Ensemble Learning

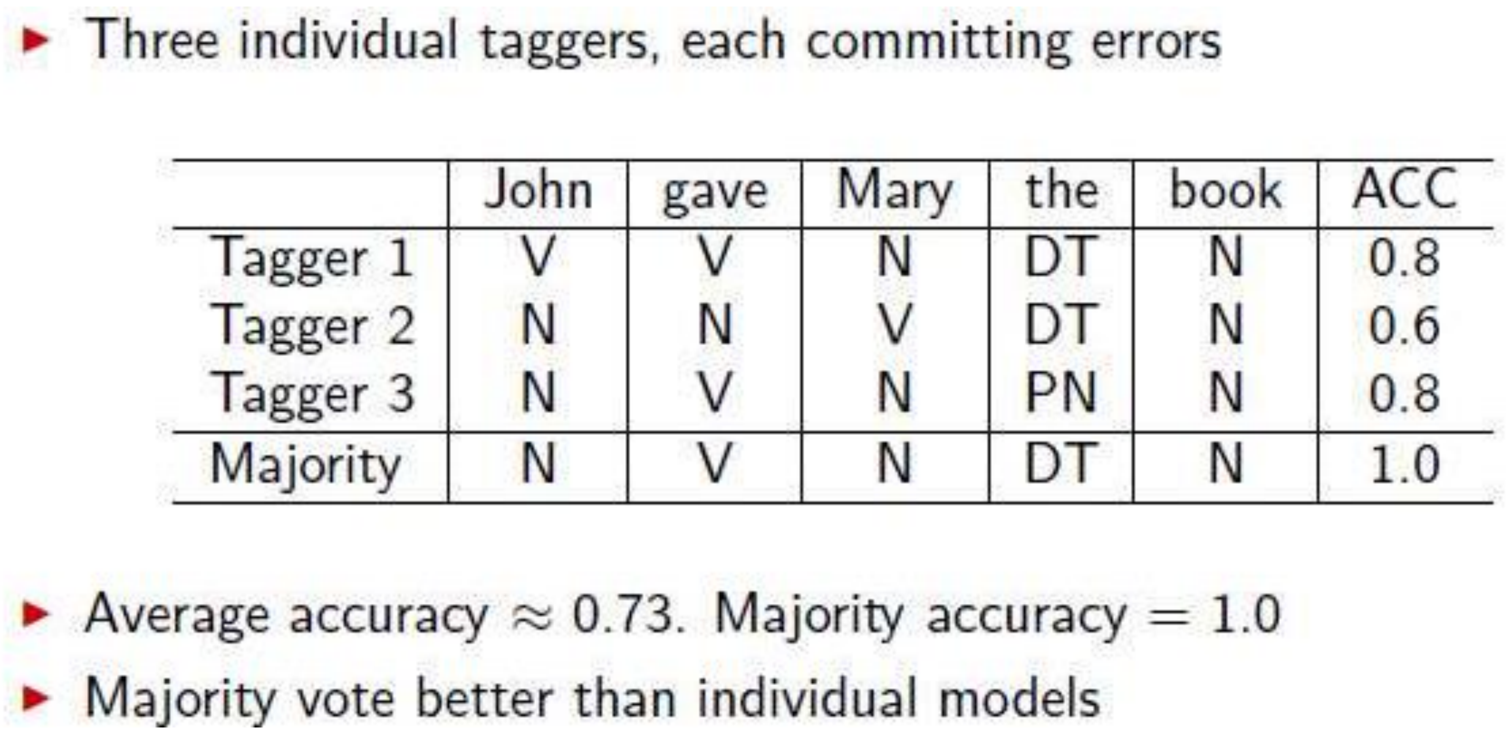
Wisdom of the Crowd: Example 1

- When you want to purchase a new car
- Will you walk up to the first car shop and purchase one based on the advice of the dealer?
  - It's highly unlikely.
- You would likely browser a few web portals about people reviews:
  - Compare different car models,
  - Checking for their features and prices.
- You will also ask friends and colleagues for their opinion.
- In short, you wouldn't directly reach a conclusion, but will instead make a decision considering the opinions of other people as well.
- Ensemble models in machine learning operate on a similar idea.
  - They combine the decisions from multiple models to improve the overall performance.

Example 2: Guess the weight of an ox

- Average of people's votes close to true weight
- Better than most individual members' votes and cattle experts' votes
- Intuitively, the law of large numbers…

Example 3: Taggers accuracy due to Majority voting



- Three individual taggers, each committing errors

	John	gave	Mary	the	book	ACC
Tagger 1	V	V	N	DT	N	0.8
Tagger 2	N	N	V	DT	N	0.6
Tagger 3	N	V	N	PN	N	0.8
Majority	N	V	N	DT	N	1.0

- Average accuracy ≈ 0.73. Majority accuracy $= 1.0$
- Majority vote better than individual models

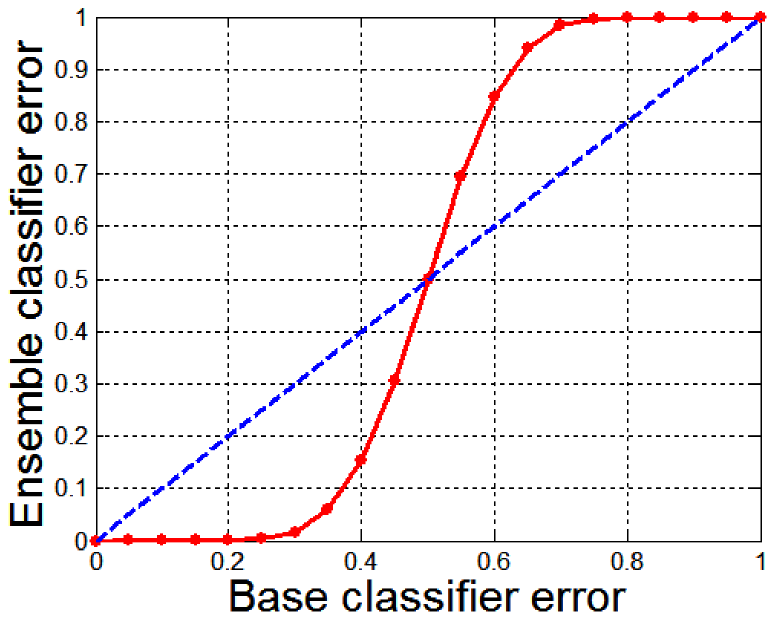
- An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples (Dietterich, 2000).
- Main discovery from some researches shows that ensembles are often much more accurate than the individual classifiers that make them up.

Diversity vs accuracy

- An ensemble of classifiers must be more accurate than any of its individual members.
- The individual classifiers composing an ensemble must be accurate and diverse:
  - An accurate classifier is one that has an error rate better than random when guessing new examples
  - Two classifiers are diverse if they make different errors on new data points, or make uncorrelated errors.

Why Ensemble Methods work?

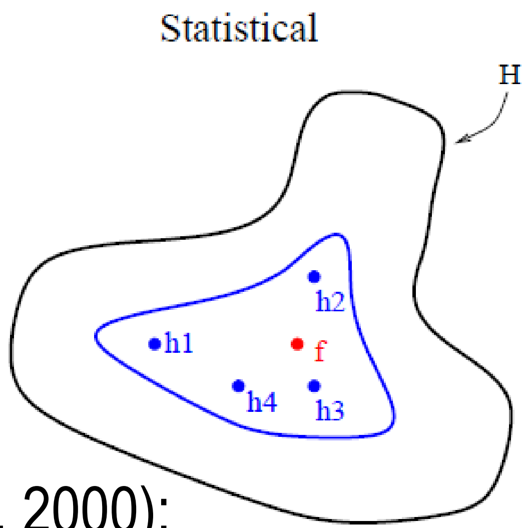
- Suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - Assume errors made by classifiers are uncorrelated
  - Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Why it works

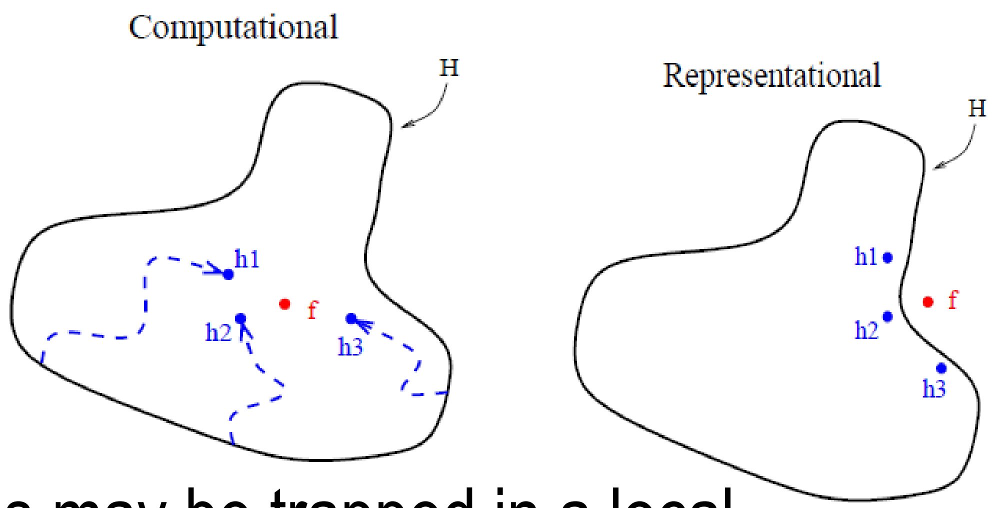
The outer curve denotes the hypothesis space H.
The inner curve denotes the set of hypotheses that all give good accuracy on the training data.
The point labeled f is the true hypothesis and we can see that by averaging the accurate hypotheses we can find a good approximation to f.



