Applied Machine Learning: Background

Ryan Kingery

May 22, 2018

Informal Definition of Machine Learning

Machine learning is the construction of algorithms that can learn from data and make predictions about data without the need of human input. This is done by training statistical models to approximate reality.

Vocabulary

Machine learning has a lot of domain specific terms that you've got to know if you want to know what's going on. Here are a few of them.

1. Features: The independent variables, i.e. x's, in the dataset. These are usually vectors, often of very high dimension. The space in which features "live" is called the *feature space*.

2. Labels/Targets: The dependent variables, i.e. y's, in the dataset, if given. These can be vectors or scalars.

3. Supervised Learning: Techniques used when both features and labels are given in the dataset. This is the most common situation in practical applications of machine learning, mainly because it's the easiest type of learning. The goal is to predict y, given x.

4. Regression: Supervised learning where the labels are assumed to be continuous (e.g. linear regression).

5. Classification: Supervised learning where the labels are assumed to be discrete (e.g. logistic regression). A special case of this is *binary classification*, where the labels only take on the binary values 0 and 1. I'd say the vast majority of machine learning situations deal with classification (think image classification or anomaly detection).

6. Decision Boundary: The boundary separating the feature space into its different classification regions. For example, in binary classification, the decision boundary separates the feature space into two regions, one where the estimated labels are $y = 0$ and the other where the labels are $y = 1$.

7. Unsupervised Learning: Techniques used when only the inputs are given in the dataset. The labels are said to be hidden or *latent*, i.e. they're there but you just don't know them. The goal is to find underlying patterns in the feature space. Note: Unlike other fields, in security it seems to be the case that unsupervised learning is more commonly used than supervised learning (think anomaly detection).

8. Clustering: Unsupervised learning techniques in which the inputs are clustered into groups via various metrics.

9. Feature Reduction: Unsupervised learning techniques that involve collapsing the dimension of the feature space down to a smaller size while still capturing most of the behavior in the dataset. Often used for visualizing high-dimensional data.

10. Generative Modeling / Density Estimation: An unsupervised learning technique whose goal is to estimate the entire joint probability distribution of the features and labels, i.e. $p(x, y) = p(y|x)p(x)$. Generative modeling is often used to create *synthetic* data (i.e. fake data that "looks like" real data), which is useful in security for things like anonymizing data for privacy reasons. Generative modeling is also used for other cool things like creating fake images that look like real images, creating fake art, and creating chat-bots. This is an area that's started to really take off only recently thanks to the advent of GANs.

11. Anomaly Detection: Could be supervised or unsupervised. The goal is to find points in a dataset that look "odd" in some statistically meaningful way. This is the bread and butter of most security applications of machine learning.

12. Generalization: The ability of a model to predict the labels of unseen inputs very well, i.e. the ability for $g(x) \approx f(x)$ very well even outside of the dataset. A good model is said to *generalize* well to unseen data.

13. Reinforcement Learning: Techniques in which the dataset has some built in notion of reward. That is, you're given inputs, plus reward values that tell you "how well you're doing". The idea is to maximize your total reward. This is the stuff that companies like OpenAI are putting their effort into, and it's commonly believed to be the best path to artificial general intelligence (AGI). In practice, though, reinforcement

learning has struggled to pick up steam because its methods tend to be very slow and cumbersome to date, so we won't really focus on it.

14. Overfitting: A situation in which the model was "over-trained" on the dataset. When this occurs, you'll find that $g(x) \approx f(x)$ very well on the points in your dataset, but very poorly on points not in your dataset. There are lots of techniques to deal with this, but it's always something to watch out for. The easiest way to help prevent this is to separate your dataset into a training set and a test set, train only on the training set, and evaluate your model only on your test set.

15. Training Set: The subset of your data that you actually train your learning algorithm on. This is the only data it is allowed to "see" until you're done with training and have a model.

16. Test Set: The subset of your data that you evaluate your model on after you're finished training. This is the set that tells you how well your model is actually doing.

17. Validation Set: The subset of your data commonly used to tweak your model (called hyperparameter tuning) prior to formal training. Note often people the test set for both testing and validation, though this can cause problems with accidental overfitting if you're not careful. Better to split them up if you have enough data.

18. Cross-Validation: The name of the methods used to separate your dataset into a training set and a test set. The most common way to do this is to take around 70% of your data as your training set and the other 30% as your test set (though this isn't a hard and fast rule, and there are exceptions). If there is no time dependence in your data, you should do this by random sampling. If there is, you probably want your test set to be your most recent data time-wise.

