CHAPTER 14

Key Ideas in Machine Learning

This semester we have covered many concepts, algorithms, and theoretical results in machine learning. Here we review and discuss some of the key ideas.

1 Introduction

Machine learning is a discipline focused on two inter-related questions: "How can one construct computer systems that automatically improve through experience?" and "What are the fundamental theoretical laws that govern every learning system, regardless of whether it is implemented in computers, humans or organizations?" The study of machine learning is important both for addressing these fundamental scientific and engineering questions, and for the highly practical computer software it has produced and fielded across many applications.

Machine learning covers a diverse set of learning tasks, from learning to classify emails as spam, to learning to recognize faces in images, to learning to control robots to achieve targeted goals. Each machine learning problem can be precisely defined as the problem of improving some measure of performance P when executing some task T, through some type of training experience E. For example, in learning an email spam filter the task T is to learn a function that maps from any given input email to an output label of spam or not-spam. The performance metric P to be improved might be defined as the accuracy of this spam classifier, and the training experience E might consist of a collection of emails, each labeled as spam or not. Alternatively, one might define a different performance metric P that assigns a higher penalty when non-spam is labeled spam, than when spam is labeled non-spam. One might also define a different type of training experience, for example by including unlabeled emails along with those labeled as spam and not-spam. Once the three components $\langle T, P, E \rangle$ have been specified fully, the learning problem is well defined.

2 Key Concepts

This semester we examined many specific machine learning problems, applications, algorithms, and theoretical results. Below are some of the key overarching concepts that emerge from this examination.

2.1 Key Perspectives on Machine Learning

It is useful to consider machine learning problems from several perspectives:

- *Machine learning as optimization.* Machine learning tasks are often formulated as optimization problems. For example, in training a neural network containing millions of parameters, we typically frame the learning task as one of discovering the parameter values that optimize a particular objective function such as minimizing the sum of squared errors in the network outputs compared to the desired outputs given by training examples. Similarly, when training a Support Vector Machine classifier, we frame the problem as a constrained optimization problem to minimize an objective function called the hinge loss. When machine learning tasks are framed as optimization problems, the learning algorithm is often itself an optimization algorithm. Sometimes we use general purpose optimization methods such as gradient descent (e.g., to train neural networks) or quadratic programming (e.g., to train Support Vector Machines). In other cases, we can derive and use more efficient methods for the specific learning task at hand (e.g., methods to calculate the maximum likelihood estimates of parameters for a Naive Bayes classifier).

- *Machine learning as probabilistic inference.* A second perspective is that machine learning tasks are often tasks involving probabilistic inference of the learned model from the training data and prior probabilities. In fact, the two primary principles for deriving learning algorithms are the probabilistic principles of Maximum Likelihood Estimation (in which the learner seeks the hypothesis that makes the observed training data most probable), and Maximum a Posteriori Probability (MAP) estimation (in which the learner seeks the most probable hypothesis, given the training data plus a prior probability distribution over possible hypotheses). In some cases, the learned hypothesis (i.e., model) may itself contain explicit probabilities (e.g., the learned parameters in a Naive Bayes classifier correspond to estimates of specific probabilities). In other cases, even though the model parameters do not correspond to specific probabilities (e.g., a trained neural network), we may still find it useful to view the training algorithm as performing probabilistic inference to find the Maximum Likelihood or the Maximum a Posteriori probability network parameters' values. Note this perspective that machine learning algorithms are performing probabilistic inference is very compatible with the above perspective that machine learning algorithms are solving an optimization problem. In most cases, deriving a learning algorithm based on the MLE or MAP principle involves first defining an objective function in terms of the parameters of the hypotheses and the training data, then applying an optimization algorithm to solve for the hypothesis parameter values that maximize or minimize this objective.

- *Machine learning as parametric programming.* Another perspective we can take on the same learning programs is that they are choosing parameter values that define a function or a computer program written in a programming language which is defined by their hypothesis space. For example, we can view deep neural networks as implementing parameterized programs, where the learned network parameters instantiate a specific program out of a set of potential programs predefined by the given network structure. As we move from simple feedforward networks, to networks with recurrent (feedback) structure, and with trainable memory units, the set of representable (and potentially learnable) programs grows in complexity.