- It is possible to build good ensembles for three fundamental reasons (Dietterich , 2000):

- **Statistical reason:** A learning algorithm can be viewed as searching a space H of hypotheses to identify the best one in the space.
- The number of training samples available is small comparing to the size of hypotheses space,
- As a result the learning algorithm could find different hypotheses(classifiers) that all give out some accuracy.
- The ensemble of these classifiers could average their votes and reduce the risk of choosing the wrong classifier.

Why it works



- Computational reason: Local search algorithms may be trapped in a local minima, and if there is enough training data, it is computationally hard to get the best hypotheses. For ensemble learning the local search could start from different points and may provide a better approximation than any of the individual ones.
- Representational reason: The true function f cannot be represented by any of the hypotheses in the space in most machine learning applications, but weighted sum of hypotheses drawn from the space may expand the space of representation functions.

Base Learners

- An ensemble contains a number of learners which are usually called base learners.
  - The generalization ability of an ensemble is usually much stronger than that of base learners.
  - Ensemble learning is able to boost weak learners which are slightly better than random guess to strong learners which can make very accurate predictions.
  - So, "base learners" are also referred as "weak learners"
- Base learners are usually generated from training data
  - A base learning algorithm can be a decision tree, neural network, or other kinds of machine learning algorithms.
- Most ensemble methods use a single base learning algorithm to produce homogeneous base learners,
  - However, some methods use multiple learning algorithms to produce heterogeneous learners.

Error in Ensemble Learning (Variance vs. Bias)

- The error emerging from any model can be broken down into three components mathematically.
- The following are these components:

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - E[\hat{f}(x)]\right]^2 + \sigma_e^2$$

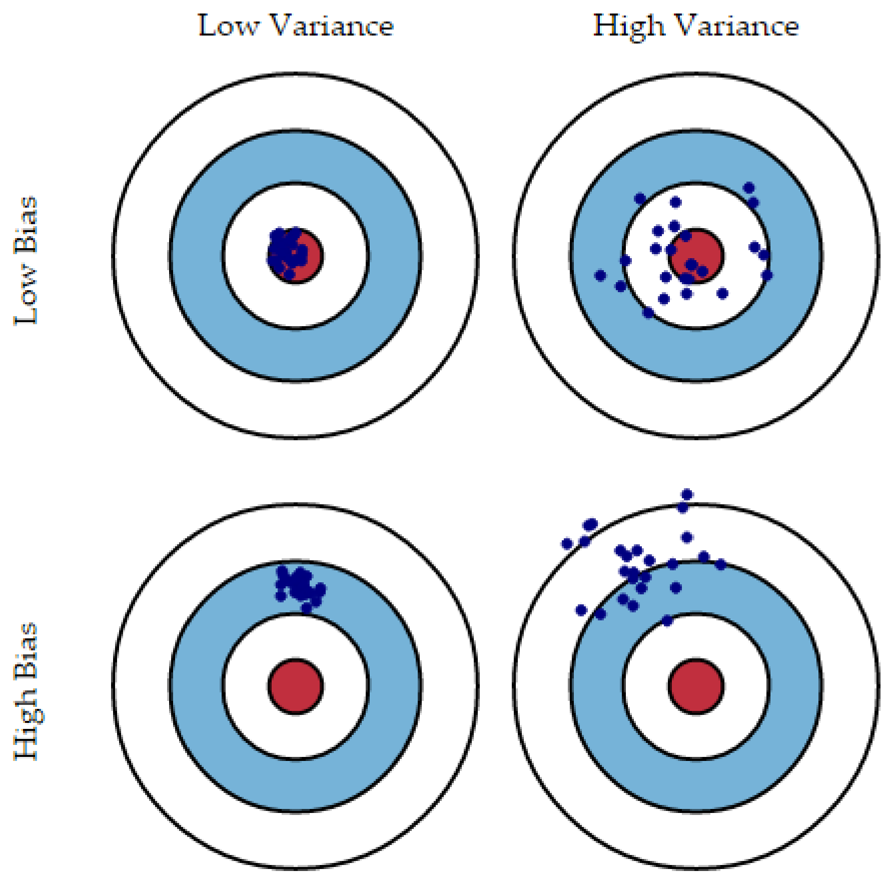
$$Err(x) = \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ Error}$$

- Bias error is useful to quantify how much on average the predicted values are different from the actual value.
  - A high bias error means we have an underperforming model which keeps on missing important trends.

- A Variance quantifies how the prediction made on same observation are different from each other.
  - A high variance model will over-fit on your training population and perform badly on any observation beyond training.

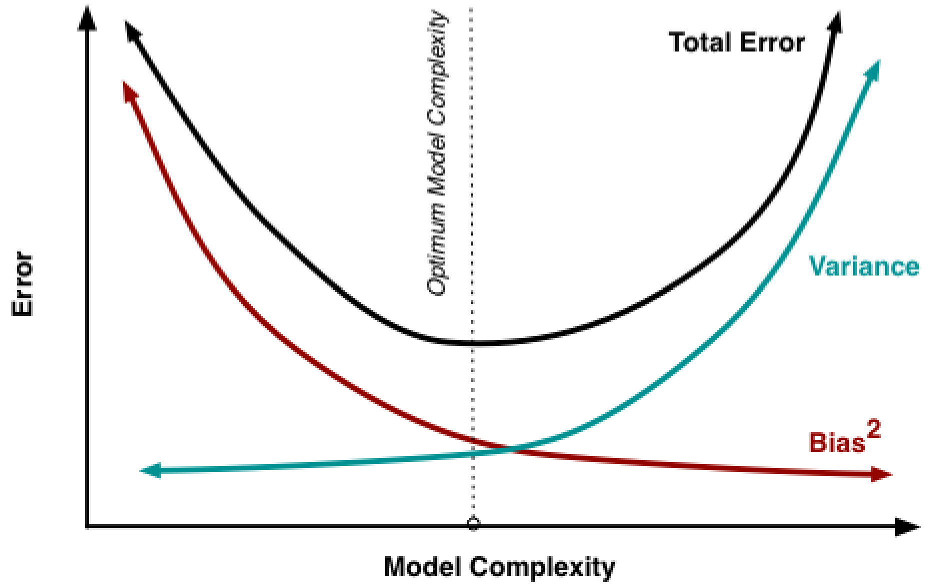
Error in Ensemble Learning (Variance vs. Bias)

- Assume that red spot is the real value and blue dots are predictions



Error in Ensemble Learning (Variance vs. Bias)

- As you increase the complexity of your model, you will see a reduction in error due to lower bias in the model.
- However, this only happens till a particular point.
- As you continue to make your model more complex, you end up over-fitting your model and hence your model will start suffering from high variance.
- A champion model should maintain a balance between these two types of errors.
- This is known as the trade-off management of bias-variance errors.
    - Ensemble learning is one way to execute this trade-off analysis.



Simple Ensemble Techniques

1. Max Voting

- The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point.
- The predictions by each model are considered as a ‘vote’. The predictions which we get from the majority of the models are used as the final prediction.

Colleague 1	**Colleague 2**	**Colleague 3**	**Colleague 4**	**Colleague 5**	**Final rating**
5	**4**	**5**	**4**	**4**	**4**

```
from sklearn.ensemble import VotingClassifier
model1 = LogisticRegression(random_state=1)
model2 = tree.DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1),
('dt', model2)], voting='hard')
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

Simple Ensemble Techniques

2. Averaging

- Similar to the max voting technique, multiple predictions are made for each data point in averaging.
- In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.
- For example, in the below case, the averaging method would take the average of all the values. i.e. $(5+4+5+4+4)/5 = 4.4$

Simple Ensemble Techniques

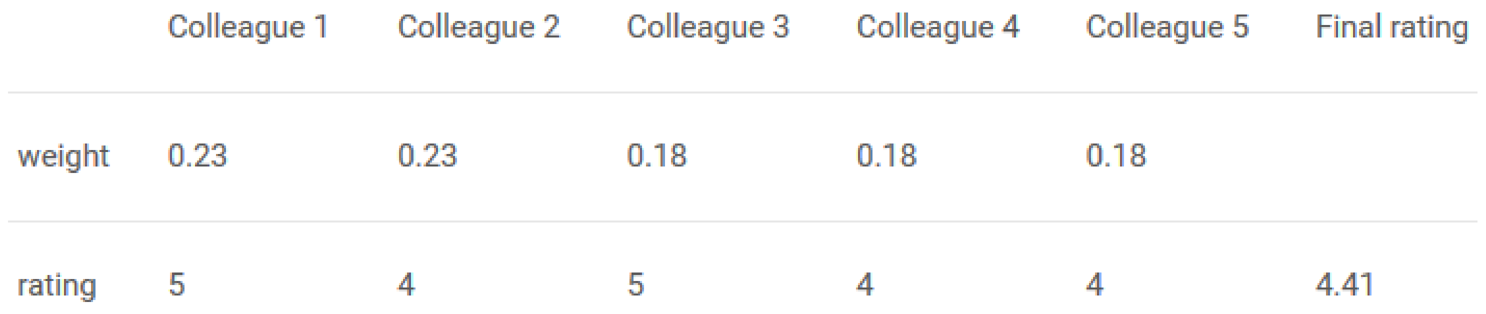
Sample Code:

