Machine Learning: FAQ

Table of Contents:

- **What is ML?**
- **Why do we care?**
- **What do we mean by pipeline?**
- **How does ML learn from data?**
- **What are features?**
- **Why do we care about features?**
- **How do you extract features? (Tokenization)**
- **What are labels?**
- **What is training?**
- **What is testing?**
- **How do training features and test features relate to each other?**
- **Why don't we train on the test set?**
- **How do ML techniques allow us to make predictions?**
- **How do we measure a classifier's success?**
- **What can go wrong when classifying data?**

What is ML?

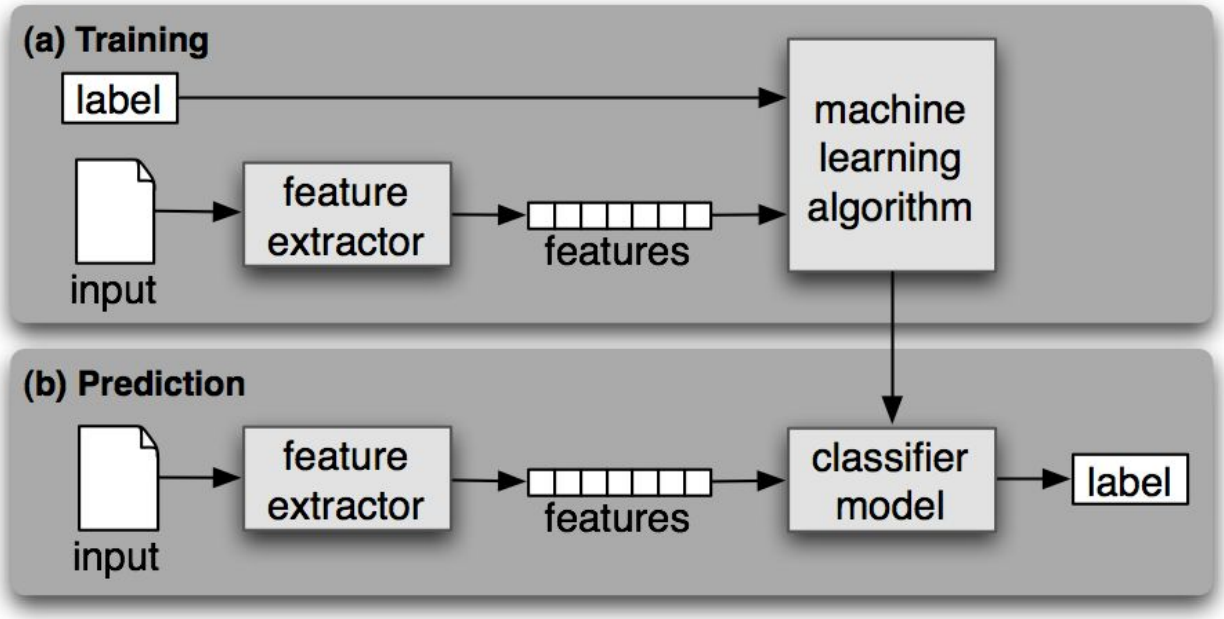
"Machine learning is a subfield of computer science[1] that evolved from the study of pattern recognition and computational learning theory in artificial intelligence.[1] In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed".[2] Machine learning explores the study and construction of algorithms that can learn from and make predictions on data.[3]" (Wikipedia on Machine Learning)

Basically, machine learning is the subset of computer science that allows a program to take in some data set and extract patterns from it. We do this by using the typical ML pipeline: training data => features => trained classifier. Using this trained classifier, we can make classifications on our test data. We convert the test data into features, and then use the classifier to make decisions on the test data.

Why do we care?

Machine learning is everywhere! Netflix uses Machine Learning to make movie recommendations for you, Facebook uses it to detect faces in your pictures, and you're going to use it to find out something cool in your final project. It is one of many different ways to extract trends and patterns from a large data set. Humans are very good at identifying correlations when data is cleverly visualized, but there are some statistical measures that allow a machine to detect things that a human would be unable to see.

What do we mean by pipeline?
In order to use machine learning, there are a number of steps which need to be taken. We refer to this as the machine learning pipeline.



This is a supervised learning pipeline (it isn't the only machine learning pipeline, though). For our training data, we have features and the correct labels for each piece of information. We use this to train our classifier. We can then create features for the test data and use the classifier to create predicted labels.

How does ML learn from data?
The training data is composed of two parts: features and labels. Using models (usually the classifier is named after the statistical model it uses, but some models don't use statistical models, like neural nets), the classifier learns what features (or combinations of features) are associated with which labels.