- *Machine learning as evolutionary search.* Note that some forms of learning do not admit an easy formulation as an optimization or probabilistic inference problem. For example, we might view natural evolution as a learning process – from generation to generation it produces increasingly successful organisms. However, in natural evolution it is not clear that there exists an explicit objective being optimized over time, or a corresonding probabilistic inference problem. Instead, the notion of "increasingly successful

organism" may itself change over time, as the environment of the organism and its set of competitors evolve as well.

2.2 Key Results

Although the field of machine learning is very much still under development, there are a number of key results that help us understand how to build practical machine learning systems:

- *There is no free lunch.* When we consider it carefully, it is clear that no system - computer program or human - has any basis to reliably classify new examples that go beyond those it has already seen during training, unless that system has some additional prior knowledge or assumptions that go beyond the training examples. In short, there is no free lunch – no way to generalize beyond the specific training examples, unless the learner commits to some additional assumptions. To see this, consider the set of hypotheses H explored by a decision tree learning system. Here H is the set of all possible decision trees that might be output by the learning program as it tries to learn some boolean classifier function $f : X \rightarrow \{0,1\}$ from labeled training examples. For simplicity, assume that each instance x in X is itself a tuple of n boolean valued features; that is, $x = \langle x_1, \ldots x_n \rangle$, where each x_i has the value 0 or 1. Now consider the question of how many training examples the decision tree learner must observe before it can identify the correct decision tree $h(x)$ among the set of all possible decision trees H. To answer this question, first notice that no matter what deterministic target function $f^* : X \rightarrow \{0,1\}$ the teacher wishes to teach, the learner will be able to find a decision tree of depth n that perfectly represents that function.[1] Put another way, the set H of all decision trees is sufficiently expressive to represent *any* function that can be defined over the instances X. Unfortunately, a consequence of this expressive power is that the learner will be uncertain which decision tree to choose, until it has seen every instance x from X as a labeled training example. This can be seen easily if we consider the case where the teacher has already provided labeled training examples corresponding to every instance x from X except for one final instance x_f which it has not yet labeled. At this point, the learner will find there are still two decision tree hypotheses that it cannot choose between. Both of these hypotheses will correctly label all of the labeled training examples seen thus far, but they will assign different labels to x_f. Only after the trainer provides labels for every single example in X will the learner be able to resolve which of the

[1] This is true because a decision tree of depth n will in this case sort each instance x from X into a unique leaf node, where either label for Y can then be assigned.

possible decision trees in H is the one corresponding to the target function being taught by the trainer. Although we use decision trees as an example here, the argument holds for any learning algorithm.

- *Three sources of error in learned functions. Bias, variance* and *unavoidable* error are three qualitatively distinct sources of error when attempting to learn some target function $f : X \rightarrow Y$, or equivalently $P(Y|X)$. First, *bias* refers to errors caused when the learner fails to consider equally each possible function that can be defined over the instances X. This can occur when the learner's hypothesis space H is insufficient to represent every function that can be labeled over X, or alternatively even if H is sufficiently expressive but the learner has some preference (bias) for choosing between two hypotheses that perform equally over the training data (e.g., a preference for short decision trees). Of course the bias might be correct or incorrect, but it is one possible source of error. Second, *variance* in the set of observed training data can be a source of error. If we consider obtaining training data by drawing a set of m examples from an underlying distribution $P(X)$, then statistical variations in this set of randomly drawn examples can lead to unrepresentative sets of training examples, which can contribute to error. Of course if we increase the number m of training examples, then we can reduce the expected impact of this kind of variance in the draw of training data. Finally, a third possible source of error is the *unavoidable error* that occurs when we attempt to learn a non-deterministic function. For example, if for a particular instance x, $y = 1$ with probability 0.6 and $y = 0$ with probability 0.4, then even if the classifier predicts the more probable $y = 1$ label for x, it will make an unavoidable error in 40% of these cases.

