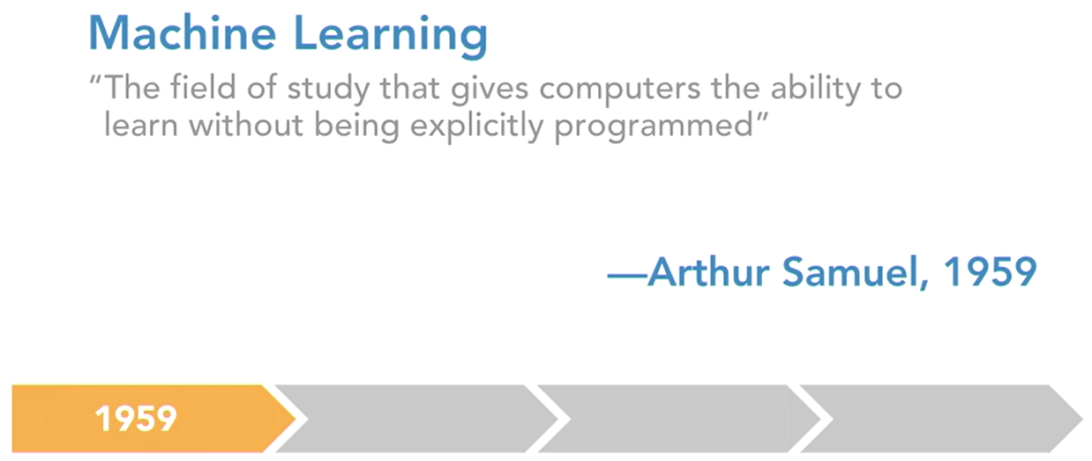
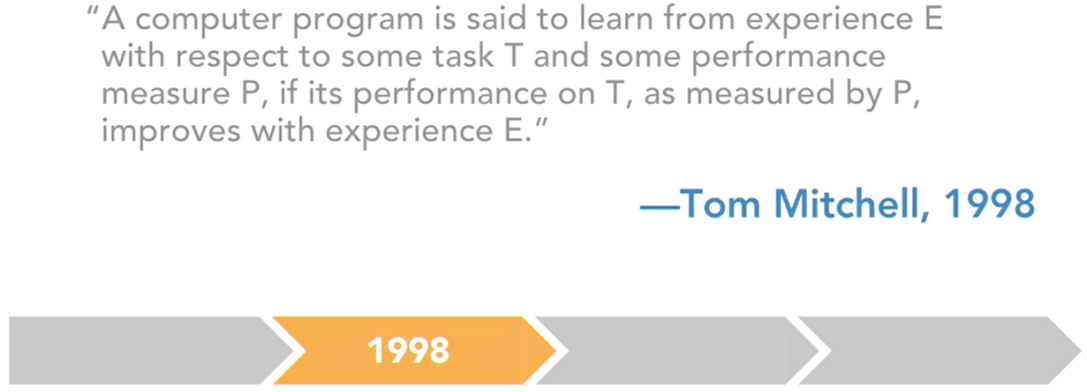
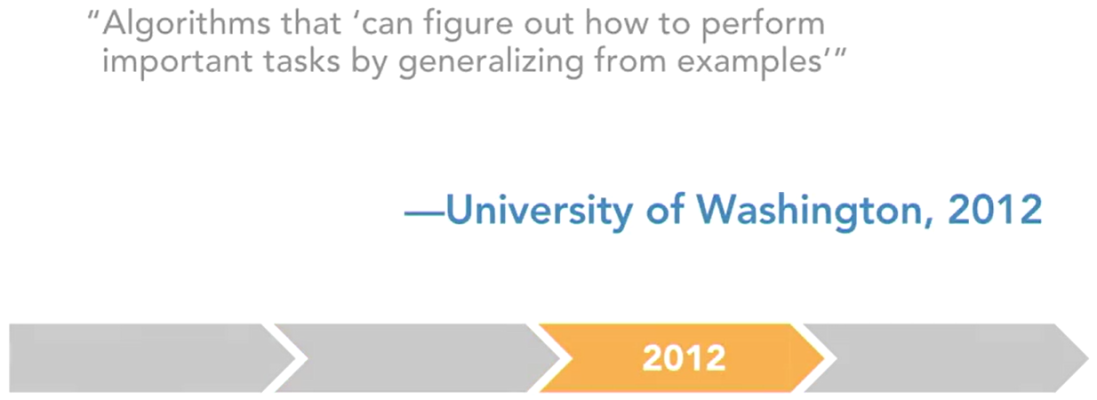
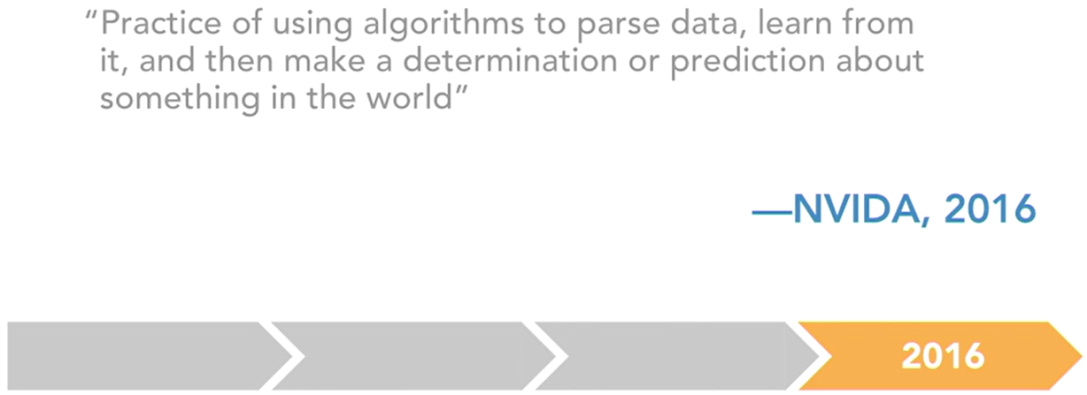
*Start a new Python project folder … do not reuse the previous workspace … As with any keyboard-driven console-like environment, developing muscle -memory for the common commands is also part of the learning curve.*

What is Machine Learning?