```
model1 =
tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()


model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1+pred2+pred3)/3
```

Simple Ensemble Techniques

3. Weighted Average

- In this method, all models are assigned different weights defining the importance of each model for prediction.
- For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.
- The result is calculated as [(5*0.23) + (4*0.23) + (5*0.18) + (4*0.18) + (4*0.18)] = 4.41.



	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
weight	0.23	0.23	0.18	0.18	0.18	
rating	5	4	5	4	4	4.41

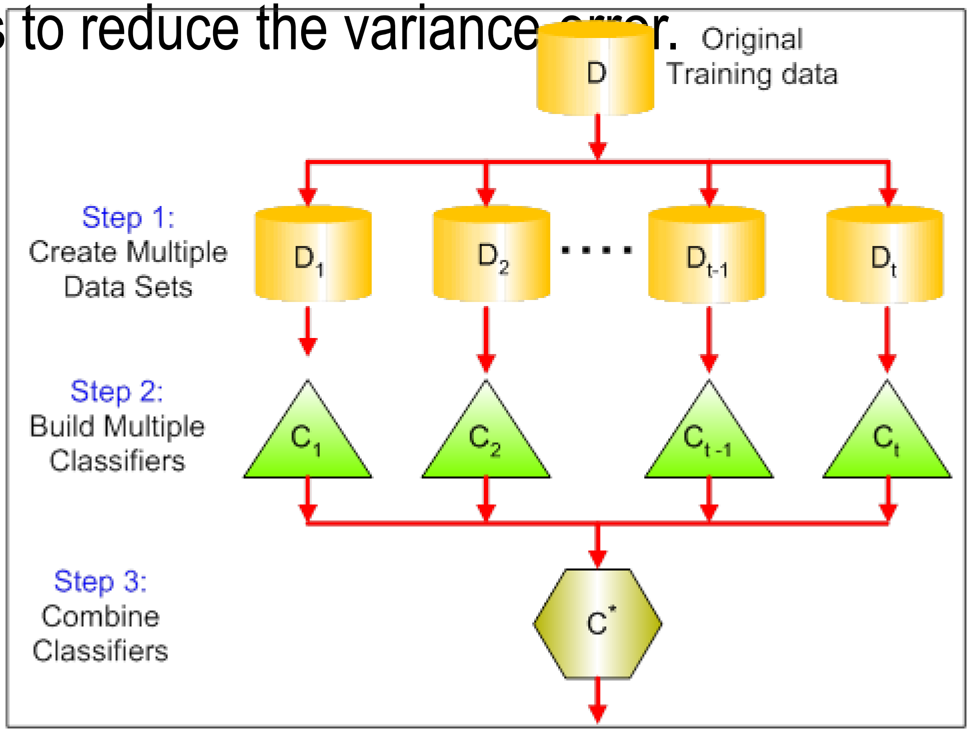
Simple Ensemble Techniques

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)
```

Commonly used Ensemble learning techniques

1. Bagging :

- Bagging tries to implement similar learners on small sample populations and then takes a mean of all the predictions.
- In generalized bagging, you can use different learners on different population. As you can expect this helps us to reduce the variance error.



Bagging

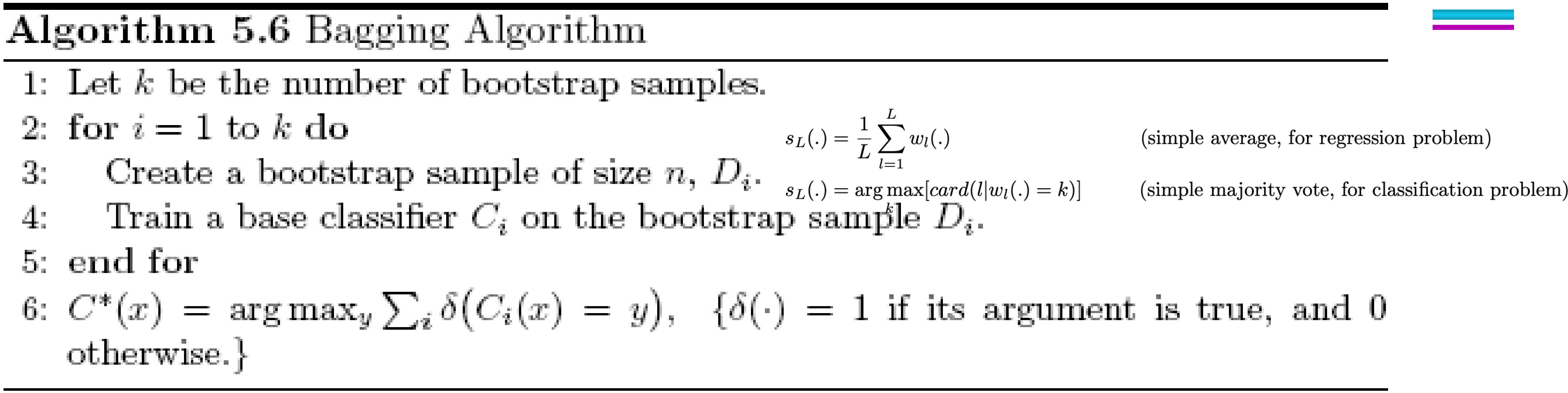
- Sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample

- Each sample has probability $(1 – 1/n)^n$ of being selected

Bagging Algorithm



Algorithm 5.6 Bagging Algorithm

```
1: Let $k$ be the number of bootstrap samples.
2: for $i = 1$ to $k$ do
3:     Create a bootstrap sample of size $n$, $D_i$.
4:     Train a base classifier $C_i$ on the bootstrap sample $D_i$.
5: end for
6: $C^*(x) = \arg\max_y \sum_i \delta(C_i(x) = y)$,  $\{\delta(\cdot) = 1$ if its argument is true, and 0
   otherwise.$\}$
```

$$s_L(.) = \frac{1}{L}\sum_{l=1}^{L} w_l(.)$$ (simple average, for regression problem)

$$s_L(.) = \arg\max_k [card(l|w_l(.) = k)]$$ (simple majority vote, for classification problem)

There are several possible ways to aggregate the multiple models fitted in parallel.

- For a regression problem, the outputs of individual models can literally be averaged to obtain the output of the ensemble model.
- For a classification problem the class outputted by each model can be seen as a vote and the class that receives the majority of the votes is returned by the ensemble model (this is called hard-voting).
  - Still, for a classification problem, we can also consider the probabilities of each class returned by all the models, average these probabilities, and keep the class with the highest average probability (this is called soft-voting).
  - Averages or votes can either be simple or weighted if any relevant weights can be used.

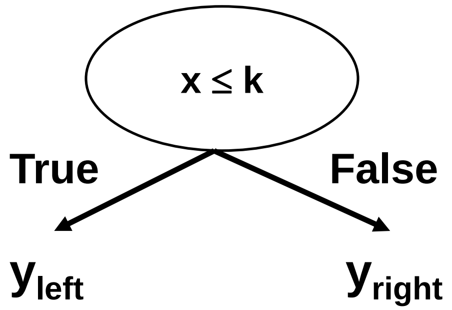
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use the majority vote to determine the class of ensemble classifier

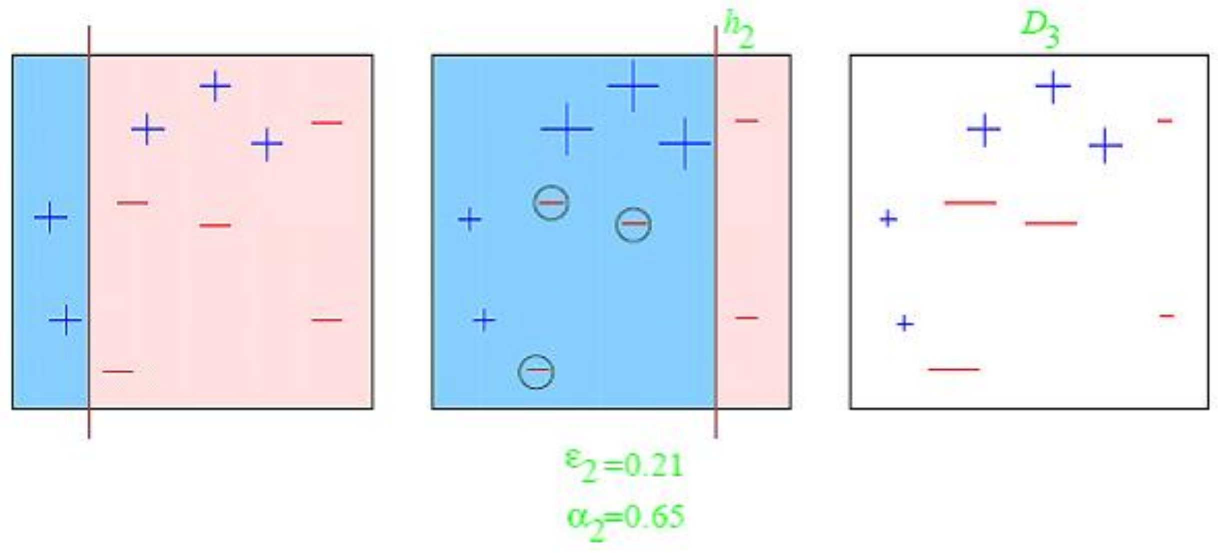
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Some Commonly used Ensemble learning techniques