19. Deep Learning: Machine learning with large, very complicated models. Usually those models are some type of neural network, and the datasets are very large (on the order of gigabytes or more usually). This is the subfield of machine learning that's getting all the hype right now.

Theory

Formal Definition: Suppose we're given a set of data $D = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, where each $x_i \in \mathbb{R}^m$ are the input features and each $y_i \in \mathbb{R}^K$ are the output labels. We assume there is some unknown function $y = f(x)$ generating the data. Machine learning is the process of using a *learning algorithm* to find a function $g(x)$ from a class $\mathcal{M}$ of model functions such that $g(x) \approx f(x)$.

Note: I'm not being completely honest with you here. We should actually be working with probability distributions instead of functions, but you'll almost always be fine in practice by thinking in terms of functions, so I won't bother getting too theoretical on you. If you want the formal details just ask me some time. Also, note that specifically I'm talking about supervised learning, a subset of machine learning that is by far the most commonly used in practice. The other common subset of machine learning, unsupervised learning, is somewhat similar, and will be described later on.

Learning Algorithms: How do we find a "good" function $g(x)$ such that $g(x) \approx y$? The idea is to construct another function that tells you how well your choice of $g(x)$ approximates y, and optimize over it to find the best choice of $g(x)$ possible given your model. This other function is called a *loss function*, denoted $L(g(x), y)$. It tells us how "bad" we're doing at each step. High loss means we're doing poorly. Low loss means we're doing well. It seems natural, then, that the goal is to find a function $g(x)$ that minimizes the loss.

Parametric Models: Minimizing a function with respect to another function is very hard (calculus of variations). Thus, in practice what we often do is take $g(x)$ to be a *parametric model* $g(x|w)$, where w are *weights* or *parameters* that need to be estimated. Instead of minimizing the loss over all possible functions $g(x)$, we minimize the loss over all possible weights $w \in \mathbb{R}^p$. This is a finite dimensional problem, hence way easier work with.

Loss functions: We're almost done with the theory. We just need to clear up a slight subtlety first. Notice that the loss function outputs a different value for each (x_i, y_i) pair. Our goal, though, is to minimize the loss over all of them. We do this by taking the *average loss* over each pair of data points. That is, we want to minimize

$$\hat{L}(g(x|w), y) \equiv \frac{1}{n} \sum_{i=1}^{n} L(g(x_i|w), y_i)$$

over all $w \in \mathbb{R}^p$. To simplify what we're doing, think of the loss as *only* a function of w. That is, let $J(w) = \hat{L}(g(x|w), y)$, where x, y are treated as constant. The goal of machine learning, then, is to find the optimal weights

$$w^* \equiv \underset{w \in \mathbb{R}^p}{\text{argmin}} J(w).$$

At this point we're done. Our best model of $f(x)$ is given by $g(x|w^*)$. That is, suppose later on we observe a point x_j and would like to predict what its label y_j would be, then we would predict $\hat{y}_j = g(x_j|w^*)$. Note from here on we will often define $\hat{y} \equiv g(x|w)$ for convenience, and because most of the literature does it too.

Common Loss Functions: These are the most common loss functions chosen in practice.

1. Squared Loss: $L(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$
2. Absolute Loss: $L(\hat{y}, y) = |y - \hat{y}|$
3. Binary Cross Entropy Loss: $L(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$
4. Categorical Cross Entropy Loss: $L(\hat{y}, y) = \sum_{k=1}^{K} -y_k \log(\hat{y}_k) - (1-y_k)\log(1-\hat{y}_k)$
5. Hinge Loss: $L(\hat{y}, y) = \max(0, 1 - yw^T x)$
6. 0/1 Loss: $L(\hat{y}, y) = \begin{cases} 1, & y = \hat{y} \\ 0, & y \neq \hat{y} \end{cases}$

Summary of Supervised Learning Process: Suppose we are given some data D. We want to build a model to predict new data well. We do so by following these steps.

1. Choose a (usually parametric) model $g(x|w)$.
2. Choose a loss function function $L(g(x|w), y)$.
3. Minimize the average loss $J(w) = \hat{L}(g(x|w), y)$ using a suitable optimization algorithm.
4. Return a "best" model $g(x|w^*)$ and use it to predict new data.