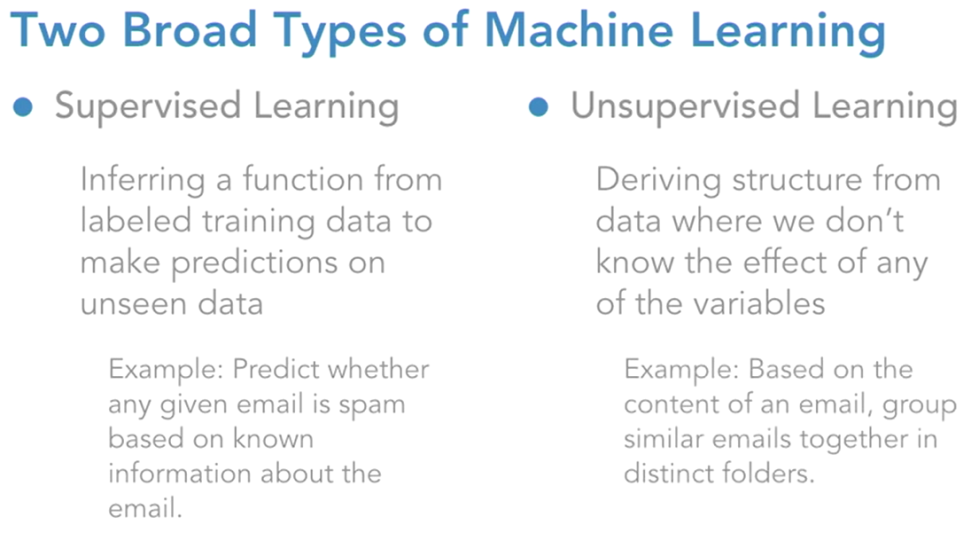




Since there is no clear, and agreed upon definition on what is machine learning, hopefully you can construct your own meaning out of the four definitions offered above.

So, develop your own idea based on idea based on the four definitions. There are two broad types of machine learning. The first, **supervised learning**. This is where you have an explicit label and it’s a task of inferring a function from that labeled training data to make a prediction on unseen data. An example of supervised learning would be a spam filter.

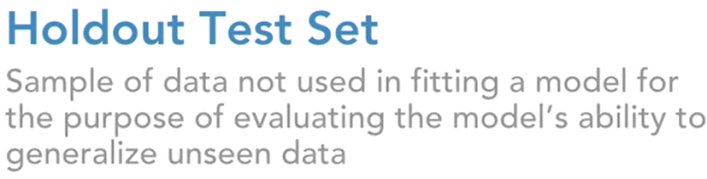
**Unsupervised learning** is where you don’t have any explicit labels and it’s about deriving structure from the data where you don’t know the effect of any of the features. In other words, you don’t have anything you’re trying to predict, necessarily. You are just trying to back out some sort of information or structure using the variables that you are given. An example of unsupervised learning would be grouping together similar emails into distinct folders based on the content.



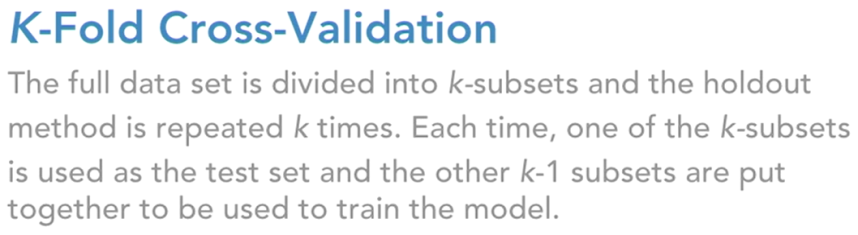
There is no right or wrong answer necessarily, but if a model can identify that these 15 emails are all in regards to a vacation to Italy and those other ones are about planning a certain family gathering, and then it can group those together into their own little bundles. That is what unsupervised learning is.

**Cross-validation and evaluation metrics**

There are several tools, tricks, methods, and metrics used to evaluate models. One of them is a holdout test set. This is a sample of data that is set aside and not used in any of the fitting of the model for the purpose of evaluating the model’s ability to generalize to unseen data.



So this is meant to simulate how the model will perform in real world scenarios. This is the entire point of building these models, to generalize and say something about the world. We will be using K-Fold Cross-Validation to evaluate our models. In this process the full data set is k-subsets and the holdout method is repeated k times.

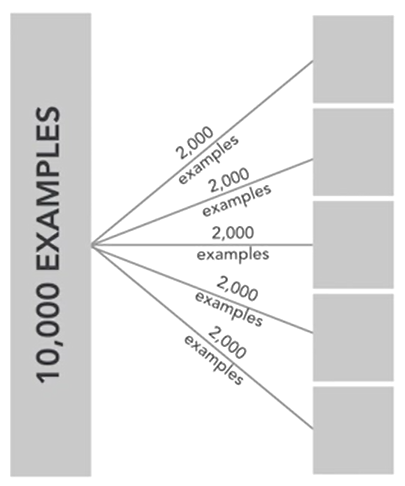


That is, in each iteration one of the k-subsets is treated as the holdout test set and the other k-1 subsets are put together to train the model. The purpose is that this gives you a little bit more robust read on the performance of the model rather than just having one single hold out test set for the model to be evaluated on. Now you have K test sets and k evaluation metrics to understand the potential performance outcomes. Let’s walk through an example of what cross validation actually looks like to make sure we are on the same page.

