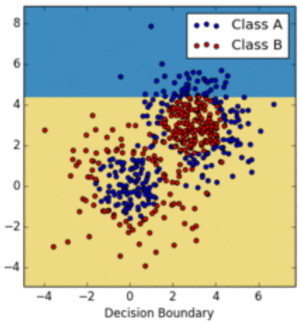
CS 236756 - Technion - Intro to Machine Learning

Tal Daniel

Tutorial 11 - Boosting & Bagging



[Image Source (https://web.eecs.umich.edu/~jabernet/eecs598course/fall2013/web/)](https://web.eecs.umich.edu/~jabernet/eecs598course/fall2013/web/)

Agenda

In [15]:
```python
# imports for the tutorial
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib notebook

from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Ensemble Learning

- **Wisdom of the Crowd** - assembling the predictions of a group of predictors (such as classifiers or regressors) often results in a better prediction than with the best individual predictor.
- **Ensemble** - a group of predictors. An *Ensemble Learning* algorithm is called an **Ensemble method**.
  - For example: **Random Forest** -train a group of Decision Tree classifiers, each is trained on a random subset of the training set. To make predicitons, we obtain the predicitons of all individual trees, and then predict the class that gets the most votes. This is one of the most powerful ML algorithms available today.

Voting Classifiers

- **Hard Voting Classifier** - aggregate the predictions of each classifier and predict the class that gets the most votes.
  - In fact, even if each classifier is a *weak learner* (it does only slightly better than random guessing), the ensemble can still be a *strong learner* (achieving high accuracy), provided there are a sufficient number of weak learners and they are sufficiently diverse.
  - **The Law of Large Numbers** - how can the above fact be explained? building an ensemble containing 1,000 classifiers that are individually correct only 51% of the time (slighly better than random guessing) and predict the majority voted class, it is possible to reach 75% accuracy if all the classifiers are perfectly independent (which is not really the case since they are trained on the same data).
  - One way to get diverse classifiers is to train them using very different algorithms (increases the chance that they will make very different types of erros and thus improving the ensemble's accuracy).
- **Soft Voting Classifier** - if all the classifiers are able to estimate class probabilities, then the class probability can be averaged over all the individual classifiers.
  - It often achieves higher performance than *hard voting* because it gives more weight to highly confident votes.

In [2]:

```
# let's load the cancer dataset, shuffle it and  speratre into train and test set
dataset = pd.read_csv('./datasets/cancer_dataset.csv')
# print the number of rows in the data set
number_of_rows = len(dataset)
print("total samples: {}".format(number_of_rows))
total_positive_samples = np.sum(dataset['diagnosis'].values == 'M')
print("total positive sampels (M): {}, total negative samples (B): {}".format(total_positive_samples, number_of_row
s - total_positive_samples))
num_train = int(0.8 * number_of_rows)
# reminder, the data looks like this
# dataset.head(10) # the dataset is ordered by the diagnosis
dataset.sample(10)
```

```
total samples: 569
total positive sampels (M): 212, total negative samples (B): 357
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
237	883263	M	20.48	21.46	132.50	1306.0	0.08355	0.08348	0.09042
159	871149	B	10.90	12.96	68.69	366.8	0.07515	0.03718	0.00309
442	90944601	B	13.78	15.79	88.37	585.9	0.08817	0.06718	0.01055
283	8912280	M	16.24	18.77	108.80	805.1	0.10660	0.18020	0.19480
477	911673	B	13.90	16.62	88.97	599.4	0.06828	0.05319	0.02224
45	857010	M	18.65	17.60	123.70	1076.0	0.10990	0.16860	0.19740
127	866203	M	19.00	18.91	123.40	1138.0	0.08217	0.08028	0.09271
561	925311	B	11.20	29.37	70.67	386.0	0.07449	0.03558	0.00000
120	865137	B	11.41	10.82	73.34	403.3	0.09373	0.06685	0.03512
444	9110127	M	18.03	16.85	117.50	990.0	0.08947	0.12320	0.10900