- *Overfitting.* We say that a particular hypothesis, h, overfits the training data if its error rate over the training data $error_{train}(h)$ provides an underestimate of its true error $error_{true}(h)$. Furthermore, we define the *degree of overfitting* to be $(error_{true}(h) - error_{train}(h))$. Overfitting is a key practical issue, because it typically is a sign that the learned hypothesis h will perform poorly when we try to use it in the future. Overfitting is most problematic when the number of training examples is small, or the complexity of the hypothesis space considered by the learner is large – both lead to a situation in which multiple hypotheses will perform equally well over the training data, and the learner will be unable to determine which hypothesis will perform best over future test data. The two most common approaches to handling overfitting are cross validation and regularization. Cross validation can be used to select the complexity of the final output hypothesis (e.g., to choose the size of the learned decision tree) based on its performance on data held out from training. Regularization is typically performed

by adding a penalty to the learning objective that penalizes the magnitude of learned parameter values (e.g., in L1 and L2 regularization), providing a bias in which the learner prefers simpler hypotheses. This increase in bias typically reduces the sensitivity of the learning algorithm to variance in the observed training examples. In many cases, regularization is equivalent to placing a prior probability distribution on the values of the parameters to be learned, then deriving their MAP estimates (e.g., L2 regularization corresponds to a zero mean Gaussian prior, whereas L1 corresponds to a zero mean Laplace prior).

- *Bayesian Networks and Graphical Models.* One important family of machine learning algorithms is based on learning an explicit representation of the joint probability distribution over a set of variables. For example, Bayesian Networks are directed acyclic graphs in which each node represents a random variable, edges represent probabilistic dependencies, and the collection of conditional probability distributions associated with each node/variable define the joint distribution over the entire set of variables. The structure of a Bayesian Network can be viewed as representing assumptions about conditional independencies among the different variables, and it entails a factorization of the joint probability of the n variables into a set of n terms. By comparing this factorization of the joint probability to the factorization obtained by the chain rule of probability, one can see explicitly how the network graph structure restricts the form of the joint distribution. More general graphical models, including undirected graphical models are also common.

- *Generative versus Discriminative Graphical Models.* When designing probabilistic learning algorithms, it can be helpful to distinguish generative versus discriminative models. Nave Bayes and Logistic regression are an example of a generative-discriminative pair of learning methods. Whereas Naive Bayes represents $P(Y|X)$ and $P(Y)$ explicitly, Logistic Regression instead learns the representation of $P(Y|X)$. These are called a generative-discriminative pair of algorithms because Logistic Regression uses a functional form for $P(Y|X)$ which is entails by the Naive Bayes assumptions. Similarly, Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) are another example of a generative-discriminative pair of algorithms for sequential data. The discriminative version typically has the advantage that during training it need not obey the constraining assumptions that its generative counterpart must.

- *Deep Neural networks.* Neural networks, or deep networks,. are a family of learning algorithms in which networks of simple, parameterized functional

units are interconnected to perform a larger computation, and where learning involves simultaneously training the parameters of all units in the network. Networks containing millions of learned parameters can be trained using gradient descent methods, often with the help of specialized GPU hardware. One important development in recent years is the growing use of a variety of types of units such as non-linear rectilinear units, and units that contain memory such as Long-Short Term Memory (LSTM) units. A second important development is the invention of specific architectures for specific families of problems, such as sequence-to-sequence architectures used for machine translation and other types of sequential data, and convolutional network architectures for problems such as image classification and speech recognition where the architecture provides outputs that are invariant of translations to network inputs (e.g. to recognize the same object in different positions in the input image, or the same speech sound at different positions in time). An important capability of deep networks is their ability to learn re-representations of the input data at different hidden layers in the network. The ability to learn such representations has led, for example, to networks capable of assigning text captions to input images, based on a learned internal representation of the image content.

- *PAC learning theory.* Results from Probably Approximately Correct (PAC) learning theory provide quantitative bounds on the degree of overfitting that will occur in specific learning settings. For the common learning problem of supervised learning of functions $f : X \rightarrow Y$, PAC theory bounds the probability δ that the degree of overfitting $[error_{true}(h) - error_{train}(h)]$ will exceed ε if the learning algorithm receives m labeled training examples drawn at random from a fixed probability distribution $P(X)$. These bounds depend not only on the number of training examples m, but also on the complexity of the set of hypotheses H considered by the learning algorithm. This body of theoretical research has uncovered important measures of the complexity of H, including the Vapnik-Chervonenkis (VC) dimension of H, and the Rademacher complexity of H. Both of these measures capture the expressive power of H to represent diverse functions that can be defined over X.