Example of Fivefold Cross-Validation

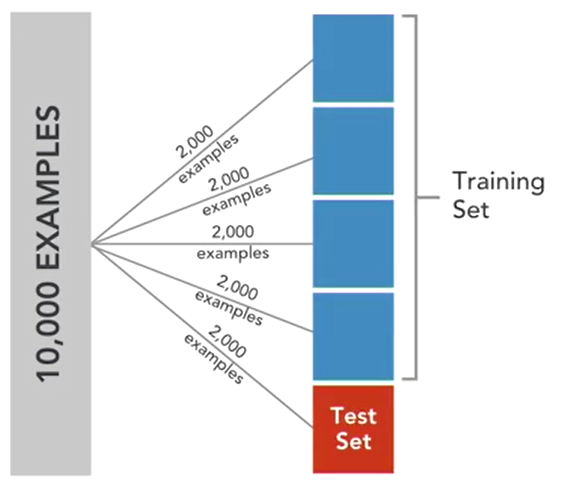


In this example, we will start with a full data set of 10,000 examples and we want to run five fold cross validation. That means k equals five. So the first step is to split that 10,000 example data set into k or five subsets.



So you’ll see that we now have five subsets of data and each has 2,000 examples. So this is sampling without replacement, so all 10,000 examples are still accounted for and it’s worth noting that these subsets will remain the same throughout this entire process.

So an example in subset one will remain in subset one all the way through the end of this cross validation.

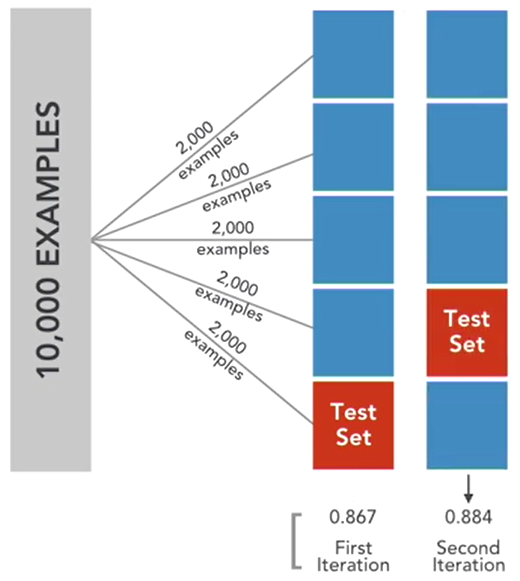


We assign one of these subsets as a test set (colored red block), which is subset five in red here and then we assign the other four that are in blue to the training set.

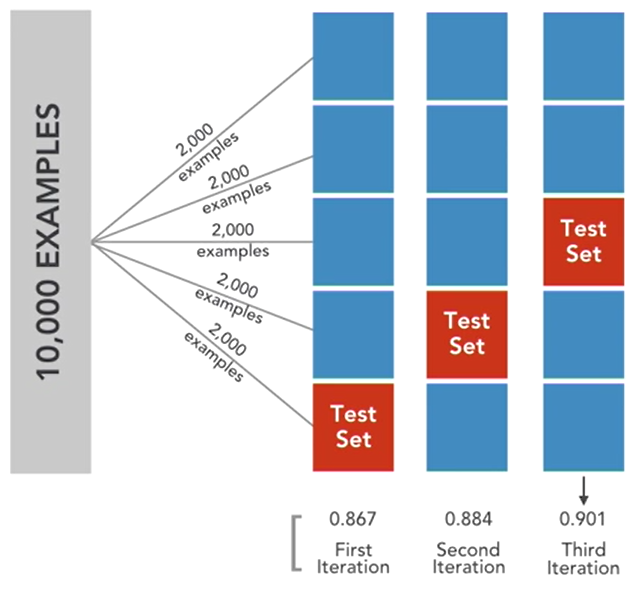


Now we will fit a model on the 8,000 examples in blue and then we will evaluate the model on the 2,000 example test set in red and then we will record the performance metric and store that away in an array. To be clear this is all handled under the hood in Psychic Learn.

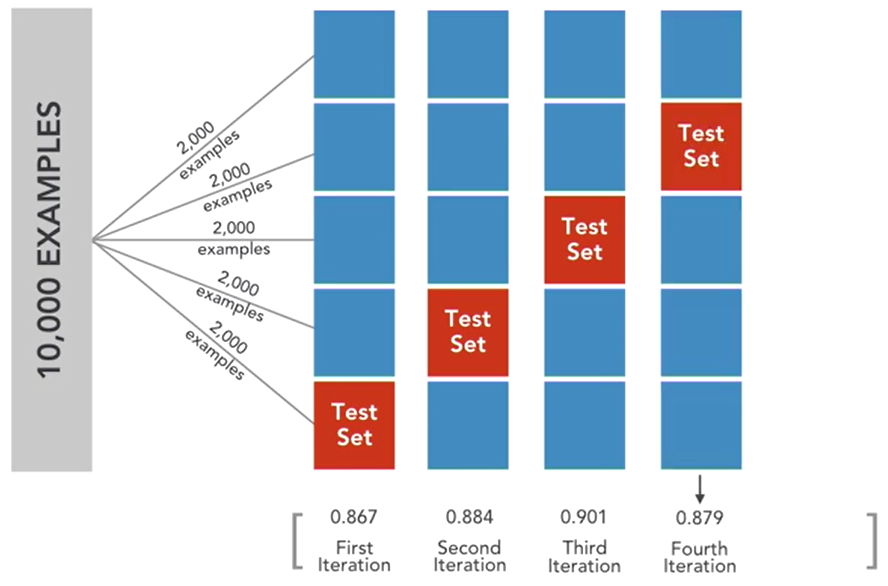
You don’t have to manually implement these steps. Psychic Learn handles all this but it’s important for you to understand what it is doing when it carries out this process. Again you will pick an evaluation metric and after the first iteration it will store the performance of the model on the holdout test set. Here we are saying that is 0.867. Next we will move on to the second iteration, where now the fourth subset is the test set and subsets one through three along with the fifth one are now the training set.