10 rows × 33 columns

In [3]:

```
# prepare the dataset
# we will take the first 2 features as our data (X) and the diagnosis as labels (y)
x = dataset[['radius_mean', 'texture_mean', 'concavity_mean']].values
y = dataset['diagnosis'].values == 'M'  # 1 for Malignat, 0 for Benign
# shuffle
rand_gen = np.random.RandomState(0)
shuffled_indices = rand_gen.permutation(np.arange(len(x)))
```

In [4]:

```
x_train = x[shuffled_indices[:num_train]]
y_train = y[shuffled_indices[:num_train]]
x_test = x[shuffled_indices[num_train:]]
y_test = y[shuffled_indices[num_train:]]
```

In [5]:

```
# pre-process - standartization
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

print("total training samples: {}, total test samples: {}".format(num_train, number_of_rows - num_train))
```

```
total training samples: 455, total test samples: 114
```

In [9]:
```
# hard voting
random_state = 38
# create different classifiers
log_clf = LogisticRegression(random_state=random_state, solver='lbfgs')
rnd_clf = RandomForestClassifier(random_state=random_state, n_estimators=100)
svm_clf = SVC(random_state=random_state)
# create a voting classifier
voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)], voting='hard')
# voting_clf.fit(x_train, y_train)
```

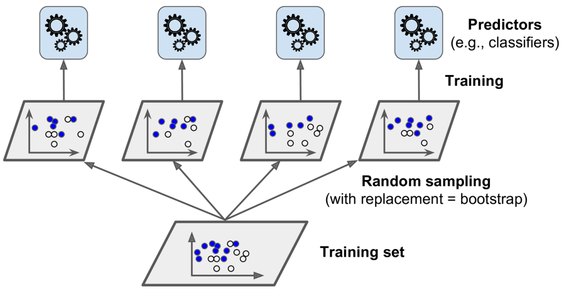
In [10]:
```
# let's look at each classifier's accuracy on the test set
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.9385964912280702
RandomForestClassifier 0.9298245614035088
SVC 0.9473684210526315
VotingClassifier 0.9473684210526315
```

Bagging (& Pasting)

- Another approach to get a diverse set of classifiers is to use the **same training algorithm** for every predictor, but to train them on **different random subsets of the training set**.
- When sampling is performed **with replacement** this method is called **bagging** (which is a short for *bootstrap aggregating*).
  - In sampling **with replacement**, each sample unit of the population can occur one or more times in the sample.
  - In statistics, resampling with replacement is called *bootstrapping*.
- When sampling is performed **without replacement** this method is called **pasting**.
- Thus, both bagging and pasting allow training instances to be sampled several times across multiple predictors, but only bagging allows training instances to be sampled several times for the same predictor.

- Illustartion:



[Image Source (https://github.com/SoojungHong/MachineLearning/wiki/Random-Forest)](https://github.com/SoojungHong/MachineLearning/wiki/Random-Forest)

- Once all predictors are trained, the ensemble can make a prediction for a new instance by collecting all the predictions of all the predictors. It usually decided by *hard voting* or average for regression.
- Each individual predictor has a higher bias than if it were trained on the original training set, but the aggregation **reduces both bias and variance**.
  - It is common to see that the ensemble has a **similar bias** but a **lower variance** than a single predictor trained on the original training set.

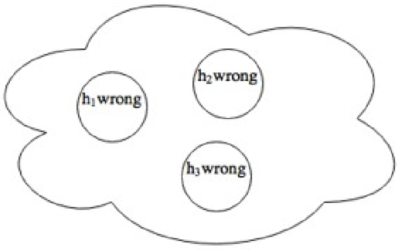
Bootstrap

- **Bootstrap Algorithm**:
  - Denote the original sample: $L_N = (x_1, x_2, \dots, x_N)$
  - Repeat M times:
    - Generate a sample L_k of size k from L_N by sampling *with replacement*.
    - Compute h from L_k (that is, train a predictor h using L_k).
  - Denote the bootstrap values $H = (h^1, h^2, \dots, h^M)$
    - Use these values for calculating all the quantities of interest.

