## Comparing the 3 friends

- Friend #1: What if we ask this friend a lot of yes-no questions? We'll always get the wrong answer so all we have to do is do the exact opposite. In this way, the complete liar gives us a lot of information.
- Friend 2: In contrast, the one who lies half of the time doesn't really help us much since we have no idea if we should believe them or not.
- Friend 3: Obviously a truthful one is super-helpful too.

So we conclude that the worst one is this one in the middle.

## The models

In the same way, we'll have our three models.

- Model one is the truthful one which is always correct
- Model two is a random one which is correct roughly half of the time
- Model three is a liar which is always wrong

We'll assign the truthful model a large positive weight, the random model weight of zero since it's useless, and the liar model a large negative weight since we'll do the exact opposite as this model says.

## The math behind this

We want our weight function to be very positive for the truthful models, zero for the useless models, and very negative for the liar models. Let's look at the accuracy :

- The truthful model has accuracy around one
- The random model has an accuracy of around 50 percent
- The liar model is accurate around zero.

So this function will help us check it out:
$y = ln(x / 1 - x)$ where x is the accuracy.

- It's very negative for values of x close to 0
- It's close to 0 for values around .5
- for example, for 0.5 it's
- $ln(0.5 / 0.5) = ln(1) = 0$
- It's very positive for values of x close to 1.

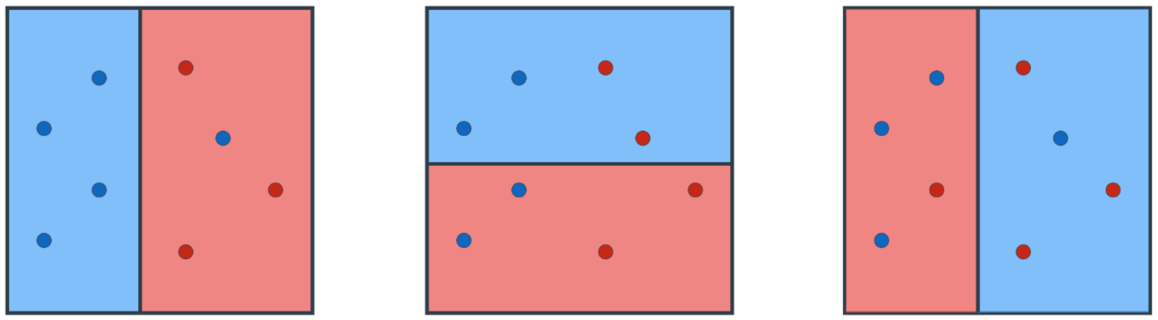
There are actually much heavier mathematical reasons for this to be the function but that's outside the scope of this course. Don't worry, for now, about potential division by zero, we'll deal with that in a bit.

We conclude that a great formula for weight is this,
$y = ln(accuracy / 1 - accuracy\ )$

## Quiz

So a small quiz, can you find the weights for these three models over here?

$$weight = \ln\left(\frac{accuracy}{1 - accuracy}\right)$$



## Weight quiz 1

Calculate the weight of the first model, with 2 significant digits.

___ln(7/1)=1.95____________________________________________________

## Weight quiz 2

Calculate the weight of the second model.

___ln(4/4)=0____________________________________________________

## Weight quiz 3

Calculate the weight of the third model, with 2 significant digits.

___ln(2/6)= -1.099____________________________________________________

# AdaBoost in sklearn

Building an AdaBoost model in sklearn is no different than building any other model. You can use scikit-learn's `AdaBoostClassifier` class. This class provides the functions to define and fit the model to your data.

```
>>> from sklearn.ensemble import AdaBoostClassifier
>>> model = AdaBoostClassifier()
>>> model.fit(x_train, y_train)
>>> model.predict(x_test)
```

In the example above, the `model` variable is a decision tree model that has been fitted to the data `x_train` and `y_train`. The functions `fit` and `predict` work exactly as before.

## Hyperparameters

When we define the model, we can specify the hyperparameters. In practice, the most common ones are

- `base_estimator`: The model utilized for the weak learners (**Warning:** Don't forget to import the model that you decide to use for the weak learner).
- `n_estimators`: The maximum number of weak learners used.