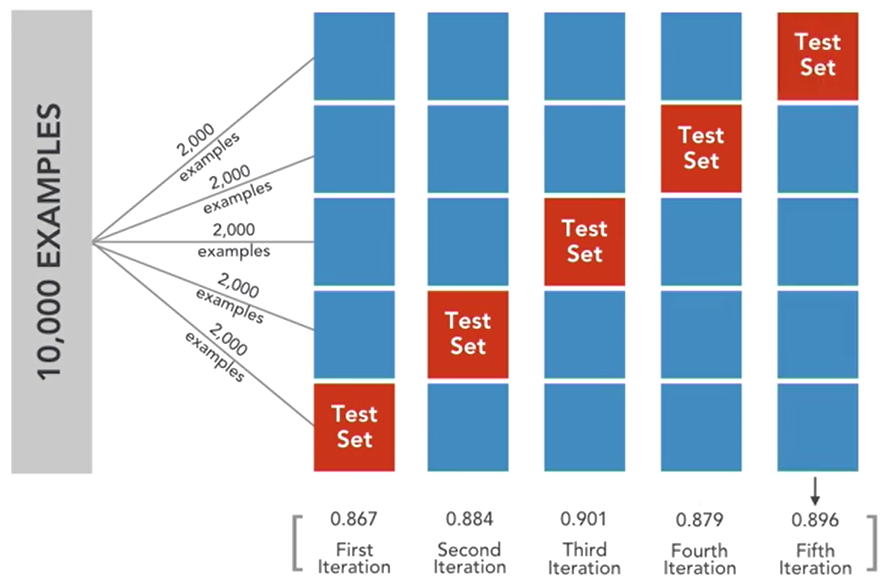
So again, we will re fit a brand new model on these 8,000 training examples and evaluate on the 2,000 test examples in that fourth subset. And then we will store the performance metric, here we are saying that is 0.884. Then again for the third iteration, the third subset will be our test set and then model will be trained on the 8,000 examples in the first, second, fourth and fifth.



So you train a brand new model evaluate it on the third subset, store the performance metric here we’ll say it’s 0.901. Again for the fourth iteration, same process as before but now the second subset is our holdout test set.

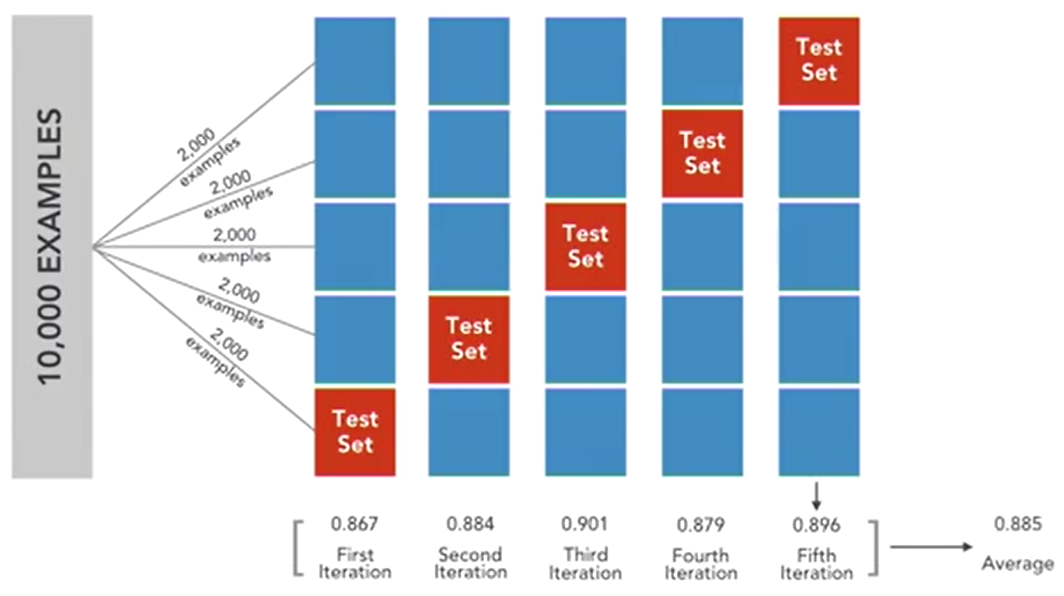


Last but not least, the first subset is the test set and two through five are the training data.



For the model, evaluate it on the first subset and store the evaluation metric. Now it is worth noting that at this point, every subset and thus every example has been used in a training set four times and in an evaluation set once. So we have now used this model configuration to fit a model on all different combinations of these examples and evaluated it on every single point in this data set.

So you can see why this can be a really powerful tool to gauge a model's ability to generalize. And then lastly, you would normally output maybe the full array of scores so all five scores or you might just output the simple average. So the average is .885. But now we see a more robust gauge of what the potential outcome of this model would be.



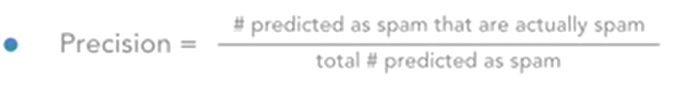
So let's just say we're using a single holdout test set and using that to gauge the performance of the model for our business. Now the first iteration you'll get a score of .867 but in average overall five is .885 and the highest is .901.