2. Boosting :

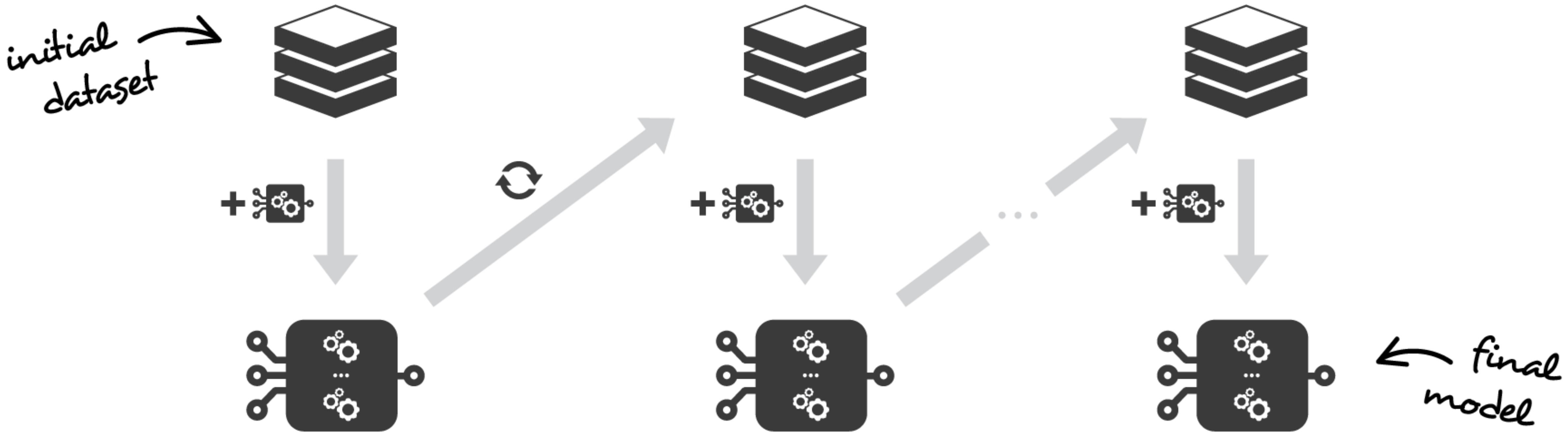
- An iterative technique which adjusts the weight of an observation based on the last classification.
- If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa.
    - Boosting in general decreases the bias error and builds strong predictive models. However, they may sometimes over fit on the training data.



Boosting

train a weak model and aggregate it to the ensemble model

update the training dataset (values or weights) based on the current ensemble model results



Stacking

- The idea of stacking is to learn several different weak learners and combine them by training a meta-model to output predictions based on the multiple predictions returned by these weak models.
- We need to define two things in order to build our stacking model:
  - the L learners we want to fit and the meta-model that combines them.
- For example, for a classification problem, we can choose as weak learners a KNN classifier, a logistic regression and a SVM, and decide to learn a neural network as meta-model.
- The neural network will take as inputs the outputs of our three weak learners and will learn to return final predictions based on it.

- We follow the following steps:

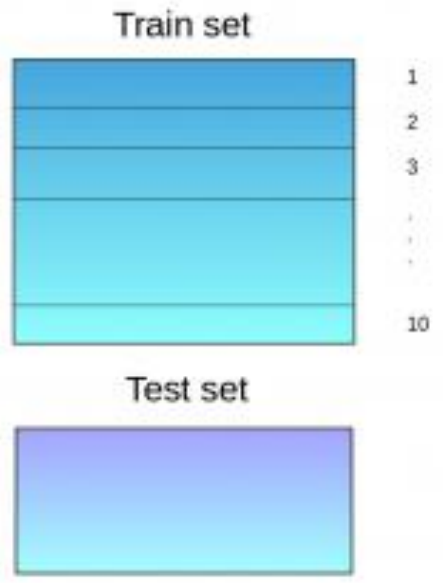
  - Split the data into two sets, the train set D, and the test set D_h.
  - Choose L weak learners and fit them to the train set D
    - Here, k-fold cross-training is used in which k-1 folds from D are used to train weak learners and then prediction is made on the k-th fold from the Dh.
  - Fit the meta-model on the test set Dh, using predictions made by the weak learners as inputs

Algorithm Stacking

```
1: Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
2: Ouput: ensemble classifier $H$
3: *Step 1: learn base-level classifiers*
4: for $t = 1$ to $T$ do
5:     learn $h_t$ based on $D$
6: end for
7: *Step 2: construct new data set of predictions*
8: for $i = 1$ to $m$ do
9:     $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
10: end for
11: *Step 3: learn a meta-classifier*
12: learn $H$ based on $D_h$
13: return $H$
```

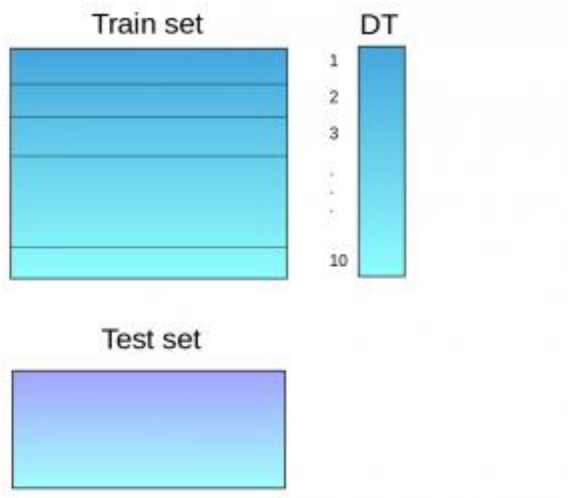
Stacking

- Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set. Below is a step-wise explanation for a simple stacked ensemble:
- The train set is split into 10 parts.



Stacking

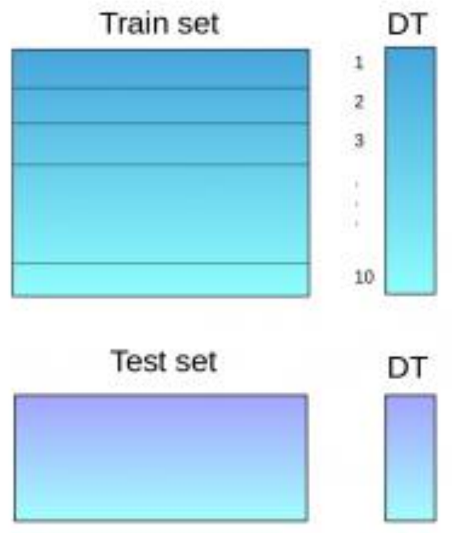
- A base model (suppose a decision tree) is fitted on 9 parts and predictions are made for the 10th part. This is done for each part of the train set.



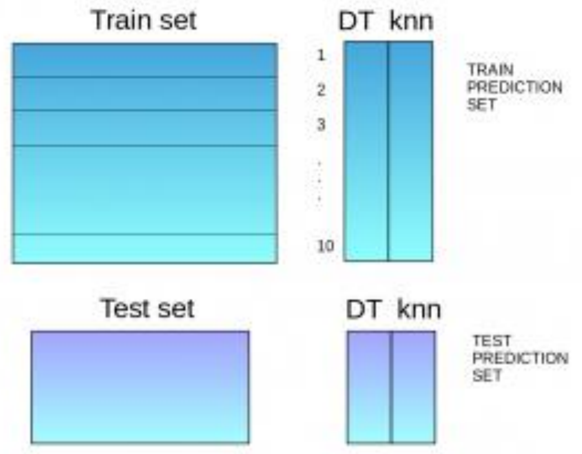
- In this way, the base model (in this case, decision tree) is fitted on the whole training dataset.

Stacking

- Using this model, predictions are made on the test set.

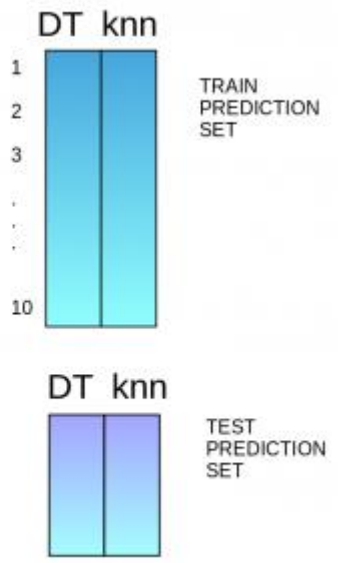


- Steps 2 to 4 are repeated for another base model (say knn) resulting in another set of predictions for the train set and test set.

