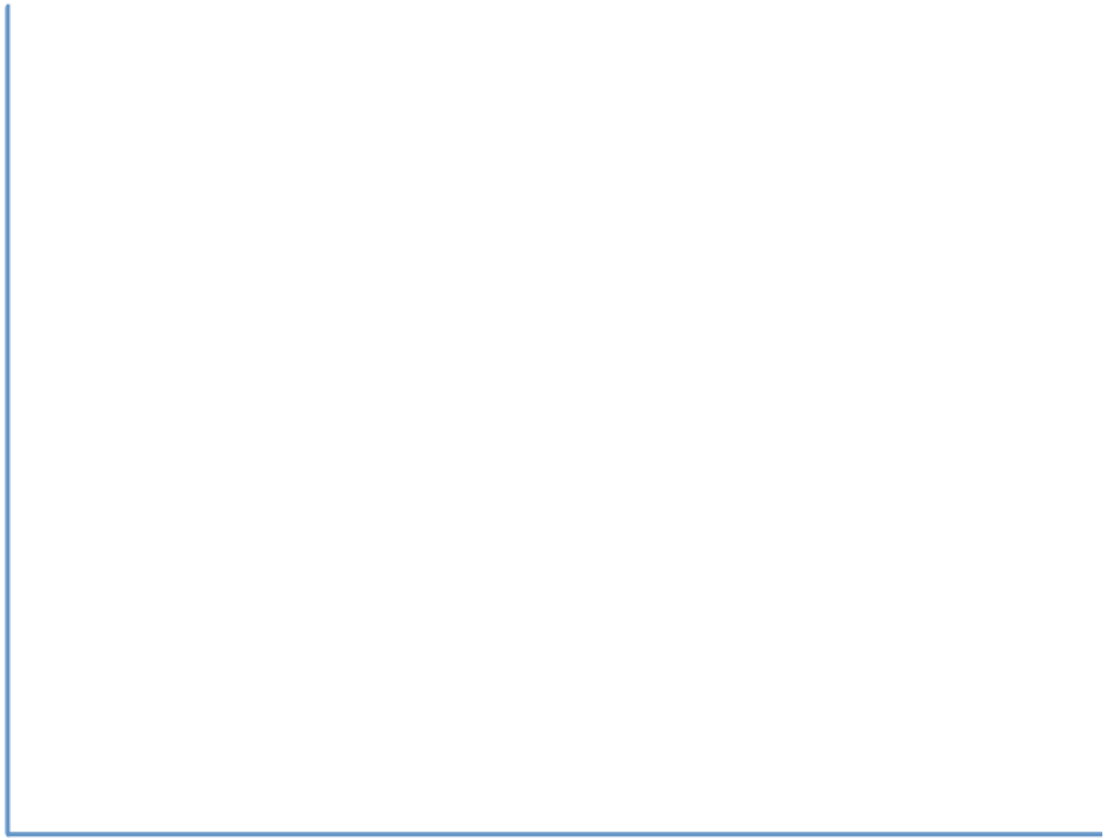
More attributes

Less attributes

Deleted

Reweighted

Normalized



Randomization and random forests

- Can *randomize the learning algorithm*, instead of input
- Some algorithms already have a random component: e.g., initial weights in a neural net
- Most algorithms can be randomized, e.g., greedy algorithms:
  - Pick *N* options at random from the full set of options, then choose the best of those *N* choices
  - E.g.: attribute selection in decision trees

Randomization and random forests

- Randomization is very generally applicable: e.g., we can use random subsets in a nearest-neighbor classifier
  - Bagging does not work well with stable classifiers such as nearest neighbour classifiers
  - Randomization could be used instead
- Randomization can be used to create a diverse ensemble of classifiers, as an alternative to bagging
  - When using decision trees, this yields the famous ***random forest*** method for building ensemble classifiers

Random Forest (Breiman 2001)

- **Random Forest**:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
  - During classification, each tree votes and the most popular class is returned

- Comparable in accuracy to Adaboost, but more robust to errors and outliers

- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

- Random forests are a reliable way to get decent performance on many classification problems. I recommend that you should at least try them out

Random Forest (Breiman 2001)

- Two Methods to construct Random Forest:
  - **Forest-RI (*random input selection*):** Randomly select, at each node, F attributes as candidates for the split at the node.
    - Only split on one of the candidates
  - **Forest-RC (*random linear combinations*):** Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)