So that's a difference of .018 from the lowest score to the average and .34 from the lowest score to the highest score. That may not seem like a big difference but in a business setting where this could impact millions of dollars, that's a huge difference. So having a read on how this model performs over five different test sets is a huge advantage to truly understanding how your model might perform in the real world. Now this gives us more confidence that the model will perform around .885 and gives us a range of plausible outcomes or some error of ours on the projection.

So let's talk about actual evaluation metrics now. So for a classification problem like this spam ham data set that we're working with, we'll generally use three main performance metrics.



The first is accuracy, so that's just the number that you've predicted correctly over the total number of observations. So if you have 10,000 observations and you've got 800 of them labeled correctly then your accuracy is 80%.

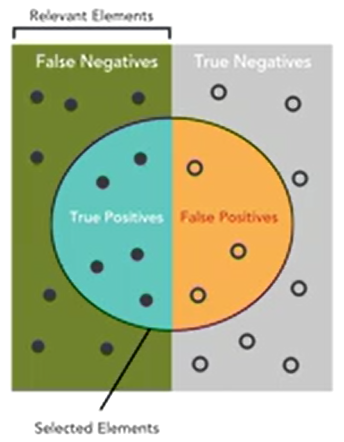


The second metric is precision; within the context of the problem that we're working with, that would be represented by the number that the model predicted as spam that are actually spam divided by the total number that the model predicted as spam.

And the last evaluation metric is called recall,



so that would be the number predicted by the model to be spam that are actually spam, so again that's the same numerator as precision but now it's just divided by the total number that are actually spam instead of the total number that are predicted as spam. So that's accuracy precision and recall. So I like this visual representation of precision and recall,



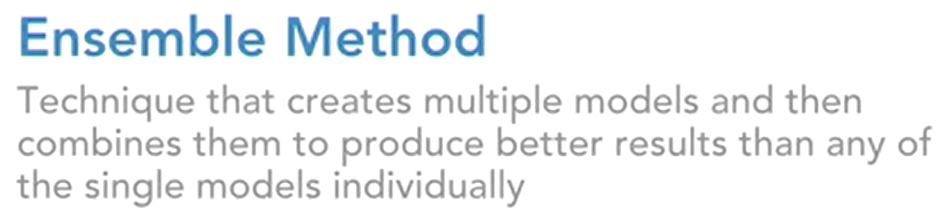


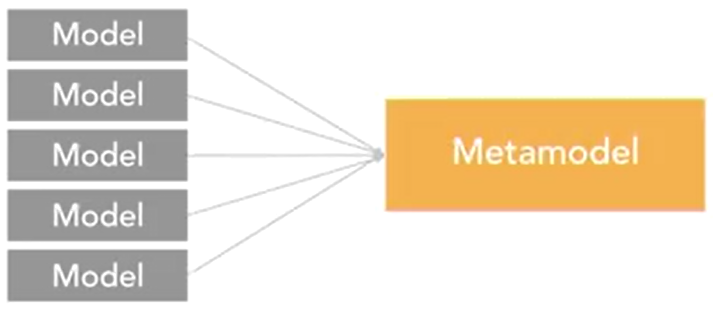
you can see that the numerator in both is the same, the amount that you correctly identified but the denominator is different; precision is all things that you said were relevant while recall is all the things that are actually relevant.

So in our example relevant means it's spam. So now precision recall give you the ability to kind of tailor the aggressiveness of your algorithm based on your business problem. For instance, if false positives are really costly then you'll want to optimize your model for precision. But if false negatives are really costly then you'll want to optimize the model for recall. Only knowing accuracy may not really give you incite into this kind of trade off.

And we'll dive deeper into those specifics as we start to build our own models. So now that we understand some evaluation metrics that we'll be using and we understand how to actually use those to evaluate the model I think we're set to actually build some models.

We're going to start getting into some actual machine learning now, starting with an introduction to random forest. Once I introduce random forest from a conceptual level, we'll jump into fitting and evaluating our own random forest model in the next lesson. I want to first note that random forest is one type of a machine learning algorithm that falls into a broader category of ensemble learners.



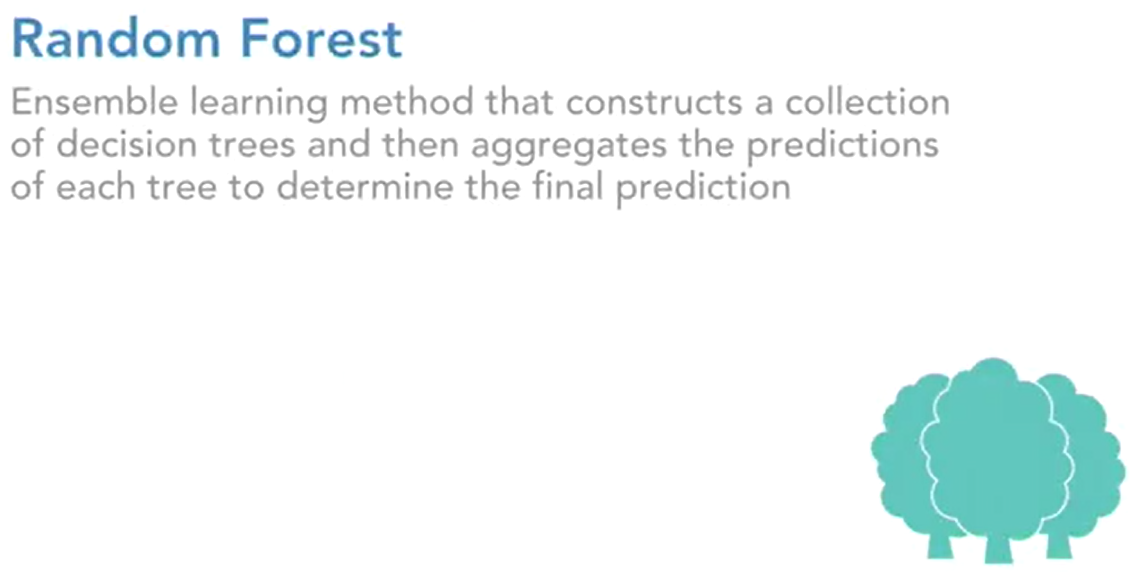


This takes advantage of the ensemble method, which is a technique that creates multiple models and then combines them to produce better results than any of the single models individually.