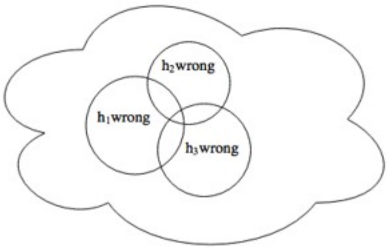
- **Bagging**:
  - Train each model with a random training set (bootsrap).
  - Each model in the ensemble has an **equal weight** in the voting.
  - Finally:

$$H(x) = sign(h^1(x) + h^2(x) + \ldots + h^M(x))$$

- One classifier can be wrong as long as the others are correct (*hard voting*)



- Since given equal weight, this may cause problems when there is overlap.



In [13]:
```
# bagging
# note: BaggingClassifiers will automatically perform 'soft voting' instead of 'hard voting'
# if the base classifier can estimate class probabilities (i.e. if it has a "predict_proba()" method).

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(),
    n_estimators=500,
    max_samples=100,
    bootstrap=True,
    n_jobs=1)
bag_clf.fit(x_train, y_train)
y_pred = bag_clf.predict(x_test)
bag_acc = accuracy_score(y_test, y_pred)
print("bagging accuracy: {:.3f}".format(bag_acc))
```

```
bagging accuracy: 0.939
```

In [14]:
```
# pasting
pas_clf = BaggingClassifier(
    DecisionTreeClassifier(),
    n_estimators=500,
    max_samples=100,
    bootstrap=False,
    n_jobs=1)
pas_clf.fit(x_train, y_train)
y_pred = pas_clf.predict(x_test)
pas_acc = accuracy_score(y_test, y_pred)
print("pasting accuracy: {:.3f}".format(pas_acc))
```

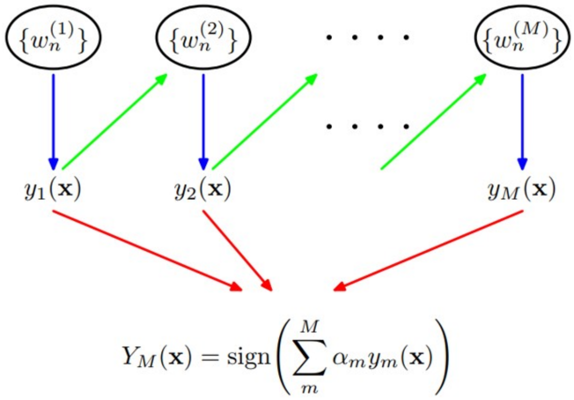
```
pasting accuracy: 0.930
```

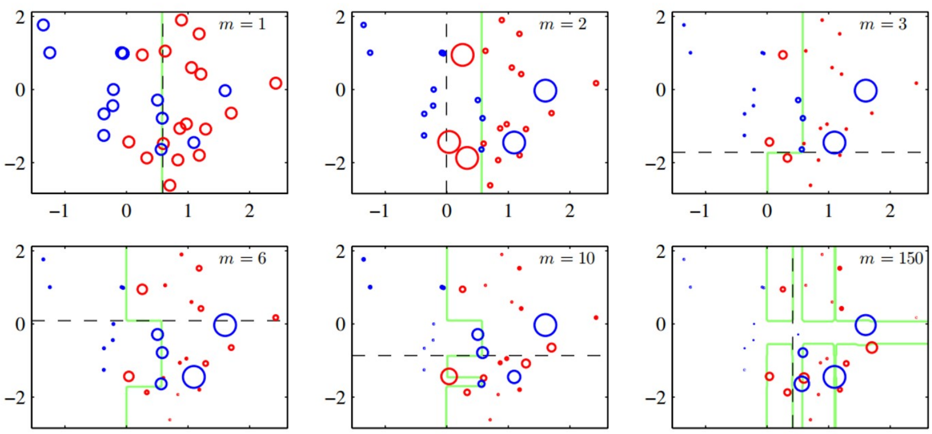
Boosting

- **Boosting** (also *hypothesis boosting*) - any Ensemble method that can combine several weak learners into a strong learner. In boosting methods, predictors are trained **sequentially**, each trying to correct its predecessor.
  - Weak Learner - as before, the error rate is slighty better than flipping a coin
  - We also define:
    - h is binary classifier such that $h \in \{-1, 1\}$
    - Error rate $Err \in [0, 1]$
- The principal difference between boosting and the committe methods is that in boosting, the base classifiers are **trained in sequence**.