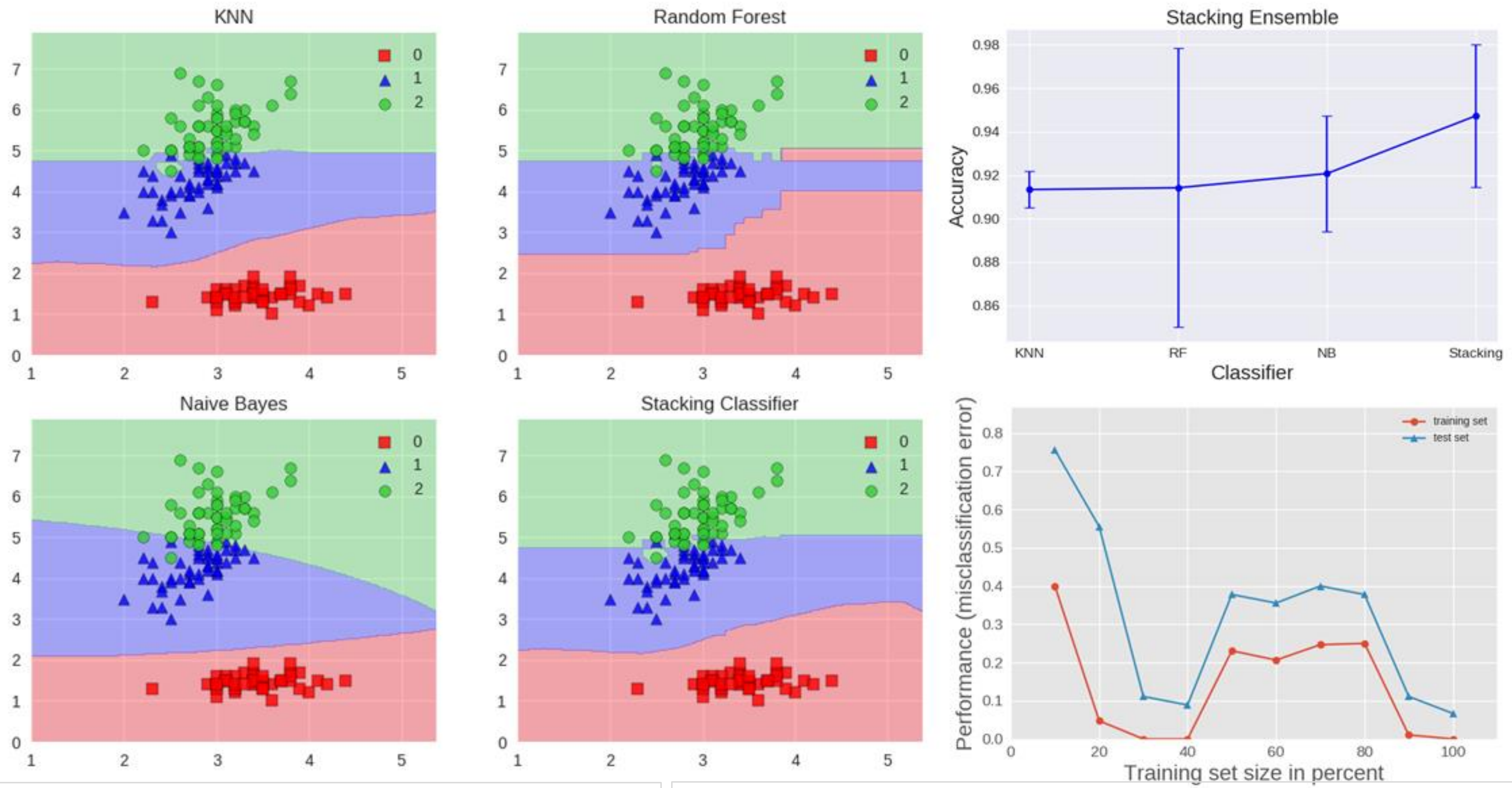


Stacking

- The predictions from the train set are used as features to build a new model.



- This model is used to make final predictions on the test prediction set.

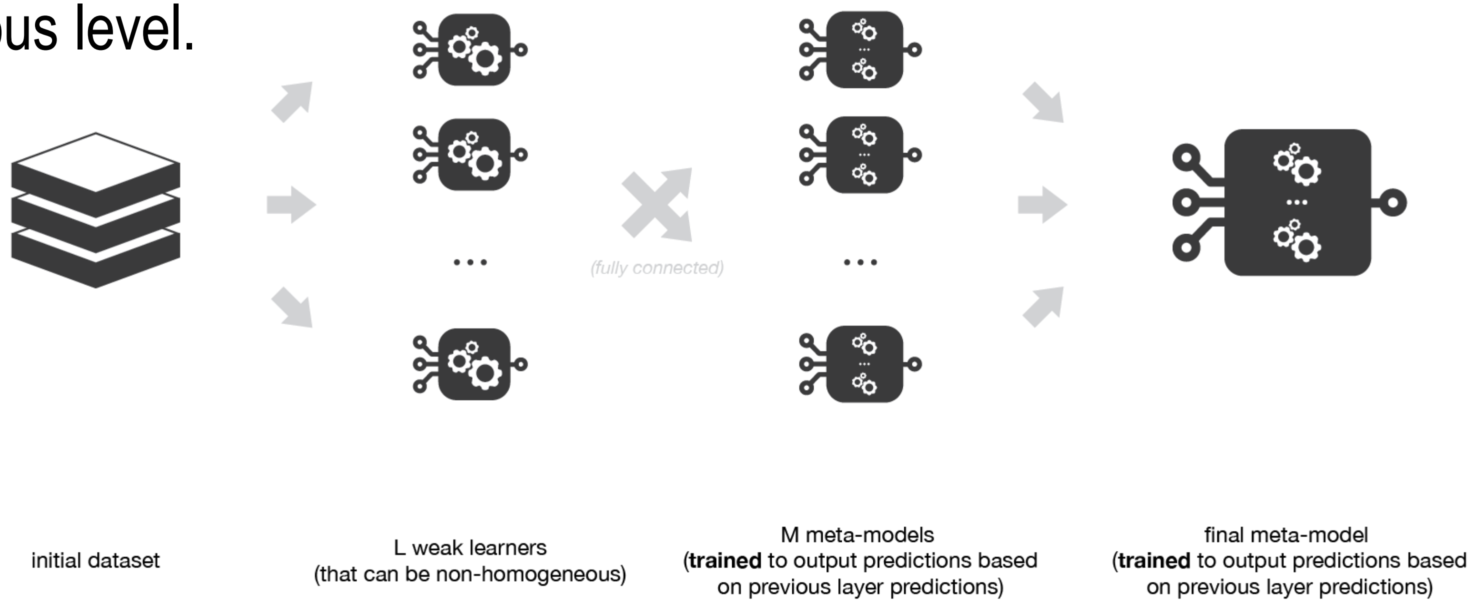


The following accuracy is visualized in the top right plot of the figure above:
Accuracy: 0.91 (+/- 0.01) [KNN]
Accuracy: 0.91 (+/- 0.06) [Random Forest]
Accuracy: 0.92 (+/- 0.03) [Naive Bayes]
Accuracy: 0.95 (+/- 0.03) [Stacking Classifier]

The stacking ensemble is illustrated in the figure above. It consists of k-NN, Random Forest, and Naive Bayes base classifiers whose predictions are combined by Logistic Regression as a meta-classifier. We can see the blending of decision boundaries achieved by the stacking classifier. The figure also shows that stacking achieves higher accuracy than individual classifiers and based on learning curves, it shows no signs of overfitting.

Multi-levels Stacking

- A possible extension of stacking is multi-level stacking. It consists in doing stacking with multiple layers.
- As an example, let's consider a 3-levels stacking.
- In the first level (layer), we fit the L weak learners that have been chosen.
- Then, in the second level, instead of fitting a single meta-model on the weak models predictions (as it was described in the previous subsection) we fit M such meta-models.
- Finally, in the third level, we fit a last meta-model that takes as inputs the predictions returned by the M meta-models of the previous level.


(fully connected)
initial dataset
L weak learners
(that can be non-homogeneous)
M meta-models
(**trained** to output predictions based on previous layer predictions)
final meta-model
(**trained** to output predictions based on previous layer predictions)