The idea behind ensemble learning is that you can combine a lot of weak models to create a single strong model. The basic idea is that this leverages the aggregate opinion of many over the isolated opinion of one.

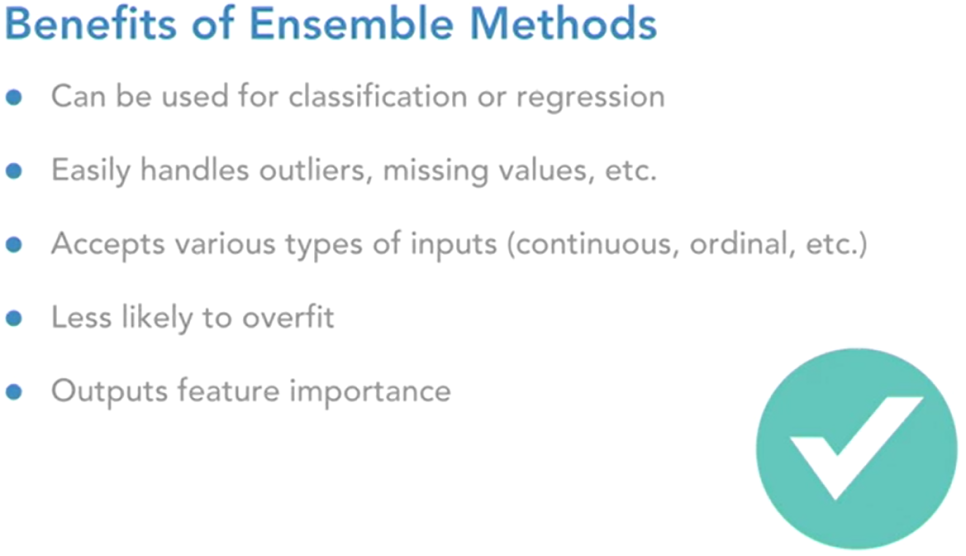


This method has a very strong theoretical motivation. With that said, we're not going to get into those details in this course. But it's just one of the many reasons that ensemble learning is a very popular method in machine learning. With that context, let's jump into defining random forest.



Random forest is an ensemble learning method that constructs a collection of decision trees and then aggregates the predictions of each tree to determine the final prediction. So in this case, your weak models are the individual decision trees, and then those are combined into the strong model that is the aggregated random forest model. So you may say, I want to build a random forest model to predict ham or spam. Let's just say that the random forest has one hundred decision trees in it. Then each of the hundred decision trees are built independently of one another, and each will output a prediction of either spam or ham.

So let's say 60 of those decision trees vote spam and 40 vote ham. Then the final prediction of the random forest model will be spam. So it's really just a simple voting method for the trees. There are a lot of benefits to using random forest. It's a very versatile and powerful machine learning algorithm. For one, it can be used for classification or regression, so that means a categorical response or a continuous response. It easily handles outliers, missing values, skewed data, the data doesn't even have to be on the same scale.

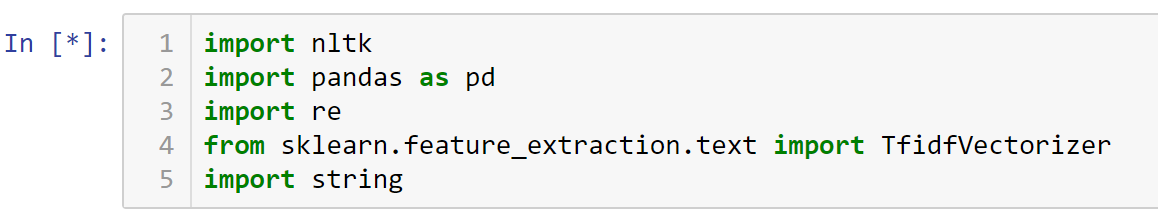


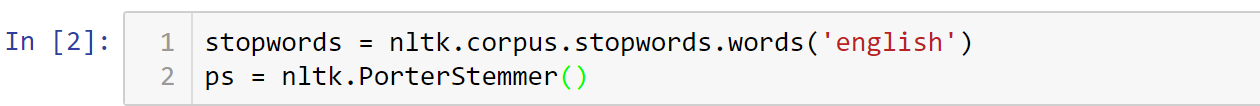
It accepts various types of inputs as well, may it be ordinal or continuous data. It's also less likely to overfit than some of the other machine learning models. We'll get more into that topic later on. And it generates a feature importance score for each feature. So random forest is really versatile, and it often makes a terrific first pass at your data, because you rarely have to do a lot of data cleaning, because it can accept pretty much anything. Beyond that, it's powerful, and it outputs feature importance to help you get a feel for which of your features are really useful, and which aren't.

So this is an extremely high-level introduction to random forest, giving us just enough information to understand what we're actually doing when we implement this in the code in the next section.

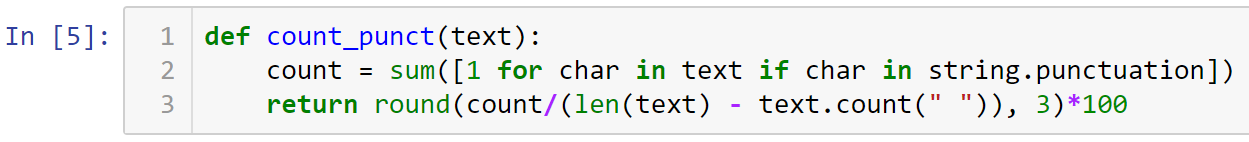
Implementing a random forest model in Python. In this exercise you will learn some of the basics about the random forest classifier in scikit-learn, and then we will learn how to fit and evaluate it using cross-validation. In this example, vectorization is done using the TfidfVectorizer.