- Each base classifier is trained using a **weighted form of the dataset**, in which the weight coefficient associated with each data point depends on the performance of the previous classifiers.
  - In particular, points that are misclassified by one of the base classifiers are given greater weight when used to train the next classifier in the sequence.
- Once all the classifiers have been trained, their predictions are then combined through a **weighted majority voting** scheme.

- Visually:





- From "Pattern Recognition and Machine Learning", Bishop, 2006

- There are many boosting methods, but we will examine one of the most popular one called *AdaBoost*.

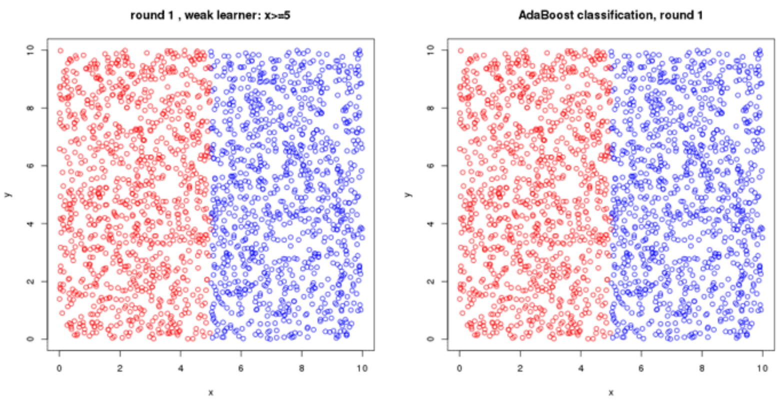
AdaBoost

- The idea of AdaBoost is to give more attention to training instances that the predecessor underfitted. This leads to a predictor that focuses more and more on the hard cases.
- The sequential learning in Boosting seems similar to Gradient Descent, only in AdaBoost predictors are added to the ensemble in order to make it better where in GD, a single predictor's paramerters are optimized to minimize an objective function.
- Once all predictors are trained, the ensemble makes predictions by assigning different weights to each predictor, depending on their **overall accuracy on the weighted training set**.

Definitions

- Class labels are $\{-1, 1\}$
- m - number of samples in the training dataset
- The weighted error rate of the t^{th} predictor:
$$\epsilon_t = \sum_{i=1}^{m} w^{(i)} \cdot 1(\hat{y}_t^{(i)} \neq y^{(i)})$$
  In the more general case where the weights are not normalized to 1:
$$\epsilon_t = \frac{\sum_{i=1}^{m} w^{(i)} \cdot 1(\hat{y}_t^{(i)} \neq y^{(i)})}{\sum_{i=1}^{m} w^{(i)}}$$
  - $\hat{y}_t^{(i)}$ is the t^{th} predictor's prediction for the i^{th} instance.

- The predictors weight of the t^{th} predictor:
$$\alpha_t = \eta \ln \frac{1 - \epsilon_t}{\epsilon_t}$$
  - η it the learning rate hyperparameter, e.g. $\frac{1}{2}$ or 1.
  - The more accurate the predictor is, the more weight the predictor will be given.
- The update rule: for $i = 1, 2, \dots, m$
$$w^{(i)} \leftarrow \begin{cases} w^{(i)} e^{-\alpha_t} & \text{if } \hat{y}_t^{(i)} = y^{(i)} \\ w^{(i)} e^{\alpha_t} & \text{if } \hat{y}_t^{(i)} \neq y^{(i)} \end{cases} = w^{(i)} e^{-\alpha_t \cdot y^{(i)} \cdot \hat{y}_t^{(i)}}$$
  - Once all the weights were calculated, they are summed. The sum is denoted Z_t. Then, all the weights are normalized by dividing each weight by Z_t.
- **Stopping criteria**:
  - The desired number of predictors is reached.
  - A perfert predictor is found.