- *Learning ensembles of functions.* In some cases we can improve the accuracy of learned functions (e.g., classifiers) by learning multiple approximations to the desired target function, then taking a weighted vote of their predictions. For example, the Weighted Majority Algorithm learns the voting weights for a given set of alternative approximations to the target function, in an online setting where a sequence of examples is presented, predictions are made after each example, and the correct label is then revealed. Af-

ter each example appears, a weighted vote is taken to produce an ensemble prediction, and weights of individual ensemble members are adjusted once the correct label is revealed. Interestingly, it can be proven that the number of mistakes made by this weighted vote, over the entire sequence of examples, is only a small multiple of the number of mistakes made by the best predictor in the ensemble, plus a term which grows only as the log of the number of members of the ensemble. A second algorithm, called AdaBoost, goes further, by learning both the voting weights and the hypotheses themselves. It operates by training a sequence of distinct hypotheses from a single set of training example, by reweighting the training examples to focus at each iteration on the examples that were previously misclassified. PAC-style theoretical results bound the degree of overfitting for AdaBoost based on the VC dimension (complexity) of the hypothesis space used by the base learner. Boosting algorithms that learn ensembles of short decision trees (decision forests) are among the most popular classification learning methods in practice.

- *Semi-supervised learning and partially observed training data.* When possible, we would like to augment supervised training of some function by additional unlabeled data that may be available. One probabilistic approach to this is Expectation Maximization (EM) where the algorithm iteratively estimates the values of any unobserved variables (e.g., the labels), then reestimates the parameters of the probabilistic graphical model. EM has the attractive property that it converges to a local maximum in the expected likelihood of the full data. A second, very different approach, when learning a function $f : X \rightarrow Y$ with a combination of labeled and unlabeled examples, is to use the unlabeled examples to estimate $P(X)$, so that each labeled example can be reweighted by its probability of occurring. A third, again very different approach is to make use of unlabeled data when learning multiple functions jointly. For example, co-training algorithms learn two or more functions based on distinct subsets of the features in X, to predict the same Y label, then train these distinct functions to both fit the correct labels on labeled training examples, and to also agree on their predictions for unlabeled examples. Many other approaches are possible as well, including approaches that learn many distinct functions whose predictions are coupled by a variety of constraints that can be tested using unlabeled examples.

- *Learning of representations.* Although much of machine learning involves learning *functions*, it also involves learning useful *representations* of the input data. For example, given a sample S of data from some d-dimensional Euclidean space $X = \Re^d$ we might wish to learn a more compact representation of the data in a lower dimensional space. One approach is to

train a model that maps data points from S into a lower dimensional space in a way that allows reconstructing the original d-dimensional data as accurately as possible. This can be accomplished via several methods, including training a neural network with a low dimensional hidden layer to output the same data point it is given as input, factoring the original data matrix S into the product of two other matrices that share a lower dimensional inner dimension. Principle Components Analysis (PCA) learns a linear re-representations of the input data in terms of an orthogonal basis whose top k dimensions give the best possible linear reconstruction of the original data. Independent Components Analysis (ICA) also learns a linear re-representations of the input data, but one where the coordinates of the transformed data are statistically independent. Another approach, this one probabilistic, is to represent the data as being generated by a probability distribution conditioned on hidden variables whose values constitute the new representation, as in mixture of Gaussians models, or a mixture of latent topics using Latent Dirichlet Allocation. In addition to these unsupervised approaches, supervised methods can be employed to learn re-representations of the data useful for specific classification or regression problems, rather than to minimize the reconstruction error of the original data. Supervised training of neural networks with hidden layers performs exactly this function, learning re-representations of the input data at its hidden layers, where the hidden layer representations are optimized to maximize the accuracy of neural network outputs.

- *Kernel methods.* Kernel methods allow us to learn highly non-linear functions, where the non-linear function corresponds to a linear function in some higher dimensional space. To be precise, a kernel function $k : X_1 \times X_1 \rightarrow \Re$ defined over some vector space X_1 calculates the dot (inner) product of two vectors from X_1, after they are projected into a second vector space X_2 via some function $\Phi : X_1 \rightarrow X_2$. In other words, $k(\mathbf{a}, \mathbf{b}) = \langle \Phi(\mathbf{a}), \Phi(\mathbf{b}) \rangle$ where $\mathbf{a}$ and $\mathbf{b}$ are any vectors in X_1. The significance of kernel methods is (1) they often allow using convex linear learning methods to learn non-linear functions, and (2) the computations they perform are efficient because their calculations of dot products in the higher dimensional space are performed efficiently by applying the kernel function instead in the original lower dimensional space. Kernel methods can be used for non-linear regression, and for non-linear classifiers such as Support Vector Machines. Kernel methods are also used to extend linear algorithms such as Principle Components Analysis (PCA), and Canonical Correlation Analysis (CCA) to handle non-linear functions.