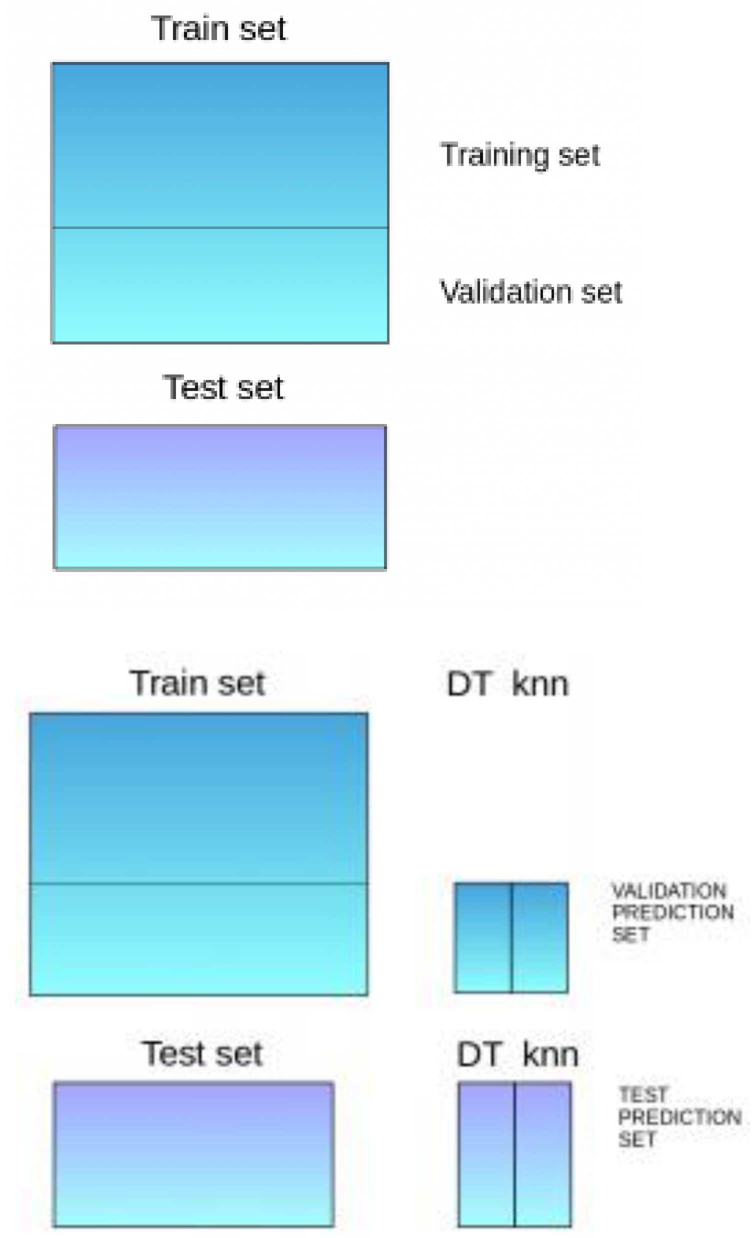
Blending

- Blending follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions.
- In other words, unlike stacking, the predictions are made on the holdout set only.
- The holdout set and the predictions are used to build a model which is run on the test set.

Blending

Steps:

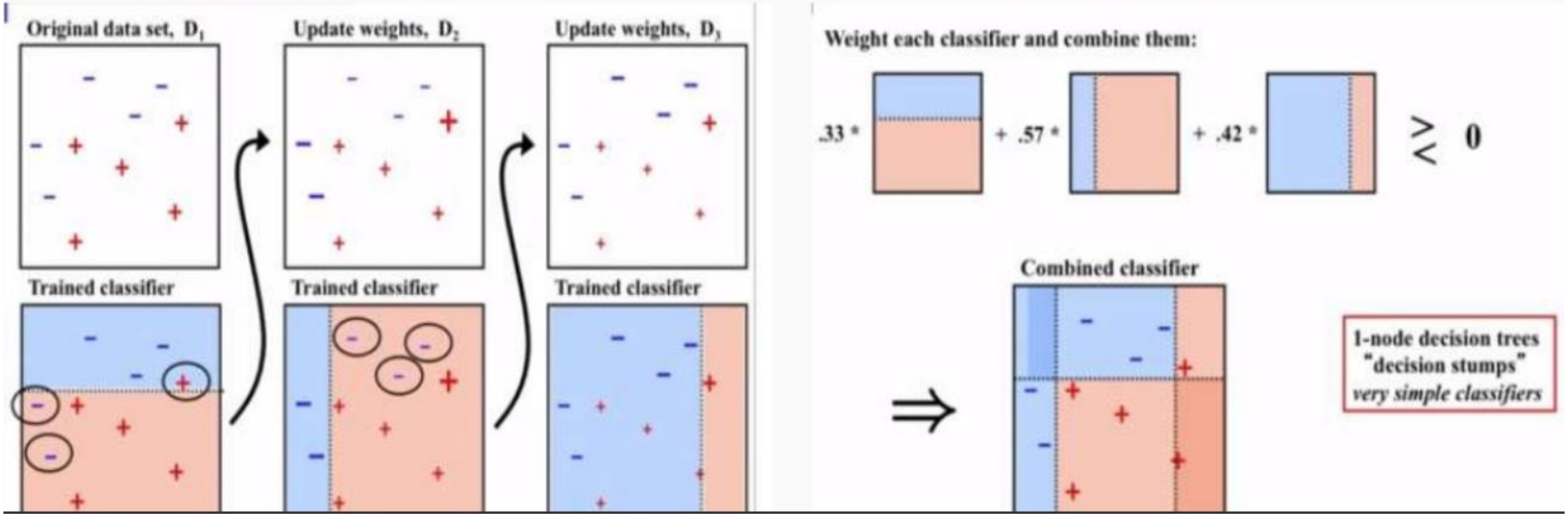
- The train set is split into training and validation sets.
- Model(s) are fitted on the training set.
- The predictions are made on the validation set and the test set.
- The validation set and its predictions are used as features to build a new model.
- This model is used to make final predictions on the test and meta-features.



AdaBoost

- Suited for bi-class classification
- Steps are as follows
- Train weak learner on training data
- Increase weights of misclassified data
- Increased weight data has more chances of getting picked in next model training
- Final prediction is function of all the participating models

AdaBoost



Original data set, D_1
Update weights, D_2
Update weights, D_3
Trained classifier
Trained classifier
Trained classifier
Weight each classifier and combine them:
.33 *
+ .57 *
+ .42 *
$\gtrless$ 0
Combined classifier
$\Rightarrow$
1-node decision trees
"decision stumps"
very simple classifiers

Dataset:

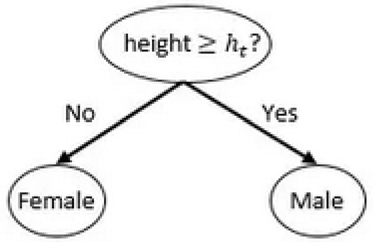
x1	x2	Decision
2	3	true
2.1	2	true
4.5	6	true
4	3.5	false
3.5	1	false
5	7	true
5	3	false
6	5.5	true
8	6	false
8	2	false

True classes are replaced as +1, and false classes are replaced as -1 in the Decision column.

Decision stumps

- Decision trees approaches problems with divide and conquer method. They might have lots of nested decision rules. This makes them non-linear classifiers.

- In contrast, decision stumps are 1-level decision trees. They are linear classifiers just like (single layer) perceptrons.
- For Example: If height of someone is greater than 1.70 meters (5.57 feet), then it would be male. Otherwise, it would be female.
- This decision stump would classify gender correctly at least 50% accuracy. That's why, these classifiers are weak learners.



Regression for adaboost

- The main principle in adaboost is to increase the weight of unclassified ones and to decrease the weight value of classified ones.
- But we are working on a classification problem. Target values in the data set are nominal values.
  - That's why, we are going to transform the problem to a regression task, i.e. set true classes to 1 whereas false classes to -1 to handle this.

x1	x2	actual	weight	weighted_actual
2	3	1	0.1	0.1
2	2	1	0.1	0.1
4	6	1	0.1	0.1
4	3	-1	0.1	-0.1
4	1	-1	0.1	-0.1
5	7	1	0.1	0.1
5	3	-1	0.1	-0.1
6	5	1	0.1	0.1
8	6	-1	0.1	-0.1
8	2	-1	0.1	-0.1

It will be 0 if the prediction is correct, will be 1 if the prediction is incorrect.

x1	x2	actual	weight	weighted_actual	prediction	loss	weight * loss
2	3	1	0.1	0.1	1	0	0
2	2	1	0.1	0.1	1	0	0
4	6	1	0.1	0.1	-1	1	0.1
4	3	-1	0.1	-0.1	-1	0	0
4	1	-1	0.1	-0.1	-1	0	0
5	7	1	0.1	0.1	-1	1	0.1
5	3	-1	0.1	-0.1	-1	0	0
6	5	1	0.1	0.1	-1	1	0.1
8	6	-1	0.1	-0.1	-1	0	0
8	2	-1	0.1	-0.1	-1	0	0

- $Epsilon\ or\ total\ error = \sum Weight\ x\ loss$ = 0.3
- Define alpha = ln[(1-epsilon)/epsilon] / 2 = ln[(1 – 0.3)/0.3] / 2 = 0.42
- w_{i+1} = w_i * math.exp(-alpha * actual * prediction) where i refers to instance number.