[Image Source (http://talimi.se/ml/adaboost/)](http://talimi.se/ml/adaboost/)

- **The AdaBoost Algorithm**:
  - Initialize the data weights coefficients $\{w^{(i)}\}_{i=1}^m$:

$$w^{(i)} = \frac{1}{m}, \forall i = 1, 2, \ldots, m$$

  - For $t = 1, \ldots, T$:
    - Fit a weak classifier $h_t(x)$ (which makes predictions $\hat{y}_t$) to the weighted training data and calculate the weighted error rate:

$$\epsilon_t = \frac{\sum_{i=1}^m w^{(i)} \cdot 1(\hat{y}_t^{(i)} \neq y^{(i)})}{\sum_{i=1}^m w^{(i)}}$$

    - Choose α_t (default $\eta = \frac{1}{2}$):

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

    - Update the weights: for $i = 1, 2, \ldots, m$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} e^{-\alpha_t} & \text{if } \hat{y}_t^{(i)} = y^{(i)} \\ w^{(i)} e^{\alpha_t} & \text{if } \hat{y}_t^{(i)} \neq y^{(i)} \end{cases} = w^{(i)} e^{-\alpha_t \cdot y^{(i)} \cdot \hat{y}_t^{(i)}}$$

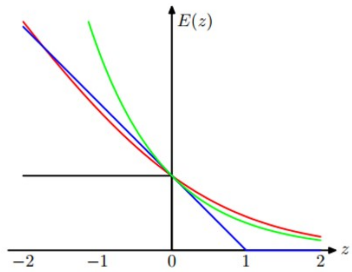
    - Normalize the weights: for $i = 1, 2, \ldots, m$

$$w^{(i)} \leftarrow \frac{w^{(i)}}{Z_t}$$

      - $Z_t = \sum_{i=1}^m w^{(i)}$
  - Use predictions using the final model, which is given by:

$$H(x) = sign(\sum_{i=1}^T \alpha_t h_t(x))$$

Exponential Loss

- So far, the loss functions we have seen:
  - 0-1 loss
  - Hinge loss
  - Log loss
- Unlike previously learnt classifiers, AdaBoost minimzes the exponential loss.
- All lossess upper bound the 0-1 loss and act as differentiable surrogate loss functions.
- 

- Optimizing the exponential loss:
  - As shown in class, the training error is upper bounded by H:

$$\frac{1}{m} \sum_i^m 1(H(x_i) \neq y_i) \leq \prod_{t=1}^T Z_t$$

    - $Z_t = \sum_i w_t^{(i)} e^{-\alpha_t y_i h_t(x_i)}$
  - At each round we minimize Z_t by:
    - Choosing the optimal h_t
    - Finding the optimal α_t
    -

$$\frac{dZ}{d\alpha} = -\sum_{i=1}^m w^{(i)} y_i h(x_i) e^{-\alpha y_i h(x_i)} = 0$$
$$-\sum_{i:y_i = h(x_i)} w^{(i)} e^{-\alpha} + \sum_{i:y_i \neq h(x_i)} w^{(i)} e^{\alpha} = 0$$
$$-e^{-\alpha}(1 - \epsilon) + e^{\alpha} \epsilon = 0$$
$$\rightarrow \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

Boosting (AdaBoost) Example By Hand

Moses is a student who wants to avoid hard courses.

In order to achieve this he wants to build a classifier that classifies courses as "easy" or "hard".