- *Distant rewards and reinforcement learning.* In standard supervised function approximation we wish to learning some target function $f : X \rightarrow Y$ from labeled training examples corresponding to input-output pairs $\langle x^{(i)}, y^{(i)} \rangle$ of f. However, in some applications, such as learning to play Chess or Go, the training experience can be much different. In such games we wish to learn a function from the current game state to the move we should make. However, the training feedback signal is not provided until the game ends, when we discover whether we have won or lost. To handle this kind of delayed feedback, reinforcement learning algorithms can be used, which are based on a probabilistic decision theoretic formalism called Markov Decision Processes. In cases where the learner can simulate the effects of each action (e.g., of any game move), algorithms such as value iteration can be used, which employ a dynamic programming approach to learn an evaluation function $V(s)$ defined over board states s. In the more difficult case where the learner is not able to simulate the effects of its actions (e.g., a car driving on a slippery road), algorithms such as Q-learning can be used to acquire a similar evaluation function $Q(s,a)$ defined over state-action pairs. One key advantage of Q-learning is that when the agent finds itself in state s it can choose the best action simply by finding the action a that maximizes $Q(s,a)$ even if it cannot predict accurately the next state that will result in taking this action. In contrast, to choose an action from state s using $V(s)$, the system must perform a look-ahead search over states resulting from candidate actions, which requires the ability to internally simulate action outcomes.

2.3 Where is Machine Learning Headed Next?

Nobody knows the answer to this question, of course, but my own opinion is that we are just at the beginning of a decades-long set of advances that will change the way we think about machine learning, computer science, and human learning. Surely we will see more research in the near term in the directions that machine learning is already headed – more research on data-intensive learning, deep neural networks, probabilistic methods, etc. But I think we will also see advances in other, very different directions. Below are some examples – advances that might happen, are not certain to happen, but if they happen then they are likely to have a major impact on the field of machine learning and on the world.

- *Machine learning from user instruction.* Today, machine learning algorithms are heavily statistical. But human learning includes other approaches as well, including learning from instruction. Think of the intelligent assistant in your phone, and think about the conversational interactions you have with it today. Today, they all involve you commanding the phone to perform

one of its predefined capabilities (e.g., tell you the weather forecast, or how to drive to the movie theater). What if you could use that conversation to teach the phone to do new things (e.g., whenever it snows at night, wake me up 30 minutes earlier, because I don't want to be late getting to work.). If phones could be taught in this way by users, we would suddenly find that we have billions of programmers - only they would be using natural language to program their phones instead of learning the language of computers.

- *Machine learning by reading.* Today, the world wide web contains much of human knowledge, but mostly in natural language which is not understood by computers. However, significant advances are now occurring in many areas of natural language processing (e.g., machine translation). If natural language understanding reaches a high enough level of competence, we might suddenly see that learning by reading becomes a dominant component of how machines learn. Machines would, unlike us humans, be able to read the entire web, and they would suddenly be better read than you and I by a factor of several million.

- *Machine learning agents instead of learning single functions.* Most machine learning today involves supervised learning of a single target function from input-output examples of that function. More and more, we are seeing AI systems that require many inter-related functions. For example, self-driving vehicles require a function to choose steering, braking, and acceleration actions, but also require functions that spot road obstacles, that prediction motions of nearby vehicles, and many others. The key lesson from our NELL research, which couples the training of thousands of functions, is that the learning problems become easier (and can take better advantage of unlabeled data) when the agent is forced to jointly learn many inter-related functions. I expect that as the field pursues learning in the context of more robot and softbot agents which require learning multiple inter-related functions, we may see sudden improvements in learning competence that make us wonder in retrospect why we spent so much time on the more difficult problem of learning single functions in isolation.