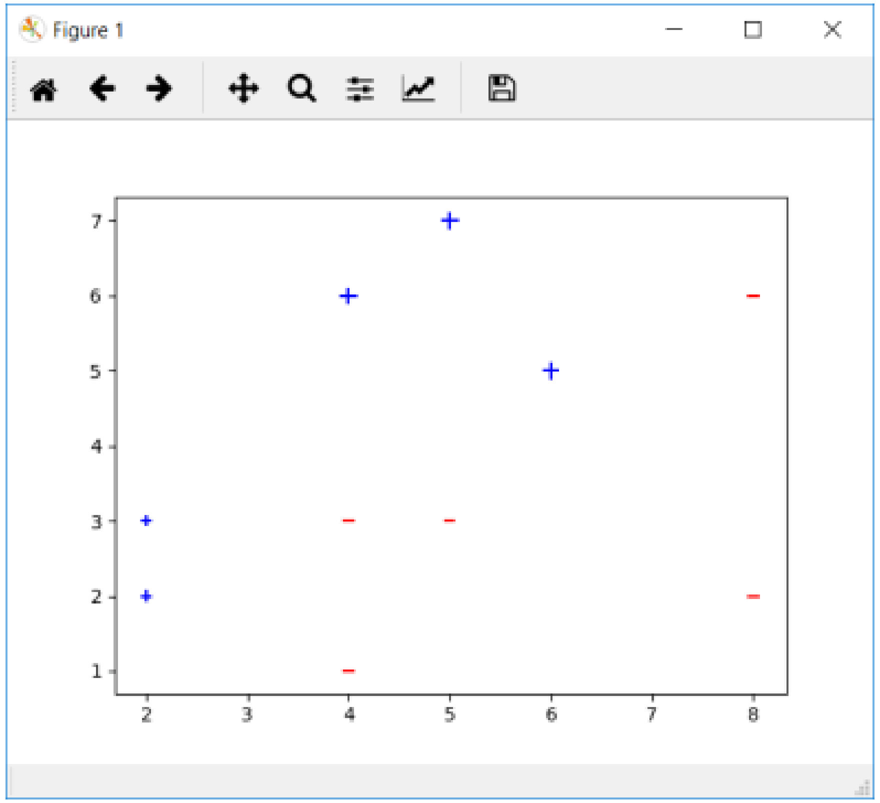
x1	**x2**	**actual**	**weight**	**prediction**	**w_(i+1)**	**norm(w_(i+1))**
2	3	1	0.1	1	0.065	0.071
2	2	1	0.1	1	0.065	0.071
4	6	1	0.1	-1	0.153	0.167
4	3	-1	0.1	-1	0.065	0.071
4	1	-1	0.1	-1	0.065	0.071
5	7	1	0.1	-1	0.153	0.167
5	3	-1	0.1	-1	0.065	0.071
6	5	1	0.1	-1	0.153	0.167
8	6	-1	0.1	-1	0.065	0.071
8	2	-1	0.1	-1	0.065	0.071

Round 2

I shift normalized w_(i+1) column to weight column in this round. Then, build a decision stump. Still, x1 and x2 are features whereas weighted actual is the target value.

x1	x2	actual	weight	weighted_actual
2	3	1	0.071	0.071
2	2	1	0.071	0.071
4	6	1	0.167	0.167
4	3	-1	0.071	-0.071
4	1	-1	0.071	-0.071
5	7	1	0.167	0.167
5	3	-1	0.071	-0.071
6	5	1	0.167	0.167
8	6	-1	0.071	-0.071
8	2	-1	0.071	-0.071

The weights of correctly classified ones decreased whereas incorrect ones increased.



Data set with tuned weights

x1	x2	actual	weight	prediction	loss	weight * loss
2	3	1	0.071	-1	1	0.071
2	2	1	0.071	-1	1	0.071
4	6	1	0.167	1	0	0.000
4	3	-1	0.071	-1	0	0.000
4	1	-1	0.071	-1	0	0.000
5	7	1	0.167	1	0	0.000
5	3	-1	0.071	-1	0	0.000
6	5	1	0.167	1	0	0.000
8	6	-1	0.071	1	1	0.071
8	2	-1	0.071	-1	0	0.000

Calculate $Epsilon\ or\ total\ error = \sum Weight\ x\ loss$
and alpha values
alpha = ln[(1-epsilon)/epsilon] / 2
for round 2.
epsilon = 0.21, alpha = 0.65

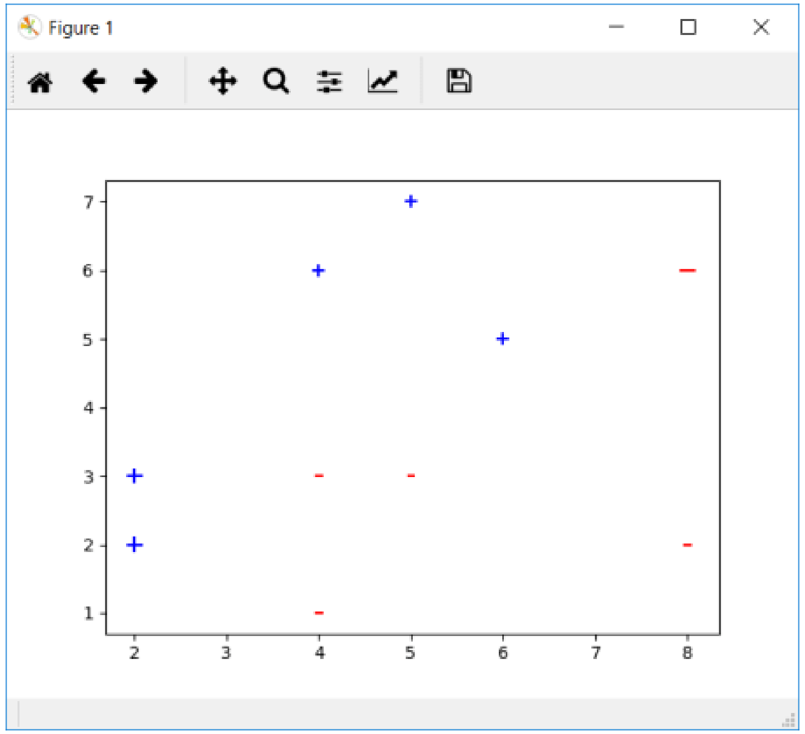
x1	x2	actual	weight	prediction	w_(i+1)	norm(w_(i+1))
2	3	1	0.071	-1	0.137	0.167
2	2	1	0.071	-1	0.137	0.167
4	6	1	0.167	1	0.087	0.106
4	3	-1	0.071	-1	0.037	0.045
4	1	-1	0.071	-1	0.037	0.045
5	7	1	0.167	1	0.087	0.106
5	3	-1	0.071	-1	0.037	0.045
6	5	1	0.167	1	0.087	0.106
8	6	-1	0.071	1	0.137	0.167
8	2	-1	0.071	-1	0.037	0.045

Round 3

I skipped calculations for the following rounds

epsilon = 0.31, alpha = 0.38

x1	x2	actual	weight	prediction	loss	w * loss	w_(i+1)	norm(w_(i+1))
2	3	1	0.167	1	0	0.000	0.114	0.122
2	2	1	0.167	1	0	0.000	0.114	0.122
4	6	1	0.106	-1	1	0.106	0.155	0.167
4	3	-1	0.045	-1	0	0.000	0.031	0.033
4	1	-1	0.045	-1	0	0.000	0.031	0.033
5	7	1	0.106	-1	1	0.106	0.155	0.167
5	3	-1	0.045	-1	0	0.000	0.031	0.033
6	5	1	0.106	-1	1	0.106	0.155	0.167
8	6	-1	0.167	-1	0	0.000	0.114	0.122
8	2	-1	0.045	-1	0	0.000	0.031	0.033

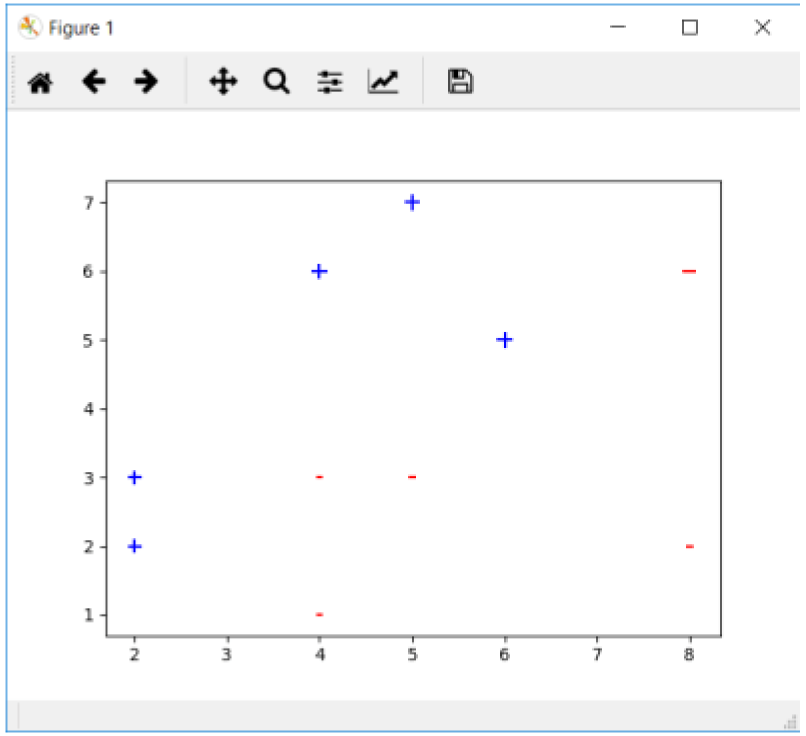


Weights in round 3

Round 4

x1	x2	actual	weight	prediction	loss	w * loss	w_(i+1)	norm(w_(i+1))
2	3	1	0.122	1	0	0.000	0.041	0.068
2	2	1	0.122	1	0	0.000	0.041	0.068
4	6	1	0.167	1	0	0.000	0.056	0.093
4	3	-1	0.033	1	1	0.033	0.100	0.167
4	1	-1	0.033	1	1	0.033	0.100	0.167
5	7	1	0.167	1	0	0.000	0.056	0.093
5	3	-1	0.033	1	1	0.033	0.100	0.167
6	5	1	0.167	1	0	0.000	0.056	0.093
8	6	-1	0.122	-1	0	0.000	0.041	0.068
8	2	-1	0.033	-1	0	0.000	0.011	0.019