He decides to classify courses' hardness by using AdaBoost with decision trees stumps (decision trees with max depth of 1) on the following data:

Course ID	Hard	Final Exam	Theoretical	Midterm	236*	Number of HW
1	Y	Y	N	Y	N	5
2	Y	N	Y	Y	N	5
3	Y	N	Y	N	Y	1
4	Y	N	Y	N	N	3
5	Y	N	Y	N	N	5
6	Y	Y	N	Y	N	5
7	Y	Y	N	Y	N	5
8	N	N	N	Y	Y	1
9	N	N	Y	N	N	1
10	Y	N	N	N	N	5

As a first step, he first determined for each possible classifier (including the trivial constant classifier), which of the data points were misclassfied.

For example, for the first classifier which classfies courses as hard if they have a final exam, the classifier is wrong on samples 2,3,4 and 5.

Classifier	Test	Value	Misclassified
A	Final Exam	Y	2,3,4,5
B	Theoretical	Y	1,6,7,9
C	Midterm	Y	3,4,5,8
D	Undergrduate	Y	1,2,4,5,6,7,8
E	# HW > 2	Y	3,10
F	# HW > 4	Y	3,4,10
G	True (const)		8,9,10
H	Final Exam	N	1,6,7,8,9,10
I	Theoretical	N	2,3,4,5,8,10
J	Midterm	N	1,2,6,7,9,10
K	Undergraduate	N	3,9,10
L	# HW < 2	Y	1,2,4,5,6,7,8,9
M	# HW < 4	Y	1,2,5,6,7,8,9
N	False (const)		1,2,3,4,5,6,7

Consider only useful classifiers

Only 6 classifiers from the table above would ever be used because the other 8 make all the same error as one of the other classifiers and then make additional erros. For example, classifiers I and N do the same mistakes as A and add to that. The 6 useful classifiers are:

Classifier	Test	Value	Misclassified
A	Final Exam	Y	2,3,4,5
B	Theoretical	Y	1,6,7,9
C	Midterm	Y	3,4,5,8
D	Undergrduate	Y	1,2,4,5,6,7,8
E	# HW > 2	Y	3,10
G	True (const)		8,9,10

AdaBoost

- We will now perform AdaBoost by calculating the weights at each iteration.
- We will calculate the 10 weights, the classification h, the error and α.
- If there is a tie, we break it by choosing the classifier that is higher on the list (lexicographical order)
- Note: in this example we assume that the weights of the data points do not affect the clasification and are just meant to calculate the final weight of each classifier.

Round 1

- Each weight is given the same value: $\frac{1}{m} = \frac{1}{10}$
- Since classifier E is the most accurate, it will serve as the classifier.
- The weight error rate of classifier E is $\epsilon_E = \frac{2}{10}$
- Thus: $\alpha_E = \frac{1}{2}\ln\frac{1-\epsilon_E}{\epsilon_E} = \frac{1}{2}\ln(4)$

Parameters	Round 1	Round 2	Round 3
w1	$\frac{1}{10}$		
w2	$\frac{1}{10}$		
w3	$\frac{1}{10}$		
w4	$\frac{1}{10}$		
w5	$\frac{1}{10}$		
w6	$\frac{1}{10}$		
w7	$\frac{1}{10}$		
w8	$\frac{1}{10}$		
w9	$\frac{1}{10}$		
w10	$\frac{1}{10}$		
h	E		
Err - ϵ	$\frac{2}{10}$		
$\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}$	$\frac{1}{2}\ln(4)$		

AdaBoost - calculating the new weights

- Recall that the un-normalized weights update:

$$\tilde{w}_{t+1}^{(i)} = w_t^{(i)} e^{-\alpha_t y_i h_t(x_i)}$$

- For the correctly classified data points (8 points):

$$\tilde{w}_{t+1}^{(i)} = \frac{1}{10} e^{-\frac{1}{2}\ln(4)} = \frac{1}{10} \cdot \frac{1}{2} = \frac{1}{20}$$

- For the incorrectly classified data points (2 points):

$$\tilde{w}_{t+1}^{(i)} = \frac{1}{10} e^{\frac{1}{2}\ln(4)} = \frac{1}{10} \cdot 2 = \frac{1}{5}$$