For example, here we define a model which uses decision trees of max_depth 2 as the weak learners, and it allows a maximum of 4 of them.

```
>>> from sklearn.tree import DecisionTreeClassifier
>>> model = AdaBoostClassifier(base_estimator =
DecisionTreeClassifier(max_depth=2), n_estimators = 4)
```

# Quiz: Weighting the Models

In order to combine models, we'll actually weight them by how well they're doing. In this way, if one of our friends just answers the questions better, we'll believe that friend more. So let's turn to a hypothetical question. We have three friends.

- One always tells the truth
- Another one tells the truth half of the time and lies half of the time, and we don't know which half
- The third one always lies.

The question is, which one of these three is the one that we should ignore the most? Equivalently, here we have three models. The first one is always correct, the second one is correct half of the time, and the third one is wrong every single time.

## Quiz Question

Which of the three models is the worst, in terms of giving us information?

a. The one that always tells the truth
b. The one that tells the truth half of the time
c. The one that always lies

## Combining values

- For the positive area, we'll add the weight
- For the negative area, we'll subtract the weight

We take the first model and add the positives and negatives of the weight to each region. Then, we do the same thing with the second model and finally, with the third model. When the value is positive, then we say blue, and when it is negative, we say red.

That's the combination of our weak learners and we can see how it forms a strong learner.

### Quiz Question

The positive weighted regions and negative weighted regions both represent what in the model?

a. The highest positive and negative values are subtracted from each other. That number should be zero.
b. They each represent one of the prediction classes

## The math behind AdaBoost

Let's assign to each data point an initial weight of one.

- Now, let's fit our **first** Learner. Before we wanted to minimize the number of errors. Now we want to minimize the sum of weights of the incorrectly classified points which as of now is the same. If we add the weights of the correctly classified points, we get a 7 and 3 for the incorrectly classified points.

- So, let's weigh the incorrectly classified points a bit more. How much? Well, let's make it enough to have this model be a 50-50 model. So, if we take these three points and weight them by a factor of 7/3 or 2.33, our model now has 7 as the sum of the weights of the correctly classified points and also 7 as the sum of the weights of the incorrectly classified points. So, this model now is lame, and we need a second one.

- Our **second** model will be this one that fits the newly weighted data best. For this one, we can see that if we add the weights of the correctly classified points, we get 11, and for the incorrectly classified ones, we get 3 again. So, let's weigh the incorrectly classified points. By how much?

- Well, enough to make this model a 50-50 model again. Notice that if we weigh these three errors each by a factor of 11/3 or 3.66, the sum of the weights of the incorrectly classified points is 11. So, our model is just a 50-50 model again. That's lame. Let's fit a third model in this newly weighted data.

- Our **third** model has the added weight of the correctly classified points? Now it's 19 and still 3 for the incorrect ones.

## Quiz Question

Why does AdaBoost apply weighting to misclassified points?

a. In order to have these points removed from the dataset since they are misclassified
[✓ b. In subsequent models, classifiers can focus on the misclassified samples more.]
c. All weights between the prediction classes should equal zero.

## Boosting with ADABOOST

Discovered by Freund and Schapire in 1996.

- We fit our **first** learner in order to maximize accuracy, or equivalently, minimize the number of errors. There are a few good ones, but one can check that we can do no better than three errors.
- The **second** learner needs to fix the mistakes that the first one has made. We take the misclassified points and make them bigger. In other words, we'll punish the model more if it misses these points. So, the next weak learner needs to focus on these more.
- Now again, we punish the points that are misclassified by the second learner by enlarging those points.
- Our **third** weak learner tries really hard to correctly classify the big points. We could keep going, but let's say three is enough.

Now, we want to combine the three models we found. I'll be more specific about combining them later, but for now, let's imagine that we're making them vote like before.
This is a bit vague on the details, but we will dive deeper in the next few videos.

So, What are the steps of the AdaBoost algorithm?

1. Maximize accuracy, minimize errors
2. Identify misclassified points from the previous step
3. Try to classify points identified in the previous step

Since our data may be huge, in general, we don't want to train many models on the same data. This would be very expensive. Instead, we'll just take subsets of it and train a weak learner on each one of these subsets. Then we'll figure out how to combine these learners. A weak learner in this case is one that splits the data either vertically or horizontally.

Notice that we never partition the data. We are completely allowed to repeat points among our subsets and to even not consider some of the points at all. At every step, we pick a fully random subset of data.