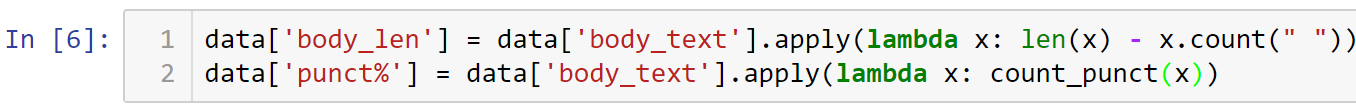
So as a refresher, that means a document term matrix where each cell is a weight of how important that word is, by measuring how frequently it occurs within that text message, relative to how frequently that word occurs across all other text messages. Also, we are creating a data frame called x\_features that does not include the label, and you'll learn why that label was kept separate in just a minute.

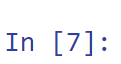


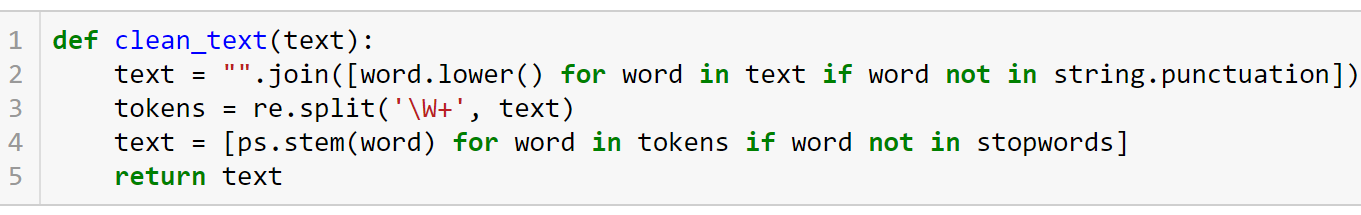


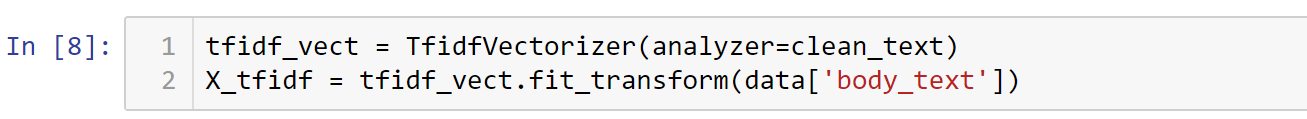




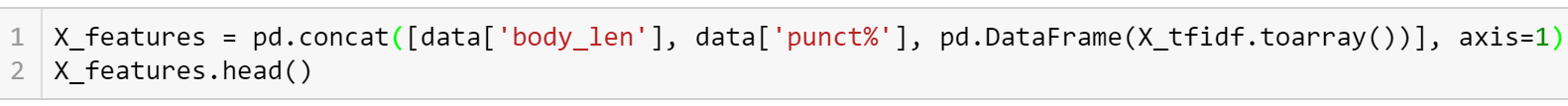




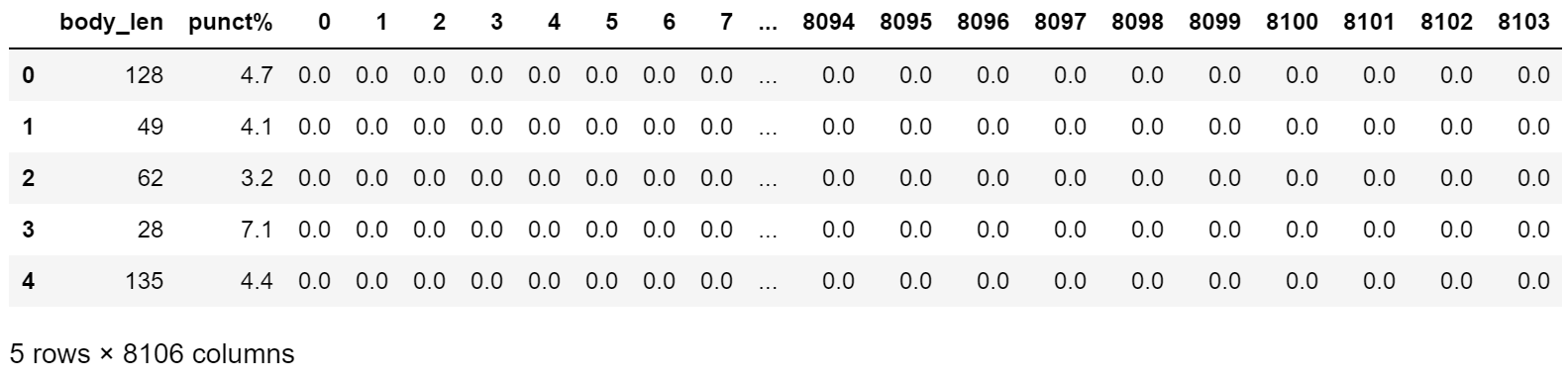




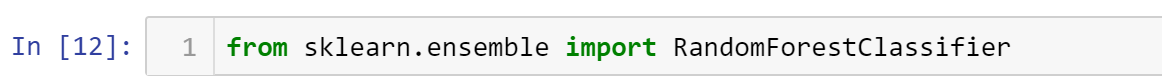


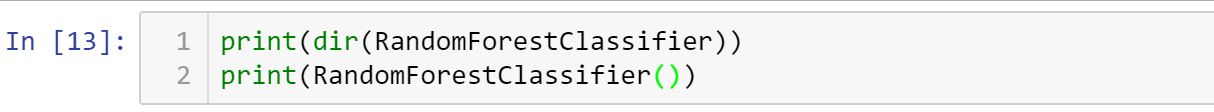


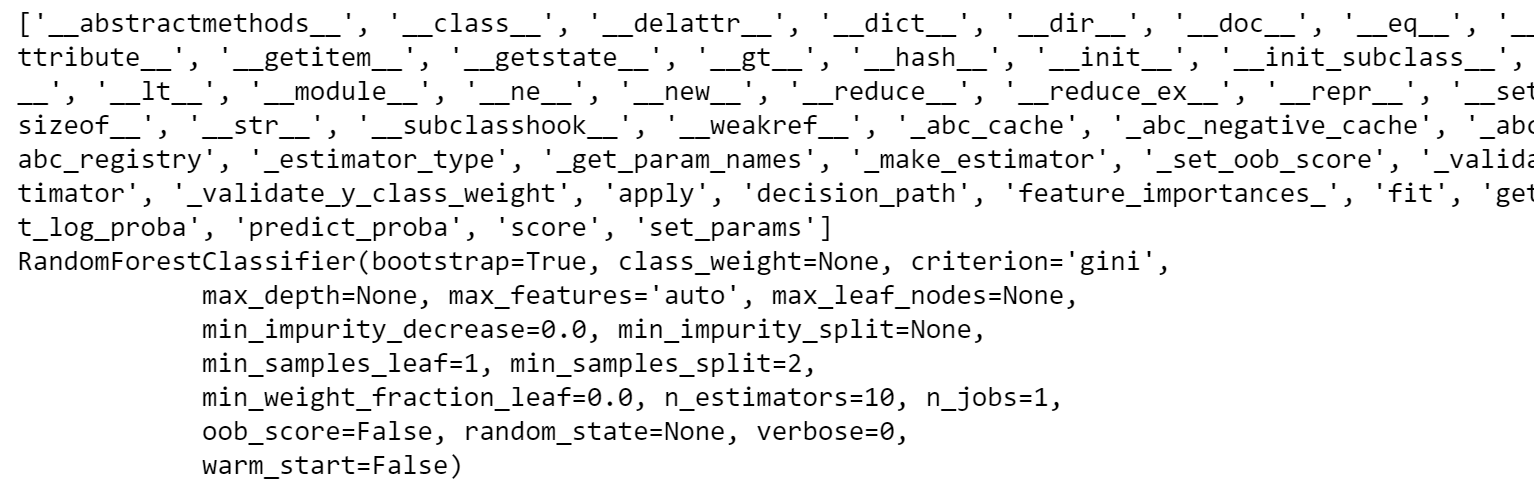




Explore RandomForestClassifier Attributes & Hyperparameters







Note the following members: ‘feature\_importance\_’, ‘fit’, ‘predict’, max\_depth, and ‘n\_extimators’.

The first is feature\_importances. This is what outputs the value of each feature to the model. Most algorithms do not provide this, and it's really helpful. Then fit is what allows you to fit your actual model and then you'll store that fit model as an object. Then, you can use this predict method from that fit model object, to make predictions on your test set.

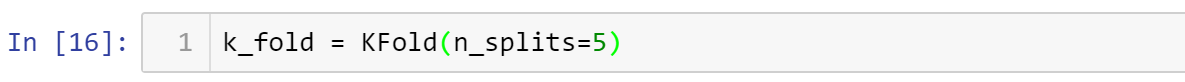
Max\_depth is going to be how deep each one of your decision trees is. You'll notice that the default is none. Basically, that means that it will build each decision tree until it minimizes some loss criteria. N\_estimators is how many decision trees that will be built within your