What are features?
Most simply, features are how we represent the data. There are many different ways to do this; for example, if we were comparing plays, and trying to predict when they were written, we could make the entire text a single feature. Much more useful, however, would be to use a bag of words model.

More generally, features are a way of summarizing information and abstracting away from the original data; this helps to remove noise in the data, which we don't necessarily want our classifier to learn, and it also makes classification problems tractable.

Why do we care about features?
Consider the case where we want to learn, from temperatures measured in Providence, whether a given day occurred in the Spring, Summer, Fall, or Winter. We have a data set which contains temperature measurements taken every second of a day, so we have 86400 pieces of information for any given day. However, we only have information for a single year, so we have 365 days of training data.

There are a few things that could go wrong if we tried to naively use all of this information when making a decision (which is to say, we shove it all into a classifier without thinking).

1. We could learn something different than what we wanted to learn. Let's say that, at 6:57:01 AM for every day in December, it happened to be the case that it was exactly 32 degrees. This could result in a classifier that says that any day which is 32 degrees at 6:57:01 AM is most likely in December, so that day must be in the winter.
2. We might not learn anything at all. There are so many features and we have so little training data that the classifier might miss the trends.

For this particular data set, we might be better off making decisions based on a different set of features - for example, the hourly average temperature. Then, we would have a feature set of 12 pieces of information per day, and this is a problem that we can solve.

How do you extract features?
Feature extraction is an art, not a science. We want to prevent the above mistakes from happening, but we don't want too little information, either. This can be a little bit of trial and error, as you saw in the lab. Sometimes, changing the features doesn't help at all, and sometimes it makes it much, much worse.

Let's say we want to do sentiment analysis, but we choose to ignore URLs. This can result in us missing important pieces of information; for example, we might ignore the fact that a tweet has a link to fmylife.com, and that's never going to be positive (unless the tweeter particularly enjoys schadenfreude).

One particular type of feature transformation is tokenization (feature transformation is typically referred to as tokenization when dealing with text data). Consider a tweet. The text of a tweet could be considered a single feature. However, this is not a particularly useful feature; two tweets will be completely different if their text is not exactly the same, and so will be treated differently by the classifier. A more useful feature would be a list of words which are in the tweet. Note that we are losing some information here; the order of the words has been lost. For this reason, classifiers often use bigrams. A bigram is two words in a row; for example, the sentence "It is raining." would produce the features ["It", "is", "raining", "It is", "is raining."] is a bigram model. You can generalize this up to n-grams.

In the above example, it appears as though our features could be transformed further. There seems to be no reason that “It” and “it” should be treated differently, or “raining.” and “raining”. Here is our full tokenization pipeline:

1. Split text by sentences.
2. Lowercase and remove punctuation.
3. Create bigrams by sentence, and grab individual words.

So, what are potential problems? Well, although removing periods seemed to make sense, removing exclamation marks might not; it is debatable whether “great” and “great!” should be mapped to the same feature. Similarly, although lowercasing “It” made a lot of sense, it is also ambiguous whether or not “stupid” and “STUPID” should be mapped to the same feature.

In general, when transforming features, we want to make the features as expressive as possible while making the feature space as small as possible. Note that these two goals are in conflict.

What are labels?

Labels are what we are ultimately interested in. If we are trying to predict which sports team is going to win a game, our labels might be “home team wins” and “home team loses”. If we are trying to predict the weather, our labels could be “sunny”, “rainy”, and “cloudy”. For most machine learning models, we require that the training data is labelled; this is called supervised learning. It is possible to do machine learning with unlabelled training data (this is called unsupervised learning) but it is much harder. Reinforcement Learning (RL) and Cluster analysis are examples of unsupervised learning.

What is training?

When you train the classifier, you are using an algorithm to make the models learn which combinations of features are correlated with which labels. The algorithm varies according to the classifier; for example, logistic regression uses gradient descent. As a result, different algorithms may make different predictions. Often, a comprehensive understanding of the inner workings of classifiers along with expert domain knowledge about how the objects you are classifying tend to behave is required to choose an appropriate classification method. This is less true for your final projects (though you are free to delve as deeply into the literature as you like), and more true if you were trying to decide if a particular set of symptoms and test results (features) imply that a patient has cancer.

What is testing?

Typically, you will get higher training accuracy scores than testing accuracy scores. This is due to the fact that the classifier inevitably learns some trends which are specific to just the training data and not to general data for that problem; this is known as overfitting. Looking at the testing score can show the discrepancy.