Boosting

- Bagging can easily be parallelized because ensemble members are created independently

- Boosting is an alternative approach which is trained sequentially instead of in parallel
  - Later classifiers in the ensemble are informed by earlier ones

- Also uses voting/averaging
  - But: weights models according to performance

Boosting

- Boosting is Iterative: new models are influenced by performance of previously built ones
  - Encourage new model to become an “expert” for instances misclassified by earlier models
  - Intuitive justification: models should be **experts** that **complement each other**
- Many variants of boosting exist. We will focus on AdaBoost.M1

Boosting using AdaBoost.M1

Model generation

```
Assign equal weight to each training instance
For t iterations:
  Apply learning algorithm to weighted dataset,
  store resulting model
  Compute model's error e on weighted dataset
  If e = 0 or e ≥ 0.5:
    Terminate model generation
  For each instance in dataset:
    If classified correctly by model:
       Multiply instance's weight by e/(1-e)
  Normalize weight of all instances
```

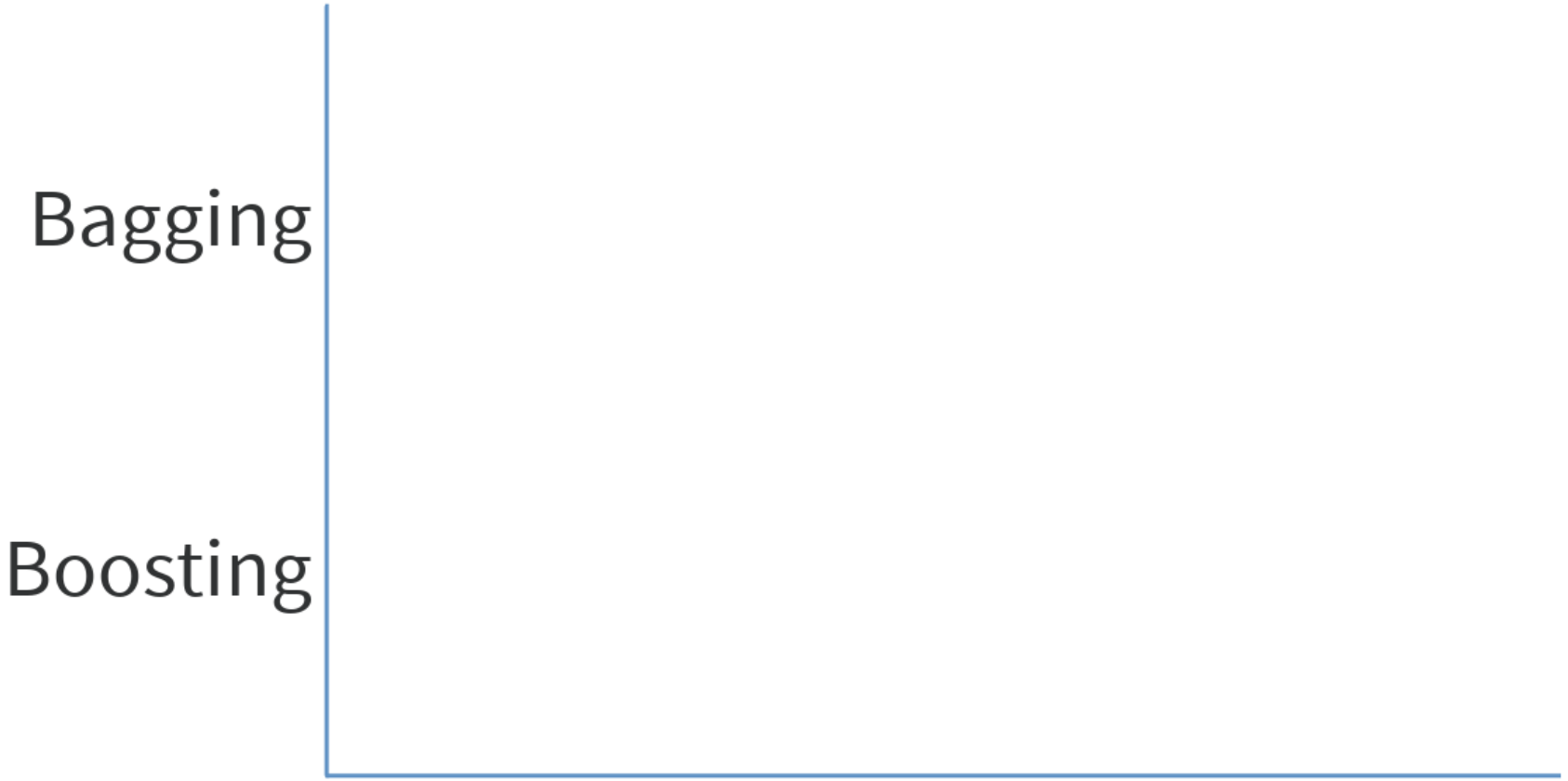
Classification

```
Assign weight = 0 to all classes
For each of the t (or less) models:
For the class this model predicts
  add -log e/(1-e) to this class's weight
Return class with highest weight
```