epsilon = 0.10, alpha = 1.10



Weights in round 4

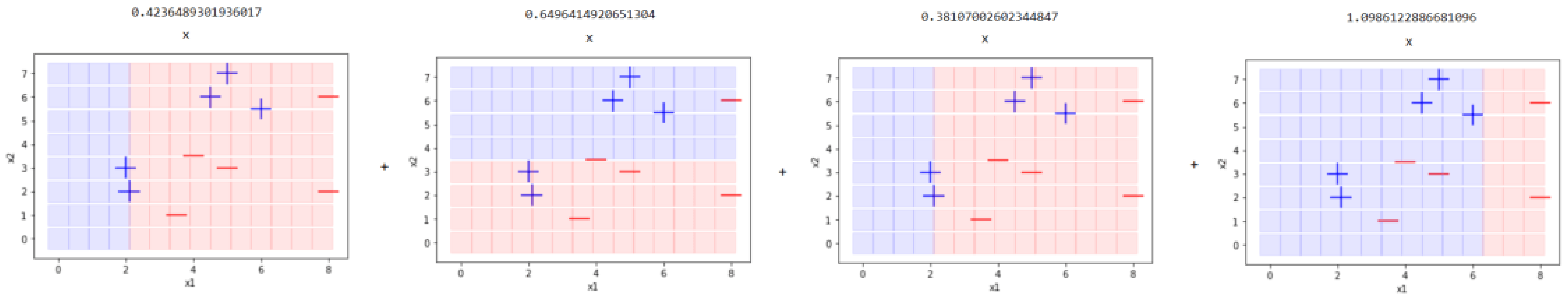
Prediction

Cumulative sum of each round’s alpha times prediction gives the final prediction

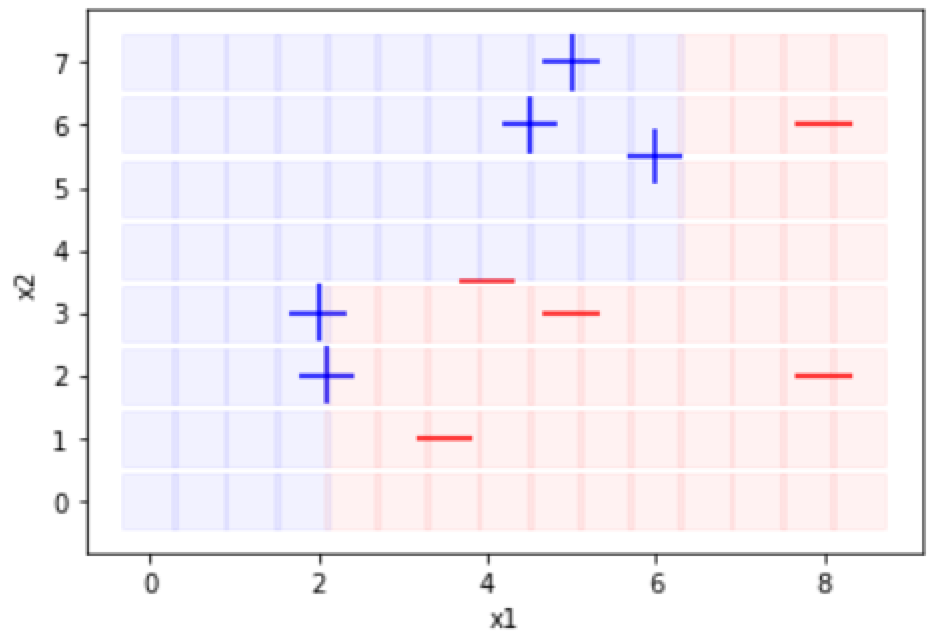
round 1 alpha	**round 2 alpha**	**round 3 alpha**	**round 4 alpha**
0.42	0.65	0.38	1.1
round 1 prediction	**round 2 prediction**	**round 3 prediction**	**round 4 prediction**
1	-1	1	1
1	-1	1	1
-1	1	-1	1
-1	-1	-1	1
-1	-1	-1	1
-1	1	-1	1
-1	-1	-1	1
-1	1	-1	1
-1	1	-1	-1
-1	-1	-1	-1

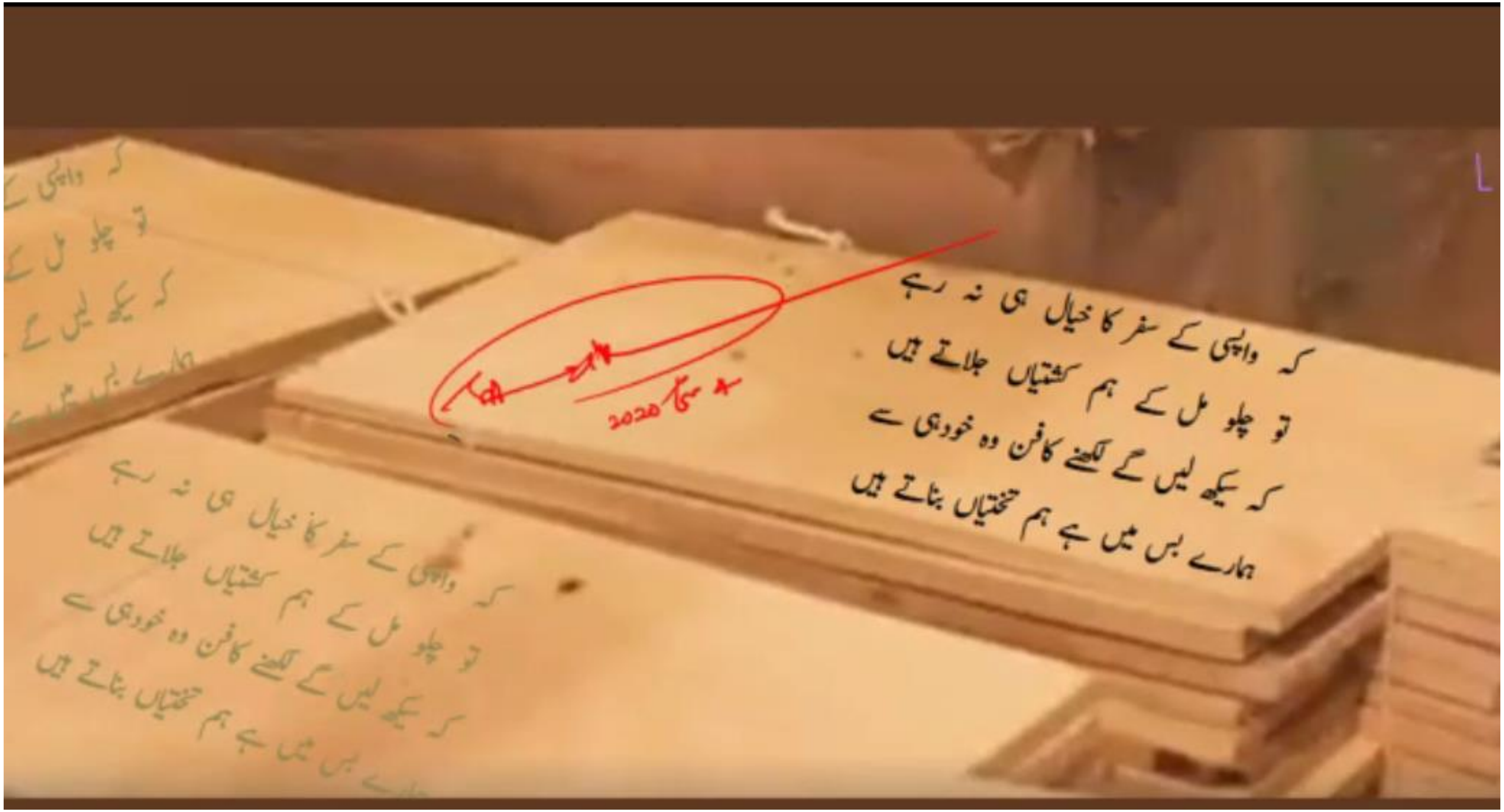
- For example, prediction of the 1st instance will be: 0.42 x 1 + 0.65 x (-1) + 0.38 x 1 + 1.1 x 1 = 1.25
- and we will apply sign function Sign(0.25) = +1 aka true which is correctly classified.

We can summarize adaboost calculations as illustrated below. There are 4 different (weak) classifier and its multiplier alphas. Instances located in blue area will be classified as true whereas located in read area will be classified as false.



If we apply this calculation for all instances, all instances are classified correctly. In this way, you can find decision for a new instance not appearing in the train set.





کہ واپسی کے سفر کا خیال ہی نہ رہے
تو چلو مل کے ہم کشتیاں جلاتے ہیں
کہ دیکھ لیں گے لکھنے کا فن وہ خودی سے
ہمارے بس میں ہے ہم تختیاں بناتے ہیں

کہ واپسی کے سفر کا خیال ہی نہ رہے
تو چلو مل کے ہم کشتیاں جلاتے ہیں
کہ دیکھ لیں گے لکھنے کا فن وہ خودی سے
ہمارے بس میں ہے ہم تختیاں بناتے ہیں

[illegible]
۸ مئی 2020