Alternatively, using cross-validation can show the same thing. In fact, depending on your data set, cross-validation is what you should do to test your data.

Cross-validation is a method to prevent overfitting. It splits your training data into equal chunks - if you do 10-fold cross validation, it will run 10 times, using a different set of 9 chunks as the training and a different last chunk as test each time.

Note that cross-validation is not a silver bullet - there may be some cases where the test and training sets are fundamentally different is some way, so high cross-validation scores may not be predictive of how well your classifier will perform. For example, if you are performing sports-related analysis and your training data is taken from the 2000-2013 baseball seasons, it makes more sense to use the 2014 season as the test data than to do cross-validation.

How do training features and test features relate to each other?
It's important to remember that the classifier will not see the original data set, it has only seen the features it was trained on. This means that your training set and test set need to be "featurized" in the same way, as in the bag of words model in the ML Blackbox lab.

If they aren't, very bad things can happen. Let's say that, when creating your set of training features, your bag of words model thinks that the 10th is "Boo!" However, you created a new bag of words model when you featurized the test set, so now the 10th word in your model is "Awesome!". Clearly, the sentiment scores you learned in training won't apply in test, so this is a problem; this is why your test features need to be in the context of your training features.

Why don't we train on the test set?
Your classifier learns everything it knows about the world from the training data. When it is training, it tries to make decisions that makes its accuracy as good as possible on the training set. This means that the classifier will perform extremely well on the training data; it also means that the classifier has seen the training data before, so it isn't a real test of its capabilities. It's basically the same as showing a student the answers to the exam, and then trying to infer their mastery of the course from how well they do.

One of the big mistakes a classifier can make is to overfit on the training set. We can try to negate this effect using cross-validation, but the performance on the test set is a very clear way to see how much the classifier overfit. Let's say we have a classifier that memorizes the input and always maps that exact input to the correct label. This sort of classifier will have 100% accuracy on the training set, but won't have learned anything else. If the test set is included in training, this classifier will have memorized the correct answers, and will have 100% accuracy on the test set as well. When using this classifier into the wild, it may not perform any better than chance.

How do ML techniques allow us to make predictions?
Congratulations! If you have a classifier, you also have a predictor! You're already on your way to using your laptop as a crystal ball to read fortunes and game the stock market. Wall Street even claims you can get the rest of the way there using statistical models from physics and assuming that elementary particles and stocks behave in the same way (We recommend not betting your life savings on this assumption).

Given a supervised classifier which you've trained and tested, you can release it into the wild. Just feed it new data, and it will keep generating labels.

How do we measure a classifier's success?
This is kind of the bulk of machine learning. There are various techniques for measuring this, but ultimately, it depends on the problem you are answering.

In computer vision, for example, the success of a classifier can be measured by how the classifier's performance compares to a human's performance.

As a rule of thumb, if your classifier is performing less well than chance for your data set (note that "chance" isn't necessarily 50%), there is an error somewhere. This is because, rather than not having learned something, it has learned a feature that is always wrong, and this shouldn't really happen.

What can go wrong when classifying data?
SkyNet. ...Though really, there are more than a few AI researchers who would be ecstatic if we could do this...

More seriously, your classifier might be wrong.

1. It can overfit.
2. It can underfit.
3. It can just be wrong.
4. Our training data could have errors.

Depending on the context in which you are using your classifier, this can have varying levels of impact. For our purposes, it's probably not a huge deal if your classifier is wrong. In other contexts, it can be extremely important to know in what ways your classifier makes mistakes.

In the ML Blackbox Lab, we talked a little bit about precision and recall. Precision is the number of things the classifier classifies as positive which were actually positive over the number of positive guesses it made. Recall is the number of positive classifications it made out of all positive classifications. For any given classifier, we can change the confidence level/threshold at which it categorizes an input. When we do that, we change where the classifier is falling on the precision-recall curve.

This isn't the easiest concept in the world, so let's look at a concrete example: let's say we're a search engine, and we want all of the search results on the first page to be relevant (because, really, who makes it onto the second page of a Google search?). In this case, we want to have very high precision, but we may not care whether or not we positively classify every single result. This would give us lower recall than we could have otherwise.

We may not always want high precision, however. Let's say we're a hospital looking at MRIs, and we want to correctly classify all tumors. We strongly prefer to positively identify all tumors, even if that means sifting through extra false positives. We want to have very high recall, and we care much less about precision.

As you can see, keeping the context of your problem in mind when you make decisions about the best ways to design your classifier is important.