- Calculate the normalization factor:

$$Z_t = 8 \cdot \frac{1}{20} + 2 \cdot \frac{1}{5} = \frac{4}{5}$$

- The final weights after normalization:
  - Correct: $w_{t+1}^{(i)} = \frac{1}{20} \cdot \frac{5}{4} = \frac{1}{16}$
  - Incorrect: $w_{t+1}^{(i)} = \frac{1}{5} \cdot \frac{5}{4} = \frac{1}{4}$

Similarly, we fill in the rest of the table:

Parameters	Round 1	Round 2	Round 3
w1	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w2	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w3	$\frac{1}{10}$	$\frac{4}{16}$	$\frac{4}{24}$
w4	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w5	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w6	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w7	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w8	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w9	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w10	$\frac{1}{10}$	$\frac{4}{16}$	$\frac{4}{24}$
h	E	B	A
Err - ϵ	$\frac{2}{10}$	$\frac{1}{4}$	$\frac{7}{24}$
$\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}$	$\frac{1}{2}\ln(4)$	$\frac{1}{2}\ln(3)$	$\frac{1}{2}\ln\frac{17}{7}$

AdaBoost - Putting the classifiers together

- The final classifier for 3 rounds of Boosting:

$$H(x) = sign(\frac{1}{2}\ln(4) \cdot h_E(x) + \frac{1}{2}\ln(3) \cdot h_B(x) + \frac{1}{2}\ln\frac{17}{7} \cdot h_A(x))$$

  - $h_c(x)$ returns +1 or -1 for $c = E, B, A$
- The data points that the final classifier is correct about them:
  - Since $\alpha_E, \alpha_B > \alpha_A$ - it is just a *majority vote*
  - Only one example (3) is misclassified

AdaBoost in Scikit-Learn

- Scikit-Learn uses a multiclass version of AdaBoost called *SAMME* (Stagewise Additive Modeling using a Multiclass Exponential loss function).
  - When there are just 2 classes, SAMME is equivalent to AdaBoost.
  - If the predictors can estimate class probabilities (i.e. they have a `predict_proba()` method), Scikit-Learn can use a variant of SAMME called *SAMMER* (R for "Real"), which relies on class probabilities rather than predictions and generally performs better.
- The following code trains an AdaBoost classifier on 600 Decision Stumps.
- Note: if the AdaBoost classifier is **overfitting** the training set, a good regularization may be reducing the number of estimators or more strongly regularize the base classifier.
- An important drawback to sequential learning is that **it cannot be parallelized**, since each predictor can only be trained after the previous predictor has been trained and evaluated. Thus, it does not scale as well as bagging or pasting.

In [16]:
```
# AdaBoost
ada_clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=600, algorithm="SAMME.R", learning_r
ate=0.5)
ada_clf.fit(x_train, y_train)
y_pred = ada_clf.predict(x_test)
ada_acc = accuracy_score(y_test, y_pred)
print("adaboost accuracy: {:.3f}".format(ada_acc))
```

```
adaboost accuracy: 0.930
```

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Simple Ensemble, Mixture of Experts - Ensembles (1): Basics (https://www.youtube.com/watch?v=Yvn3--rIdZg)
- Bagging - Ensembles (2): Bagging (https://www.youtube.com/watch?v=Rm6s6gmLTdg)
- Boosting, AdaBoost - Machine Learning Lecture 34 "Boosting / Adaboost" -Cornell CS4780 (https://www.youtube.com/watch?v=toOAToTaGV4)
  - MIT - 6.034 Artificial Intelligence - Learning: Boosting (https://www.youtube.com/watch?v=UHBmv7qCey4)
  - Ensembles (4): AdaBoost (https://www.youtube.com/watch?v=ix6IvwbVpw0)

Credits

- Icons from Icon8.com (https://icons8.com/) - https://icons8.com (https://icons8.com)
- Datasets from Kaggle (https://www.kaggle.com/) - https://www.kaggle.com/ (https://www.kaggle.com/)
- Examples and code snippets were taken from "Hands-On Machine Learning with Scikit-Learn and TensorFlow" (http://shop.oreilly.com/product/0636920052289.do)