Comments on AdaBoost.M1

- Boosting needs weights … but
- can adapt learning algorithm ... or
- can apply boosting *without* weights:
  - Resample data with probability determined by weights
  - Disadvantage: not all instances are used
  - Advantage: if error > 0.5, can resample again
- The AdaBoost.M1 boosting algorithm stems from work in *computational learning theory*
- Theoretical result:
  - Training error decreases exponentially as iterations are performed
- Other theoretical results:
  - Works well if base classifiers are not too complex and
  - their error does not become too large too quickly as more iterations are performed

Which algorithm is more vulnerable to overfitting?



More comments on boosting

- Continue boosting after training error = 0?

- Puzzling fact: generalization error continues to decrease!
  - Seems to contradict Occam's Razor

- Possible explanation:
  consider *margin* (confidence), not just error
  - A possible definition of *margin*: difference between estimated probability for true class and nearest other class (between –1 and 1)
  - Margin continues to increase with more iterations

More comments on boosting

- AdaBoost.M1 works well with so-called *weak* learners; only condition: error does not exceed 0.5
  - Example of weak learner: decision stump

- In practice, boosting sometimes overfits if too many iterations are performed (in contrast to bagging)

Stacking

- Question: how to build a *heterogeneous* ensemble consisting of different types of models (e.g., decision tree and neural network)
  - Problem: models can be vastly different in accuracy

- Idea: to combine predictions of base learners, do *not* just vote, instead, use *meta learner*
  - In stacking, the base learners are also called *level-0 models*
  - Meta learner is called *level-1 model*
  - Predictions of base learners are input to meta learner

- Base learners are usually different learning schemes

Suppose we perform stacking, using the predictions of the level 0 models on the training data as attributes for training the level 1 model. This might not be a good idea due to

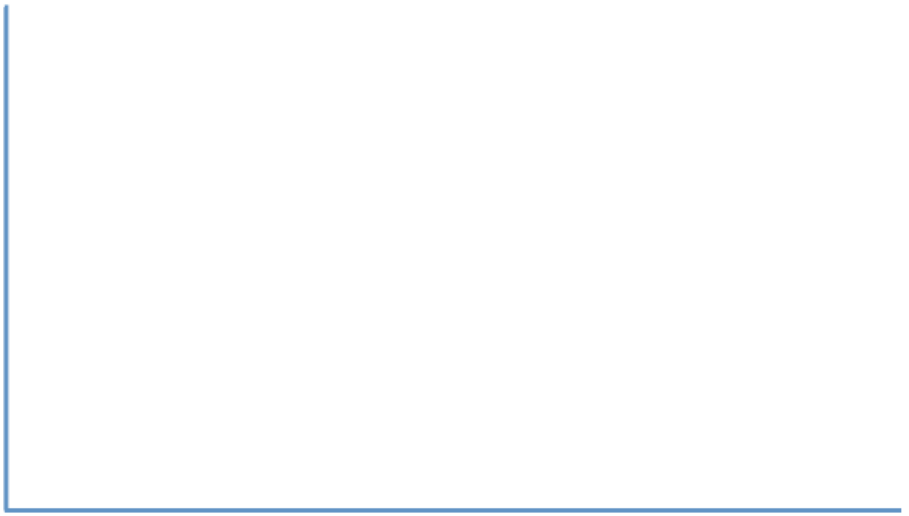
The bias

The variance

Overfitting

Underfitting

Instability of the procedure



Generating the level-1 training data

- Training data for level-1 model contains predictions of level-0 models as attributes; class attribute remains the same

- **Problem:** we cannot use the level-0 models predictions on their *training* data to obtain attribute values for the level-1 data