## How do we combine our weak learners?

By voting. We over impose each learner on the data. Because we have three learners in our example, if two or more of them predict blue, then that region is blue, and if two or more of them predict red, then that region is red. If we have an even number of models, we can pick any way we want to break ties. Although with lots of points and lots of models, it's hard to imagine that we would get a tie somewhere.

### Quiz Question

What is the purpose of voting in Bagging?

a. Voting is used to set the priority of weak learners.
b. Voting is the last step after training and is used to combine the weak learner results.

## The problem with decision trees

Let's say we have a large table with lots and lots of columns. So, we create our Decision Tree. And we end up with answers like the following.

If a client is a male between 15 and 25 in the US, on Android, in school, likes tennis, pizza, but does not like long walks on the beach, then they're likely to download Pokemon Go.

This is not good. This almost looks like the tree just memorized the data. It's overfitting. *Decision Trees tend to overfit a lot.*

In the continuous case, this can also happen. The Decision Tree has many nodes which end up giving us a complicated boundary that pretty much borders every point with a small square. *This is also overfitting as it doesn't generalize well to the data.*

How do we solve this?

Pick some of the columns randomly from our data. Build a Decision Tree in those columns. Now, pick some other columns randomly and build a Decision Tree in those, and do it again. When we have a new data point, say this person over here, we just let all the trees make a prediction and pick the one that appears the most.

For example, these trees decided that this person will download Snapchat, WhatsApp, and WhatsApp. So, the ensemble of trees will recommend *WhatsApp*. Since we used a bunch of trees on randomly picked columns, this is called a **random forest**.

# But, there are better ways to pick the columns than randomly.

### Quiz Question

What is a consequence of Decision Trees that overfit?

a. They do not generalize well
b. They are computationally heavy

# Ensembles

This whole lesson is about how we can combine (or ensemble) the models that we have already seen in a way that makes the combination of these models better at predicting than the individual models.

Commonly the "weak" learners you use are decision trees. In fact, the default for most ensemble methods is a decision tree in sklearn. However, this value can change to any of the models seen so far.

## Why Would We Want to Ensemble Learners Together?

There are two competing variables in finding a well-fitting machine learning model:

- **Bias**
- **Variance**.

It is common in job interviews for you to be asked about this topic and how it pertains to different modeling techniques.

**Bias**: When a model has a high bias, this means it doesn't do a good job of bending to the data. An example of an algorithm that usually has a high bias is linear regression. Even with completely different datasets, we end up with the same line fit to the data. When models have high bias, this is bad.

**Variance**: When a model has high variance, this means that it changes drastically to meet the needs of every point in our dataset. Linear models like the one above have low variance, but high bias. An example of an algorithm that tends to have high variance and low bias is a decision tree (especially decision trees with no early stopping parameters). A decision tree, as a high variance algorithm, will attempt to split every point into its own branch if possible. This is a trait of high variance, low bias algorithms - they are extremely flexible to fit exactly whatever data they see.

## Quiz Question

Which of the following descriptions are correct? (There is more than one correct answer)

a. When a model has a high bias, this means it doesn't do a good job of bending to the data.
b. When a model has a low bias, this means it doesn't do a good job of bending to the data.
c. When a model has low variance, this means that it changes drastically to meet the needs of every point in the dataset.
d. When a model has high variance, this means that it changes drastically to meet the needs of every point in our dataset.

By combining algorithms, we can often build models that perform better by meeting in the middle in terms of bias and variance. There are some other tactics that are used to combine algorithms in ways that help them perform better as well. These ideas are based on minimizing bias and variance based on mathematical theories, like the central limit theorem.

**Introducing Randomness Into Ensembles**

Another method that is used to improve ensemble methods is to introduce randomness into high variance algorithms before they are ensembled together. The introduction of randomness combats the tendency of these algorithms to overfit (or fit directly to the data available). There are two main ways that randomness is introduced:

- **Bootstrap the data** - that is, sampling the data with replacement and fitting your algorithm to the sampled data.
- **Subset the features** - in each split of a decision tree or with each algorithm used in an ensemble, only a subset of the total possible features are used.

In fact, these are the two random components used in the next algorithm you are going to see called **random forests**.

## Quiz Question

True or False. Introducing randomness into high variance algorithms is an effective method to improve ensemble methods (assuming done before they are ensembled together).

a. True
b. False