Examples

Example (Linear Regression): Suppose we're given a dataset D with $x_i, y_i \in \mathbb{R}$. We want to build a simple model to predict y_j given an unseen point x_j. Possible approach:

1. Choose a model
$$\hat{y} = g(x|w_0, w_1) = w_0 + w_1 x,$$
where $w_0, w_1 \in \mathbb{R}$ are the weights to be estimated.
2. Take the loss function to be the squared loss
$$L(\hat{y}, y) = \frac{1}{2}(y - w_0 - w_1 x)^2.$$

3. Since this problem is simple, we can minimize with basic vector calculus. Set

$$J(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(y_i - w_0 - w_1 x_i)^2.$$

   Minimize $J(w_0, w_1)$ by setting its gradient equal to zero and solving for w_0, w_1. These will be w_0^*, w_1^*.

4. The best fit model is then $\hat{y} = w_0^* + w_1^* x$.

Example (Logistic Regression): Suppose we're given a dataset D with $x_i \in \mathbb{R}$ and $y_i = 0, 1$. We want to build a simple model to predict $y_j = 0, 1$ given an unseen point $x_j \in \mathbb{R}$. Possible approach:

1. Choose a model

$$\hat{y} = g(x|w_0, w_1) = \frac{1}{1 + e^{-(w_0 + w_1 x)}},$$

   where $w_0, w_1 \in \mathbb{R}$ are the weights to be estimated.

2. Take the loss function to be the cross entropy

$$L(\hat{y}, y) = -y \log\left(\frac{1}{1 + e^{-(w_0 + w_1 x)}}\right) - (1 - y)\log\left(1 - \frac{1}{1 + e^{-(w_0 + w_1 x)}}\right).$$

3. This optimization problem, like almost all optimization problems in machine learning, can't be solved using basic methods, so we have to rely on an optimization algorithm (e.g. gradient descent) to find w_0^*, w_1^*. Set

$$J(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n} -y_i \log\left(\frac{1}{1 + e^{-(w_0 + w_1 x_i)}}\right) - (1 - y_i)\log\left(1 - \frac{1}{1 + e^{-(w_0 + w_1 x_i)}}\right).$$

4. The best model is then

$$\hat{y} = \frac{1}{1 + e^{-(w_0^* + w_1^* x)}}.$$

   Note that this outputs *continuous* values between 0 and 1! Thus, to actually predict y given x, we simply round $\hat{y}$ to the nearest integer.

Example (k Nearest Neighbors): Suppose we're given a dataset D with $x_i \in \mathbb{R}$ and $y_i = 0, 1$. The goal is again to find a model to predict $y_j = 0, 1$ given an unseen point $x_j \in \mathbb{R}$. The k nearest neighbors algorithm is a simple, non-parametric (i.e. no weights) method to do this. Suppose you want to classify a new point x_j to be $y_j = 0$ or $y_j = 1$. A very simple way to do this is the following: Find the k closest points to x_j in the dataset and see how they're classified. If most of them are 0, classify $y_j = 0$. If most of them are 1, classify $y_j = 1$. The algorithm described is so simple that it's usually not even formulated as a loss minimization problem. However, you *can* formulate it the way I described above by using the 0/1 loss.

Common Learning Algorithms

Supervised Learning

1. Regression
   (a) Linear Regression
   (b) Polynomial Regression
   (c) Nonlinear Regression
   (d) Stepwise Regression
   (e) Neural Networks
   (f) Regression Trees and Random Forests
2. Classification
   (a) Logistic Regression
   (b) k Nearest Neighbors (KNN)
   (c) Neural Networks
   (d) Naive Bayes
   (e) Support Vector Machines (SVMs)
   (f) Classification Trees and Random Forests

Unsupervised Learning

1. Clustering
   (a) k Means Clustering

   (b) Gaussian Mixture Models (GMMs)
   (c) Hierarchical Clustering
   (d) Hidden Markov Models (HMMs)
   (e) DBSCAN
   (f) k Nearest Neighbors (KNN)

2. Feature Reduction
   (a) Principle Component Analysis (PCA)
   (b) Singular Value Decomposition (SVD)
   (c) Multidimensional Scaling (MDS)
   (d) Linear Discriminant Analysis (LDA)

3. Generative Modeling / Density Estimation
   (a) Kernel Density Estimation (KDE)
   (b) Hidden Markov Models (HMMs)
   (c) Gaussian Mixture Models (GMMs)
   (d) Naive Bayes
   (e) Latent Dirichlet Allocation (LDA)
   (f) Generative Adversarial Networks (GANs)