random forest, so the default is 10. These defaults mean, your random forest would build 10 decision trees of unlimited depth. Then, there would be a vote among these 10 trees to determine the final prediction. Those are the only hyperparameters that we'll be using, but many of these other ones can be impactful. I encourage you to explore some of these other hyperparameters.

Explore RandomForestClassifier through Cross-Validation

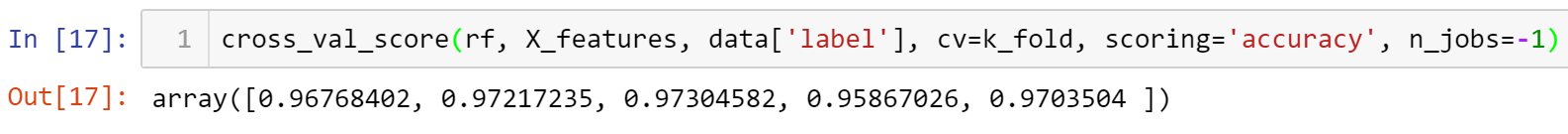




Setting n\_jobs to negative one basically just allows this to run faster by building the individual decision trees in parallel. Now, let’s split the model to five subsets …



Now, let’s run it … wait a few minutes ...



It will take just a minute. You'll notice in the first iteration, the model was trained and then evaluated on a test set and correctly predicted 96.9% of the samples.

And then the second iteration, again, it was trained on a different training set and evaluated on a different test set and it accurately predicted 97.5% of the samples. So that's pretty good. That's a quick taste of how to run through cross-validation. The most simple example is pretty easy. We really only had four lines of code including the import statement, and it returns a pretty good read on the range of outcomes for a model given this data and hyperparameter setting.

● All submissions should be separate from other exercises and quests. Please do not lump all your answers into one document and re-using that same workspace to gain multiple points. Thanks.

● Place your name at the bottom of your code, download your Python program in html format, and submit your work in Canvas.