  - Assume we have a perfect rote learner as one of the level-0 learner
  - Then, the level-1 learner will learn to simply predict this level-0's learners predictions, rendering the ensemble pointless

Generating the level-1 training data

- To solve this, we generate the level-1 training data by running a *cross-validation* for each of the level-0 algorithms
  - Then, the predictions (and actual class values) obtained for the *test instances* encountered during the cross-validation are collected
  - This pooled data obtained from the cross-validation for each level-0 model is used to train the level-1 model

More on stacking

- Stacking is hard to analyze theoretically: “black magic”
- If the base learners can output class probabilities, use those as input to meta learner instead of plain classifications
  - Makes more information available to the level-1 learner
- Important question: which algorithm to use as the meta learner (aka level-1 learner)?
  - In principle, any learning scheme
  - In practice, prefer “relatively global, smooth” models because
  - base learners do most of the work and
  - this reduces the risk of overfitting
- Note that stacking can be trivially applied to numeric prediction too

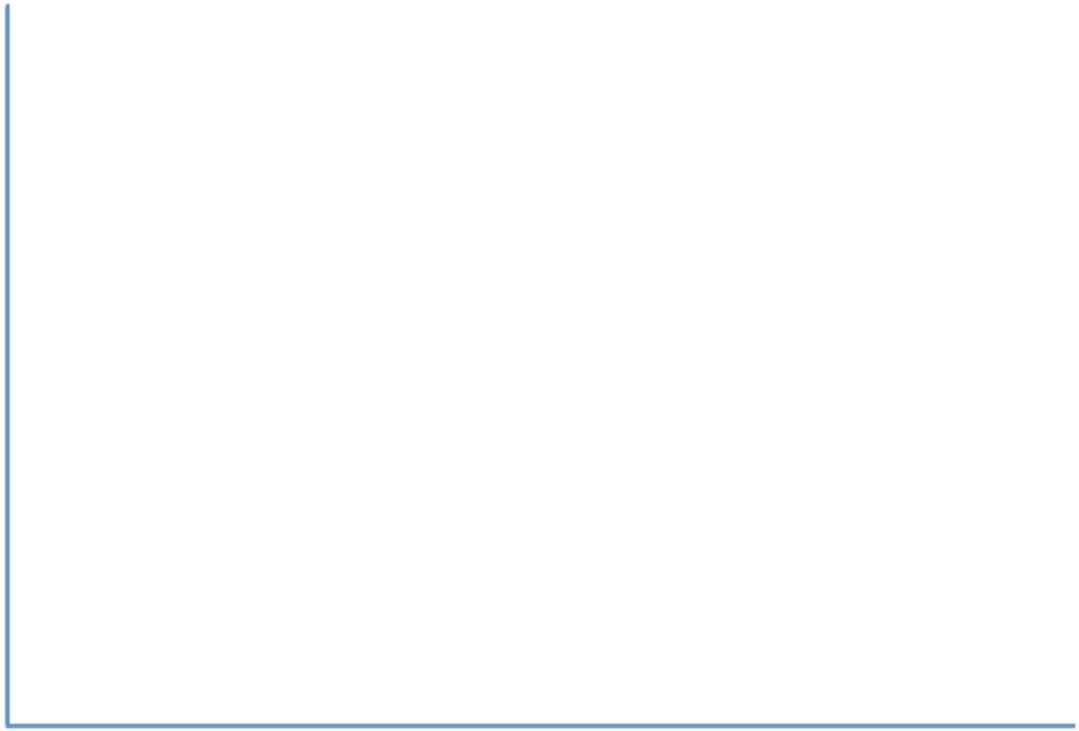
In the _____ algorithm, weighting is used to give more influence to more successful models

Bagging

Boosting

Random forests

Stacking



Weka Demo

- Labor relations dataset

- WEKA Classifier
  - trees.J48 (baseline)
  - meta.Bagging
  - trees.RandomForest
  - meta.AdaboostM1
  - meta.Stacking

Think-Pair-Share: Netflix Prize

- In the Netflix Prize Competition, the goal was to predict the ratings (1-5 stars) that a user would give a movie, based on the other ratings

- Suppose you have developed 500 different regression algorithms for predicting the ratings, some of which work better than others.

  Design an effective method to combine them into a single model. **Justify** your choices.

  Design an evaluation strategy to test whether the ensemble works better than each of the individual models

NETFLIX